

35th International Conference on Concurrency Theory


CONCUR 2024, September 9–13, 2024, Calgary, Canada

Edited by

Rupak Majumdar
Alexandra Silva



Editors

Rupak Majumdar 

Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
rupak@mpi-sws.org

Alexandra Silva 

Cornell University, Ithaca, NY, USA
alexandra.silva@gmail.com

ACM Classification 2012

Theory of computation → Concurrency; Theory of computation → Categorical semantics; Theory of computation → Process calculi; Theory of computation → Markov decision processes; Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking; Theory of computation → Automata over infinite objects

ISBN 978-3-95977-339-3

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-339-3>.

Publication date

September, 2024

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): <https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.CONCUR.2024.0

ISBN 978-3-95977-339-3

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

LIPICs – Leibniz International Proceedings in Informatics

LIPICs is a series of high-quality conference proceedings across all fields in informatics. LIPICs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Luca Aceto (Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University, Brno, CZ)
- Meena Mahajan (*Chair*, Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Rupak Majumdar and Alexandra Silva</i>	0:ix
Committees	
.....	0:xi

Invited Talks

Constrained Horn Clauses for Program Verification and Synthesis	
<i>Arie Gurfinkel</i>	1:1–1:1
Principles of Persistent Programming	
<i>Azalea Raad</i>	2:1–2:1
Verifying Concurrent Search Structures	
<i>Thomas Wies</i>	3:1–3:1

Regular Papers

Centralized vs Decentralized Monitors for Hyperproperties	
<i>Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Daniele Gorla, and Jana Wagemaker</i>	4:1–4:19
MITL Model Checking via Generalized Timed Automata and a New Liveness Algorithm	
<i>S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan</i>	5:1–5:19
Causally Deterministic Markov Decision Processes	
<i>S. Akshay, Tobias Meggendorfer, and P. S. Thiagarajan</i>	6:1–6:22
Fairness and Consensus in an Asynchronous Opinion Model for Social Networks	
<i>Jesús Aranda, Sebastián Betancourt, Juan Fco. Díaz, and Frank Valencia</i>	7:1–7:17
Bidding Games with Charging	
<i>Guy Avni, Ehsan Kafshdar Goharshady, Thomas A. Henzinger, and Kaushik Mallik</i>	8:1–8:17
Risk-Averse Optimization of Total Rewards in Markovian Models Using Deviation Measures	
<i>Christel Baier, Jakob Piribauer, and Maximilian Starke</i>	9:1–9:20
Passive Learning of Regular Data Languages in Polynomial Time and Data	
<i>Mrudula Balachander, Emmanuel Filiot, and Raffaella Gentilini</i>	10:1–10:21
Left-Linear Rewriting in Adhesive Categories	
<i>Paolo Baldan, Davide Castelnovo, Andrea Corradini, and Fabio Gadducci</i>	11:1–11:24
History-Determinism vs Fair Simulation	
<i>Udi Boker, Thomas A. Henzinger, Karoliina Lehtinen, and Aditya Prakash</i>	12:1–12:16

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva



Leibniz International Proceedings in Informatics

LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Power of Counting Steps in Quantitative Games <i>Sougata Bose, Rasmus Ibsen-Jensen, David Purser, Patrick Totzke, and Pierre Vandenhove</i>	13:1–13:18
As Soon as Possible but Rationally <i>Véronique Bruyère, Christophe Grandmont, and Jean-François Raskin</i>	14:1–14:20
RobTL: Robustness Temporal Logic for CPS <i>Valentina Castiglioni, Michele Loreti, and Simone Tini</i>	15:1–15:23
Effect Semantics for Quantum Process Calculi <i>Lorenzo Ceragioli, Fabio Gadducci, Giuseppe Lomurno, and Gabriele Tedeschi</i> ...	16:1–16:22
Invariants for One-Counter Automata with Disequality Tests <i>Dmitry Chistikov, Jérôme Leroux, Henry Sinclair-Banks, and Nicolas Waldburger</i>	17:1–17:21
Weighted Basic Parallel Processes and Combinatorial Enumeration <i>Lorenzo Clemente</i>	18:1–18:22
Computing Inductive Invariants of Regular Abstraction Frameworks <i>Philipp Czerner, Javier Esparza, Valentin Krasotin, and Christoph Welzel-Mohr</i> .	19:1–19:18
Behavioural Metrics: Compositionality of the Kantorovich Lifting and an Application to Up-To Techniques <i>Keri D'Angelo, Sebastian Gurke, Johanna Maria Kirss, Barbara König, Matina Najafi, Wojciech Różowski, and Paul Wild</i>	20:1–20:19
Reversible Transducers over Infinite Words <i>Luc Dartois, Paul Gastin, Loïc Germerie Guizouarn, R. Govind, and Shankaranarayanan Krishna</i>	21:1–21:22
An Automata-Based Approach for Synchronizable Mailbox Communication <i>Romain Delpy, Anca Muscholl, and Grégoire Sutre</i>	22:1–22:19
Regular Games with Imperfect Information Are Not That Regular <i>Laurent Doyen and Thomas Soullard</i>	23:1–23:19
Validity of Contextual Formulas <i>Javier Esparza and Rubén Rubio</i>	24:1–24:17
A Unifying Categorical View of Nondeterministic Iteration and Tests <i>Sergey Goncharov and Tarmo Uustalu</i>	25:1–25:22
Phase-Bounded Broadcast Networks over Topologies of Communication <i>Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder</i>	26:1–26:16
Inapproximability in Weighted Timed Games <i>Quentin Guilmant and Joël Ouaknine</i>	27:1–27:15
Faster and Smaller Solutions of Obliging Games <i>Daniel Hausmann and Nir Piterman</i>	28:1–28:19
Strategic Dominance: A New Preorder for Nondeterministic Processes <i>Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç</i>	29:1–29:20
Around Classical and Intuitionistic Linear Processes <i>Juan C. Jaramillo, Dan Frumin, and Jorge A. Pérez</i>	30:1–30:19

Bi-Reachability in Petri Nets with Data <i>Lukasz Kamiński and Sławomir Lasota</i>	31:1–31:20
Minimising the Probabilistic Bisimilarity Distance <i>Stefan Kiefer and Qiyi Tang</i>	32:1–32:18
Automating Memory Model Metatheory with Intersections <i>Aristotelis Koutsouridis, Michalis Kokologiannakis, and Viktor Vafeiadis</i>	33:1–33:16
On Continuous Pushdown VASS in One Dimension <i>Guillermo A. Pérez and Shrisha Rao</i>	34:1–34:20
Nominal Tree Automata with Name Allocation <i>Simon Prucker and Lutz Schröder</i>	35:1–35:17
Branching Bisimilarity for Processes with Time-Outs <i>Gaspard Reghem and Rob J. van Glabbeek</i>	36:1–36:22
A Spectrum of Approximate Probabilistic Bisimulations <i>Timm Spork, Christel Baier, Joost-Pieter Katoen, Jakob Piribauer, and Tim Quatmann</i>	37:1–37:19
Progress, Justness and Fairness in Modal μ -Calculus Formulae <i>Myrthe S. C. Spronck, Bas Luttik, and Tim A. C. Willemse</i>	38:1–38:22
Coinductive Techniques for Checking Satisfiability of Generalized Nested Conditions <i>Lara Stoltenow, Barbara König, Sven Schneider, Andrea Corradini, Leen Lambers, and Fernando Orejas</i>	39:1–39:20
A PSPACE Algorithm for Almost-Sure Rabin Objectives in Multi-Environment MDPs <i>Marnix Suilen, Marck van der Vegt, and Sebastian Junges</i>	40:1–40:17

■ Preface

This volume contains the contributions accepted for the 35th International Conference on Concurrency Theory (CONCUR), held in 2024. CONCUR serves as an annual scientific forum for researchers, developers, and students working to expand the field of concurrency theory and its applications. CONCUR 2024 was organized in Calgary, Canada between 9 and 13 September, 2024, as part of CONFEST 2024. Along with CONCUR, CONFEST also featured the QEST+FORMATS conference, as well as several workshops.

For CONCUR 2024, we received 80 submissions and accepted 37 for presentation at the conference. We are grateful for the hard work of our program committee as well as the many external experts who produced 240 reviews and engaged in lively discussions. We wish to thank the authors for submitting their work to CONCUR, and we congratulate the authors of all accepted papers. We look forward to a scientifically interesting conference in September.

We would like to thank the CONCUR invited speakers, Arie Gurfinkel (University of Waterloo), Azalea Raad (Imperial College London), and Thomas Wies (New York University), as well as the CONFEST Unifying Speaker, Corina Pasareanu (NASA Ames/Carnegie Mellon University), and the QEST+FORMATS invited speaker, Mor Harchol-Balter (Carnegie Mellon University).

In 2020, CONCUR and the IFIP WG 1.8 on Concurrency Theory initiated the test-of-time award to honor significant contributions to Concurrency Theory that were published at CONCUR. This year’s award goes to Stephen D. Brookes and Peter W. O’Hearn, for their papers “A semantics for concurrent separation logic” and “Resources, concurrency and local reasoning,” respectively, both published in CONCUR 2004.

The proceedings of CONCUR 2024 are freely available through the LIPIcs series. We thank Diwakar Krishnamurthy, the general chair, as well as the rest of the organizing team, and the University of Calgary for their assistance in organizing CONFEST 2024. We look forward to welcoming you in Calgary!



■ Committees

Program committee

Alessandro Abate	Oxford
Alexandra Silva (co-chair)	Cornell University
Andreas Pavlogiannis	Aarhus
Antonín Kucera	Masaryk University
Ashutosh Trivedi	UC Boulder
Barbara König	University of Duisburg-Essen
Benjamin Kaminski	Saarland University
Cinzia Di Giusto	Univ of Nice
Colin Gordon	Drexel University
Constantin Enea	Ecole Polytechnique
Dan Ghica	Huawei
Dana Fisman	Ben-Gurion University
Daniele Gorla	University of Rome La Sapienza
Davide Sangiorgi	University of Bologna
Emmanuele d’Oswaldo	University of Konstanz
Frits Vaandrager	Radboud University
Guillermo Pérez	University of Antwerp
Ilaria Castellani	INRIA Sophia-Antipolis
James Worrell	Oxford
Jana Wagemaker	Reykjavik
Kirstin Peters	Augsburg University
Klaus v. Gleissenthall	Vrije Uni Amsterdam
Michele Boreale	University of Florence
Nadia Labai	AWS
Orna Kupferman	Hebrew University
Roland Meyer	TU Braunschweig
Rupak Majumdar (co-chair)	Max Planck Institute for Software Systems
S Akshay	Indian Institute of Technology Bombay
Sadegh Soudjani	Max Planck Institute for Software Systems
Sebastian Junges	Radboud University & Nijmegen
Stefan Milius	FAU Erlangen-Nürnberg
Subhajit Roy	IIT Kanpur
Thejaswini K.S.	ISTA
Umang Mathur	National University of Singapore
Valeria Vignudelli	ENS Lyon
Yu-Fang Chen	Academica Sinica Taiwan

Steering committee

Luca Aceto	Reykjavik University
Christel Baier	Technical University Dresden
Pedro D’Argenio	Universidad Nacional de Córdoba
Wan Fokkink (chair)	Vrije University Amsterdam
Catuscia Palamidessi	Ecole Polytechnique
Jiri Srba	Aalborg University

35th International Conference on Concurrency Theory (CONCUR 2024).




Editors: Rupak Majumdar and Alexandra Silva



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Constrained Horn Clauses for Program Verification and Synthesis

Arie Gurfinkel   

University of Waterloo, ON, Canada

Abstract

First Order Logic (FOL) is a powerful formalism that naturally captures many interesting decision and optimization problems. In recent years, there has been a tremendous progress in automated logic reasoning tools, such as Boolean SATisfiability Solvers and Satisfiability Modulo Theory solvers. This enabled the use of logic and logic solvers as a universal solution to many problems in Computer Science, in general, and in Program Analysis, in particular. Most new program analysis techniques formalize the desired analysis task in a fragment of FOL, and delegate the analysis to a SAT or an SMT solver.

In this talk, we focus on a fragment of FOL called Constrained Horn Clauses (CHC) and the CHC solver SPACER. CHCs arise in many applications of automated verification. They naturally capture such problems as discovery and verification of inductive invariants; Model Checking of safety properties of finite- and infinite-state systems; safety verification of push-down systems (and their extensions); modular verification of distributed and parameterized systems; type inference, and many others.

Using CHC separates the process of developing a proof methodology (also known as generation of Verification Condition (VC)) from the algorithmic details of deciding whether the VC is correct. Such a flexible design simplifies supporting multiple proof methodologies, multiple languages, and multiple verification tasks with a single framework, without sacrificing performance and scalability.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Constrained Horn Clauses

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.1

Category Invited Talk



© Arie Gurfinkel;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 1; pp. 1:1–1:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Principles of Persistent Programming

Azalea Raad  

Imperial College London, UK

Abstract

Persistent programming is the art of developing programs that operate on persistent (non-volatile) states that survive program termination, be it planned or abrupt (e.g. due to a power failure). Persistent programming poses several important challenges: 1) persistent systems have complex – and often unspecified – semantics in that operations do not generally persist in their execution order; 2) software bugs in persistent settings can lead to permanent data corruption; and 3) traditional testing techniques are inapplicable in persistent settings. Can formal methods come to the rescue?

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Persistent Programming

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.2

Category Invited Talk

Funding UKRI fellowship MR/V024299/1, EPSRC grant EP/X037029/1 and VeTSS



© Azalea Raad;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Verifying Concurrent Search Structures

Thomas Wies  

New York University, New York, NY, US

Abstract

Search structures support the fundamental data storage primitives on key-value pairs: insert a pair, delete by key, search by key, and update the value associated with a key. Concurrent search structures are parallel algorithms to speed access to search structures on multicore and distributed servers. For these data structures to be efficient, the underlying parallel algorithms need to perform fine-grained synchronization between threads. This makes them notoriously difficult to design and implement correctly. Indeed, bugs are routinely found both in actual implementations and in the designs proposed by experts in peer-reviewed publications. Often, these bugs elude testing-based quality control due to complex thread interactions that only manifest after deployment, and under conditions that are difficult to replicate. Given the critical role that concurrent search structures play in today's software infrastructure, it is therefore highly desirable to verify their correctness using formal methods, preferably in an automated fashion.

In this talk, I will present a framework for obtaining linearizability proofs for concurrent search structures that are modular, reusable, and amenable to automation. The framework takes advantage of recent advances in local reasoning techniques based on concurrent separation logic. I will provide an overview of these techniques and discuss their use for verifying both lock-based and lock-free concurrent search structures such as concurrent (skip)lists, hash structures, binary search trees, B trees, and log-structured merge trees.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Concurrent search structures

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.3

Category Invited Talk

Funding This work is funded in parts by the United States National Science Foundation under grants CCF-2304758 and CCF-1815633. Further funding came from an Amazon Research Award Fall 2021. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not reflect the views of Amazon.

Acknowledgements This talk is based on joint work with many of my students and colleagues, including Siddharth Krishna, Roland Meyer, Nisarg Patel, Dennis Shasha, Alexander Summers, and Sebastian Wolff.



© Thomas Wies;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 3; pp. 3:1–3:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Centralized vs Decentralized Monitors for Hyperproperties

Luca Aceto  


Dept. of Computer Science, Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy

Antonis Achilleos  

Dept. of Computer Science, Reykjavik University, Iceland

Elli Anastasiadi  

Uppsala University, Sweden

Adrian Francalanza  

University of Malta, Malta

Daniele Gorla  

Dept. of Computer Science, “Sapienza” University of Rome, Italy

Jana Wagemaker  

Dept. of Computer Science, Reykjavik University, Iceland

Abstract

This paper focuses on the runtime verification of hyperproperties expressed in Hyper-*rech*HML, an expressive yet simple logic for describing properties of sets of traces. To this end, we consider a simple language of monitors that observe sets of system executions and report verdicts w.r.t. a given Hyper-*rech*HML formula. We first employ a unique omniscient monitor that centrally observes all system traces. Since centralised monitors are not ideal for distributed settings, we also provide a language for decentralized monitors, where each trace has a dedicated monitor; these monitors yield a unique verdict by communicating their observations to one another. For both the centralized and the decentralized settings, we provide a synthesis procedure that, given a formula, yields a monitor that is correct (i.e., sound and violation complete). A key step in proving the correctness of the synthesis for decentralized monitors is a result showing that, for each formula, the synthesized centralized monitor and its corresponding decentralized one are weakly bisimilar for a suitable notion of weak bisimulation.

2012 ACM Subject Classification Theory of computation → Operational semantics; Theory of computation → Modal and temporal logics; Theory of computation → Logic and verification

Keywords and phrases Runtime Verification, hyperlogics, decentralization

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.4

Related Version *Full Version*: <https://arxiv.org/abs/2405.12882> [2]

Funding This work has been supported by the project “Mode(l)s of Verification and Monitorability” (MoVeMent) (grant No 217987) of the Icelandic Research Fund.

Elli Anastasiadi: Elli Anastasiadi’s research has been supported by grant VR 2020-04430 of the Swedish Research Council.

1 Introduction

Runtime verification (RV) [12] is a verification technique that observes system executions to determine whether some given specification is satisfied or violated. This runtime analysis is usually conducted by a computational entity called a *monitor* [33]. RV is a lightweight verification technique that is carried out as the system under observation executes, thereby



© Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Daniele Gorla, and Jana Wagemaker;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 4; pp. 4:1–4:19

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

avoiding scalability issues caused by the state-explosion problem, as is the case for model checking. Recently, RV has been extended to parallel set-ups [17, 24, 45], and a large body of work in that setting aims to verify *hyperproperties* at runtime [1, 18, 19, 27, 30].

Hyperproperties [27] are sets of *hypertraces*, *i.e.* sets of traces that may be seen as describing different system executions or the contributions of different sequential processes to a system execution. As argued in [22], many properties of concurrent and distributed systems can be viewed as hyperproperties. When verifying hyperproperties at runtime, several traces (*i.e.* several execution sequences) can be observed instead of just one, possibly at the same time. Several extensions of temporal logics, such as HyperLTL, HyperCTL* [26], Hyper²LTL [14], have been defined to express hyperproperties. Extensions of standard logics to hyper properties also include variations of the μ -calculus, such as [1], setting the basis for the logic used in this paper, and [36], which studies an asynchronous semantics.

Since they were proposed by Clarkson and Schneider in [27], hyperproperties have become a fundamental, trace-based formalism for expressing security and privacy properties, verified using static and dynamic techniques [10, 14, 15, 18, 22, 23, 25, 30] implemented in a variety of tools [13, 15, 29]. There is a large body of work, such as [10, 23, 37], detailing several algorithms for monitoring (fragments of) hyperlogics under different assumptions and providing several correctness guarantees. However, these proposals either construct a centralized monitoring algorithm that has access to all traces in the observed hypertrace, or verify single trace properties, over a distributed set-up¹. Having an omniscient monitor simplifies the runtime analysis since the monitoring algorithm can compare all traces as needed by simply accessing different parts of its local memory. But this power comes with drawbacks. For starters, centralized monitors are unrealistic for distributed systems, where trace analysis is typically localised to network nodes so as to minimize communication across locations. Moreover, centralized monitors create single points of failure during verification [8]. Furthermore, it can be problematic to store all the traces locally, especially in light of the wide availability of multi-core systems. The goal of the decentralized monitor synthesis from logical specifications presented in this paper is to permit distributed monitor choreographies with *local* trace views whose components communicate in order to verify *global* properties (such as hyperproperties). Decentralized monitors have been shown to avoid high contentions leading to vastly improved scalability [8]. They also offer better privacy guarantees whenever they are stationed locally at the nodes where the respective traces are generated [35, 39]. To the best of our knowledge, such a message-passing monitoring set-up has never been studied for the purpose of verifying hyperproperties so far.

In this paper, we study procedures for the *automated synthesis of centralized and decentralized monitors* from hyperproperties described in the logic Hyper-recHML [1]. This logic extends the linear-time [51] μ -calculus [40] (also known as Hennessy-Milner logic with recursion [44]) with constructs to describe properties of hypertraces inspired by the work on HyperLTL (namely variables ranging over traces, modal operators parametrized by trace variables, matching/mismatching between trace variables, and existential and universal quantification over them). Hyper-recHML can describe hyperproperties not expressible in HyperLTL or HyperCTL*, such as properties that speak about consensus (see Example 2) and periodicity (see Example 3). Furthermore, Hyper-recHML supports a general, syntax-driven monitor synthesis that can handle both the aforementioned hyperproperties, at least in the centralized case (see also the discussion in Section 5).

¹ See e.g. [20, 21, 31, 35] for distributed monitoring algorithms for classic trace-based logics.

In both the centralized and decentralized set-ups, we work in the parallel model [30], where a fixed number of system executions is processed in parallel by monitors in an online fashion. We specify monitors using a process-algebraic formalism that builds on the one presented in [5, 34] to define a class of monitors called regular. Such monitors are easy to describe, resemble (alternating) automata, and have sufficient expressive power to provide standard monitoring guarantees. Moreover, their algebraic structure supports the compositional definition of their operational semantics and monitor synthesis procedures from formulas, building on previous work relating algebraic process calculi with RV [6, 9, 16, 32, 33, 38, 42, 43].

In the centralized case, for each formula in the fragment of Hyper-*rec*HML limited to greatest-fixed-point operators, our synthesis procedure yields a monolithic monitor that has access to all the traces in an observed hypertrace. However, in order to synthesize decentralized monitors for a sufficiently expressive fragment of the logic, it is necessary to extend the monitor capabilities with communication, as shown already in [1]. For instance, to monitor for the property “If there is a trace where event a occurs, then there exists another trace where event b does not occur thereafter”, monitors observing different traces need to communicate to record that event a occurred in some trace at some point and that there is some trace where b does not occur from that point onwards. Allowing monitors to send and receive messages significantly complicates their operational semantics (see Section 4), the monitor synthesis procedure (see Section 4.2), and all consequent proofs. The operational semantics for communicating monitors is one of the main contributions of the paper since its design is crucial to obtain the correctness guarantees provided by the synthesis procedure for decentralized monitors. In particular, the semantics of decentralized monitors and their synthesis from formulas have to be designed carefully to ensure that monitors are reactive (they are always ready to process any system event) and input-enabled (they can always receive any input from other monitors in their environment), properties that are desirable in any decentralized RV set-up.

We show that both *the centralized and the decentralized monitor synthesis procedures are correct*. More precisely, the monitors synthesized from formulas are *sound* and *violation-complete*, meaning that (1) if the monitor synthesized from a formula φ reports a positive (resp., negative) verdict when observing a hypertrace T , then T does (resp., does not) satisfy φ , and (2) if T does not satisfy φ , then its associated monitor will report a negative verdict when observing T (see Theorems 7 and 8, and Corollaries 10 and 11). The proof of correctness in the decentralized case is considerably more technical than the corresponding proof in the centralized setting, due to the intricate communication semantics. To address the resulting technical challenges, we develop a proof strategy where we prove the correctness of the decentralized monitor synthesis procedure using the centralized one as a yardstick.

This methodology is one of the key contributions we offer in this study. More precisely, in Section 4.1 *we identify six properties of a decentralized monitor synthesis that make it “principled”* (see Definition 13) and we show that, when a decentralized monitor synthesis is principled, the centralized and decentralized monitors synthesized from a formula are related by a suitable notion of weak bisimulation (Theorem 14). Apart from supporting the definition of decentralized monitor synthesis procedures, this result allows us to reduce the correctness of our decentralized monitor synthesis to that of the centralized one, which can in turn drive the definition of further synthesis procedures in future work. We also conjecture that our methodology provides a path to proving similar results for other models of communicating monitors independent of the monitoring strategy. In summary, our contributions are the following:

4:4 Centralized vs Decentralized Monitors for Hyperproperties

- a framework for monitoring hyperproperties by a central monitor that has access to all locations (Section 3) and a decentralized monitoring set-up for hyperproperties, with monitors that communicate (Section 4);
- a synthesis function that returns a correct centralized monitor for every formula without least fixed points (Section 3);
- a synthesis function that returns a correct (decentralized) choreography of communicating monitors for every formula without least fixed points that has no location quantifier within a fixed point operator (Section 4); and
- a methodology to prove the correctness of a synthesis of communicating monitors, by establishing a list of desirable properties and relating the behavior of the decentralized monitors to that of the corresponding centralized monitor (Definition 13 and Theorem 14). Omitted proofs, due to space constraints, can be found in [2].

2 The Model and the Logic

Let Act be a finite set of actions with at least two elements², ranged over by a, b ; the set of (infinite) traces over Act is $\text{Trc} = \text{Act}^\omega$, ranged over by t . Given a finite and non-empty set of locations \mathcal{L} ranged over by ℓ , a hypertrace T on \mathcal{L} is a function from \mathcal{L} to Trc ; the set of hypertraces on \mathcal{L} is denoted by $\text{HTrc}_{\mathcal{L}}$. \mathcal{L} and Act are fixed throughout this paper. A hypertrace describes a (distributed) system with $|\mathcal{L}|$ users, and every user is located at a unique location chosen from \mathcal{L} . A system behavior is captured by a hypertrace T on \mathcal{L} , mapping every user to the trace they perform.

For $t, t' \in \text{Trc}$, we write $t \xrightarrow{a} t'$ whenever $t = at'$. Let $A : \mathcal{L} \rightarrow \text{Act}$; for $T, T' \in \text{HTrc}_{\mathcal{L}}$, we write $T \xrightarrow{A} T'$ whenever $T(\ell) \xrightarrow{A(\ell)} T'(\ell)$, for every $\ell \in \mathcal{L}$. Notice that, for each T , there is a *unique* pair A and T' such that $T \xrightarrow{A} T'$: more precisely, for every $\ell \in \mathcal{L}$, we have that $A(\ell) = a$ and $T'(\ell) = t'$, whenever $T(\ell) = at'$. We denote the A and T' just defined by $hd(T)$ and $tl(T)$ respectively. For a partial function $f : D \rightarrow E$ (where D and E are sets ranged over by d and e , respectively), we denote by $\text{dom}(f)$ the set $\{d \in D \mid f(d) \text{ is defined}\}$ and by $\text{rng}(f)$ the set $\{e \in E \mid \exists d \in \text{dom}(f). f(d) = e\}$. Notation $f[d \mapsto e]$ denotes the (partial) function mapping d to e and behaving like f otherwise.

2.1 The Logic Hyper-recHML

We consider Hyper-recHML as the logic to specify *hyperproperties*. We assume two disjoint and countably infinite sets Π and V of *location variables* and *recursion variables*, ranged over by π and x , respectively. Formulas of Hyper-recHML are constructed as follows:

$$\varphi ::= \text{tt} \mid \text{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \max x. \varphi \mid \min x. \varphi \mid x \mid \exists \pi. \varphi \mid \forall \pi. \varphi \mid \pi = \pi \mid \pi \neq \pi \mid [a_\pi] \varphi \mid \langle a_\pi \rangle \varphi$$

Apart from the basic boolean constructs, we include the greatest and least fixed-point operators to describe unbounded and/or infinite behaviors in a finitary manner,³ existential/universal quantifiers and equality/inequality tests on location variables, and the usual Hennessy-Milner modalities where $[a_\pi]$ stands for “necessarily after a at the location bound to π ”, and $\langle a_\pi \rangle$ denotes “possibly after a at the location bound to π ”. A formula is said to be *guarded* if every recursion variable appears within the scope of a modality

² When Act is a singleton, every property in the logic becomes equivalent to true or false.

³ In LTL, this behavior is captured by the ‘Until’ and ‘Release’ operators, but these are less expressive than fixed-points; see [7].

■ **Table 1** The semantics of Hyper-recHML.

$\llbracket \text{tt} \rrbracket_\sigma^\rho = \text{HTrc}_{\mathcal{L}}$	$\llbracket \text{ff} \rrbracket_\sigma^\rho = \emptyset$	$\llbracket x \rrbracket_\sigma^\rho = \rho(x)$
$\llbracket \varphi \wedge \varphi' \rrbracket_\sigma^\rho = \llbracket \varphi \rrbracket_\sigma^\rho \cap \llbracket \varphi' \rrbracket_\sigma^\rho$		$\llbracket \varphi \vee \varphi' \rrbracket_\sigma^\rho = \llbracket \varphi \rrbracket_\sigma^\rho \cup \llbracket \varphi' \rrbracket_\sigma^\rho$
$\llbracket \max x. \psi \rrbracket_\sigma^\rho = \bigcup \{S \mid S \subseteq \llbracket \psi \rrbracket_\sigma^{\rho[x \mapsto S]}\}$		$\llbracket \min x. \psi \rrbracket_\sigma^\rho = \bigcap \{S \mid S \supseteq \llbracket \psi \rrbracket_\sigma^{\rho[x \mapsto S]}\}$
$\llbracket \exists \pi. \varphi \rrbracket_\sigma^\rho = \bigcup_{\ell \in \mathcal{L}} \llbracket \varphi \rrbracket_{\sigma[\pi \mapsto \ell]}^\rho$		$\llbracket \forall \pi. \varphi \rrbracket_\sigma^\rho = \bigcap_{\ell \in \mathcal{L}} \llbracket \varphi \rrbracket_{\sigma[\pi \mapsto \ell]}^\rho$
$\llbracket \pi = \pi' \rrbracket_\sigma^\rho = \begin{cases} \text{HTrc}_{\mathcal{L}} & \text{if } \sigma(\pi) = \sigma(\pi') \\ \emptyset & \text{otherwise} \end{cases}$		$\llbracket \pi \neq \pi' \rrbracket_\sigma^\rho = \begin{cases} \text{HTrc}_{\mathcal{L}} & \text{if } \sigma(\pi) \neq \sigma(\pi') \\ \emptyset & \text{otherwise} \end{cases}$
$\llbracket [a_\pi] \varphi \rrbracket_\sigma^\rho = \{T \mid \text{hd}(T)(\sigma(\pi)) = a \text{ implies } \text{tl}(T) \in \llbracket \varphi \rrbracket_\sigma^\rho\}$		
$\llbracket \langle a_\pi \rangle \varphi \rrbracket_\sigma^\rho = \{T \mid \text{hd}(T)(\sigma(\pi)) = a \wedge \text{tl}(T) \in \llbracket \varphi \rrbracket_\sigma^\rho\}$		

within its fixed-point binding. All formulas are assumed to be guarded (without loss of expressiveness [41]). We write $\text{FVloc}(\varphi)$ to denote the free location variables of φ , and $\text{FVrec}(\varphi)$ for the free recursion variables.

► **Remark 1.** We consider formulas where bound location variables are all pairwise distinct (and different from the free variables); hence, the formula $\forall \pi. [a_\pi] \exists \pi. \varphi$ denotes the formula $\forall \pi. [a_\pi] \exists \pi'. (\varphi\{\pi'/\pi\})$, where $\varphi\{\pi'/\pi\}$ stands for the capture-avoiding substitution of π' for π in φ . A similar notation for other kinds of substitutions is used throughout the paper. ◻

The semantics of a Hyper-recHML formula φ is defined over $\text{HTrc}_{\mathcal{L}}$ by exploiting two partial functions: $\rho: V \rightarrow 2^{\text{HTrc}_{\mathcal{L}}}$, which assigns a set of hypertraces on \mathcal{L} to all free recursion variables of φ , and $\sigma: \Pi \rightarrow \mathcal{L}$, which assigns a location to all free location variables of φ . In what follows, we tacitly assume that the free recursion and location variables in a formula φ are always included in $\text{dom}(\rho)$ and $\text{dom}(\sigma)$, respectively.

The semantics for formulas in Hyper-recHML is given through the function $\llbracket - \rrbracket_\sigma^\rho$ as shown in Table 1. A formula $\langle a_\pi \rangle \varphi$ holds true at hypertrace T if the trace in T at the location bound to π starts with an a and $\text{tl}(T)$ satisfies φ ; by contrast, a formula $[a_\pi] \varphi$ can also hold true if the trace in T at the location associated to π does not start with an a . Whenever φ is *closed* (i.e., without any free variable), the semantics is given by $\llbracket \varphi \rrbracket_\emptyset^\emptyset$, where \emptyset denotes the partial function with empty domain. Notationally, we shall simply write $\llbracket \varphi \rrbracket$ instead of $\llbracket \varphi \rrbracket_\emptyset^\emptyset$. We say that T satisfies the closed formula φ if $T \in \llbracket \varphi \rrbracket$.

► **Example 2.** For example, consider the set of actions $\{a, b\}$; then, the hyperproperty

$$\varphi_a = \forall \pi. \max x. (\langle b_\pi \rangle x \vee \exists \pi'. (\pi' \neq \pi \wedge \langle a_{\pi'} \rangle x)) \quad (1)$$

is a consensus-type property stating that, at every position of every trace, whenever there is an a there is another trace that also has a . Using the semantic definition of the logic, it is not hard to see that the hypertrace T_1 over the set of locations $\{\ell_1, \ell_2, \ell_3\}$ that maps ℓ_1 to a^ω , ℓ_2 to ba^ω and ℓ_3 to $(ba)^\omega$ does not satisfy the property φ_a : what breaks the property is the first position. On the other hand, the hypertrace T_2 that maps ℓ_1 to a^ω , ℓ_2 to $(ab)^\omega$ and ℓ_3 to $(ba)^\omega$ does satisfy φ_a because at each position there are two traces that exhibit an a . ◻

2.2 On the Expressiveness of Hyper-recHML

The logic Hyper-recHML adapts linear-time μ HML [44] to express properties of hypertraces, just as HyperLTL and HyperCTL* [26] are variations on LTL [47] and CTL* [28], respectively, interpreted over hypertraces. It is well known that μ HML is more expressive than LTL and CTL* [52]. It is, therefore, natural to wonder whether Hyper-recHML can express properties that cannot be described using HyperLTL and HyperCTL*.

We claim that the strictness of the inclusion of LTL in μ HML is preserved for their hyper-extensions. To justify our claim, we present two arguments to demonstrate that Hyper-recHML is more expressive than HyperLTL, which rely on classic results on the inexpressiveness of LTL, the embedding of LTL in μ HML, and the ability of Hyper-recHML to quantify over traces more liberally than HyperLTL.

First, we recall that Wolper showed in [52] that the property “event a occurs at all even positions in a trace” cannot be expressed in LTL (see [52, Corollary 4.2] that is based on Theorem 4.1 in that reference). We will refer to this property as φ_e , where “ e ” stands for even, and adapt it to a hypertrace setting.

► **Example 3.** Let φ_{h_e} be the hyperproperty on the set of actions $\{a, b\}$ that results from adding an existential trace quantifier $\exists\pi$ at the beginning of φ_e , and replacing all modalities with π -indexed ones:

$$\varphi_{h_e} = \exists\pi. \max x. ([a_\pi]\langle a_\pi \rangle x \wedge [b_\pi]\langle a_\pi \rangle x) \quad (2)$$

This is a liveness property that describes the periodicity of events; when evaluated over singleton hypertraces, it coincides with the evaluation of φ_e . \lrcorner

The hyperproperty φ_{h_e} defined above can be used to prove the following result.

► **Proposition 4.** *Hyper-recHML is more expressive than HyperLTL.*

The second witness to the fact that Hyper-recHML is more expressive than HyperLTL is the possibility to use quantifiers in any part of a formula. For example, the hyperproperty φ_a defined in (1) can potentially spawn an unbounded number of quantifiers, by unfolding the recursion when encountering a events.

► **Proposition 5.** *Hyper-recHML is more expressive than HyperCTL*.*

We shall see later on that part of this additional expressiveness of Hyper-recHML is present in the fragments for which we synthesize monitors.

3 Centralized Monitoring

The set of centralized monitors CMon is given by the following grammar:

$$\text{CMon} \ni m ::= \text{yes} \mid \text{no} \mid \text{end} \mid a_\ell.m \mid m + m \mid m \oplus m \mid m \otimes m \mid \text{rec } x.m \mid x$$

Notationally, we denote with \odot any of \otimes and \oplus , and use v to range over the verdicts $\{\text{yes}, \text{no}, \text{end}\}$. The operational semantics of centralized monitors is given in Table 2. Notice that monitors that wait for an action at some location (as prescribed by writing a_ℓ) and do not see that action therein (as stated by A) stop their monitoring activity, by reporting **end**.

Monitors can yield *verdicts* at any point of their computation. This is represented by the judgement \Rightarrow , whose intended use is to evaluate monitors and reach a verdict, whenever possible. The rules are given in Table 3; as one may expect, verdict evaluation is non-deterministic, due to the presence of $+$. Also notice that there can be multiple ways to infer

■ **Table 2** The operational semantics for centralized monitors, where $\odot \in \{\otimes, \oplus\}$.

$v \xrightarrow{A} v$	$\frac{A(\ell) = a}{a_\ell.m \xrightarrow{A} m}$	$\frac{A(\ell) \neq a}{a_\ell.m \xrightarrow{A} \text{end}}$	$\frac{m\{\text{rec } x.m/x\} \xrightarrow{A} m'}{\text{rec } x.m \xrightarrow{A} m'}$	$\frac{m \xrightarrow{A} m'}{m + n \xrightarrow{A} m'}$
	$\frac{n \xrightarrow{A} n'}{m + n \xrightarrow{A} n'}$		$\frac{m \xrightarrow{A} m' \quad n \xrightarrow{A} n'}{m \odot n \xrightarrow{A} m' \odot n'}$	

■ **Table 3** Verdict evaluation for centralized monitors (up to commutativity of $+$, \otimes , and \oplus).

$v \Rightarrow v$	$\frac{m \Rightarrow \text{end}}{n \Rightarrow \text{end}}$	$\frac{m \Rightarrow \text{yes}}{m \oplus n \Rightarrow \text{yes}}$	$\frac{m \Rightarrow \text{no}}{m \otimes n \Rightarrow \text{no}}$	$\frac{m \xrightarrow{A} m' \quad T \xrightarrow{A} T'}{m \triangleright T \mapsto m' \triangleright T'}$
	$\frac{m \Rightarrow \text{no}}{m \oplus n \Rightarrow \text{no}}$	$\frac{m \Rightarrow \text{yes}}{m \otimes n \Rightarrow \text{yes}}$	$\frac{m\{\text{rec } x.m/x\} \Rightarrow v}{\text{rec } x.m \Rightarrow v}$	$\frac{m \Rightarrow v}{m \triangleright T \mapsto v}$
$\frac{m \Rightarrow v}{m + n \Rightarrow v}$	$\frac{n \Rightarrow \text{end}}{m \odot n \Rightarrow \text{end}}$	$\frac{n \Rightarrow \text{yes}}{m \oplus n \Rightarrow \text{yes}}$	$\frac{n \Rightarrow \text{no}}{m \otimes n \Rightarrow \text{no}}$	
	$\frac{n \Rightarrow v}{m \oplus n \Rightarrow v}$	$\frac{n \Rightarrow v}{m \otimes n \Rightarrow v}$		

■ **Table 4** The instrumentation rules for centralized monitors.

the same verdict for the same monitor: e.g., for $\text{yes} \oplus \text{no}$ we can either use the third or the (symmetric version of the) fourth rule from the first line of Table 3. However, the inferred value is of course the same (i.e., yes , in the previous situation).

We instrument a monitor m on a hypertrace T based on the rules of Table 4. As usual, we write \mapsto^* for the reflexive-transitive closure of \mapsto .

From Formulas to Centralized Monitors

We derive monitors for the subset of formulas without least fixed-points, denoted with Hyper-maxHML. More precisely, given a formula φ , we want to derive a monitor that, when monitoring a hypertrace T , returns no if and only if T does not belong to the semantics of φ ; furthermore, if it returns yes , then T belongs to the semantics of φ . All regular properties of infinite traces that can be monitored for violations with the aforementioned guarantees can be expressed without using least fixed-point operators (see the maximality results presented in [5, Proposition 4.18] and [7, Theorem 5.2] in the setting of logics interpreted over infinite traces). Intuitively, we use least fixed-points to describe liveness properties, whose violation does not have a finite witness in general.

The definition of the synthesized monitor is given by induction on φ . This definition is parametrized by a partial function σ , assigning a location to all the free location variables of φ ; when φ is closed, we consider $\text{cm}_\emptyset(\varphi)$. The formal definition is given in Table 5. The interesting cases are for the quantifiers (that are treated as conjunctions and disjunctions, respectively) and for the modal operators.

► **Example 6.** Let $\mathcal{L} = \{1, 2\}$ and $\text{Act} = \{a, b\}$, and consider the formula (2). The monitor synthesis in Table 5 produces the following monitor m when applied to that formula:

$$m = \bigoplus_{\ell \in \{1, 2\}} \text{rec } x.((a_\ell.(a_\ell.x + b_\ell.\text{no}) + b_\ell.\text{yes}) \otimes (b_\ell.(a_\ell.x + b_\ell.\text{no}) + a_\ell.\text{yes})).$$

■ **Table 5** Centralized monitor synthesis.

$\text{cm}_\sigma(\text{tt}) = \text{yes}$	$\text{cm}_\sigma(\text{ff}) = \text{no}$	$\text{cm}_\sigma(x) = x$	$\text{cm}_\sigma(\max x.\varphi) = \text{rec } x.\text{cm}_\sigma(\varphi)$
$\text{cm}_\sigma(\varphi \wedge \varphi') = \text{cm}_\sigma(\varphi) \otimes \text{cm}_\sigma(\varphi')$			$\text{cm}_\sigma(\varphi \vee \varphi') = \text{cm}_\sigma(\varphi) \oplus \text{cm}_\sigma(\varphi')$
$\text{cm}_\sigma(\forall \pi.\varphi) = \bigotimes_{\ell \in \mathcal{L}} \text{cm}_{\sigma[\pi \mapsto \ell]}(\varphi)$			$\text{cm}_\sigma(\exists \pi.\varphi) = \bigoplus_{\ell \in \mathcal{L}} \text{cm}_{\sigma[\pi \mapsto \ell]}(\varphi)$
$\text{cm}_\sigma(\pi = \pi') = \begin{cases} \text{yes} & \text{if } \sigma(\pi) = \sigma(\pi') \\ \text{no} & \text{otherwise} \end{cases}$			$\text{cm}_\sigma(\pi \neq \pi') = \begin{cases} \text{yes} & \text{if } \sigma(\pi) \neq \sigma(\pi') \\ \text{no} & \text{otherwise} \end{cases}$
$\text{cm}_\sigma([a_\pi]\varphi) = a_{\sigma(\pi)}.\text{cm}_\sigma(\varphi) + \sum_{b \neq a} b_{\sigma(\pi)}.\text{yes}$		$\text{cm}_\sigma(\langle a_\pi \rangle \varphi) = a_{\sigma(\pi)}.\text{cm}_\sigma(\varphi) + \sum_{b \neq a} b_{\sigma(\pi)}.\text{no}$	

When monitor m is instrumented with the hypertrace T mapping location 1 to a^ω and location 2 to $(ab)^\omega$, the verdict **no** cannot be reached: indeed, T satisfies the formula φ since the trace at location 1 has a at all positions. On the other hand, when m is instrumented with the hypertrace T' mapping location 1 to b^ω and location 2 to $(ab)^\omega$, the **no** verdict is reached after the monitor has observed the first two actions at locations 1 and 2; this is in line with the fact that T' does not satisfy φ_{h_e} . \lrcorner

The main results of this section are that the centralized monitors synthesized from formulas report sound verdicts and their verdicts are complete for formula violations. We refer the reader to [7] for a discussion on notions of correctness for monitors and the significance of soundness and violation-completeness. The proofs can be found in [2].

► **Theorem 7 (Soundness).** *Let $\varphi \in \text{Hyper-maxHML}$ be a closed formula and $T \in \text{HTrc}_{\mathcal{L}}$. If $\text{cm}_\emptyset(\varphi) \triangleright T \mapsto^* \text{no}$, then $T \notin \llbracket \varphi \rrbracket$; if $\text{cm}_\emptyset(\varphi) \triangleright T \mapsto^* \text{yes}$, then $T \in \llbracket \varphi \rrbracket$.*

► **Theorem 8 (Violation Completeness).** *Let $\varphi \in \text{Hyper-maxHML}$ be a closed formula and $T \in \text{HTrc}_{\mathcal{L}}$. If $T \notin \llbracket \varphi \rrbracket$, then $\text{cm}_\emptyset(\varphi) \triangleright T \mapsto^* \text{no}$.*

4 Decentralized Monitoring

When verifying a distributed system, having a central authority that performs any type of runtime verification is a strong assumption, as it reduces the appeal of distribution. Thus, we study to what extent hyperproperties can be monitored by decentralized monitors.

We associate monitors to locations, denoted by ℓ , and monitors associated to ℓ monitor only actions required to happen at ℓ , thus allowing the processing of events to happen locally. This imposes some form of coordination between monitors at different locations. For this reason, we introduce the possibility for monitors to communicate.

We define a communication alphabet Com , ranged over by c , over some finite alphabet of communication constants Con (that contains Act), ranged over by γ , as

$$\text{Com} \ni c ::= (!G, \gamma) \mid (?G, \gamma),$$

where $G \subseteq \mathcal{L}$ and $\gamma \in \text{Con}$. We have a communication action $(!G, \gamma)$ for sending γ to group G (multicast communication), and one $(?G, \gamma)$ for receiving γ from any monitor from the set G . Point-to-point communication can be represented by taking singleton sets for G .

The syntax of decentralized monitors is given by the following grammar:

$$\text{DMon} \ni M ::= [m]_\ell \mid M \vee M \mid M \wedge M$$

$$\text{LMon} \ni m ::= \text{yes} \mid \text{no} \mid \text{end} \mid a.m \mid c.m \mid m + m \mid m \oplus m \mid m \otimes m \mid \text{rec } x.m \mid x$$

■ **Table 6** The operational semantics for decentralized local monitors (up to commutativity of $+$, \otimes and \oplus), where we let λ denote either a , $(!G, \gamma)$ or $(? \ell, \gamma)$ for $\ell \in \mathcal{L}$, $G \subseteq \mathcal{L}$.

$a.m \xrightarrow{a} m$	$\frac{\ell \in G}{(?G, \gamma).m \xrightarrow{(? \ell, \gamma)} m}$	$(!G, \gamma).m \xrightarrow{(!G, \gamma)} m$	$v \xrightarrow{a} v$
$\frac{m \{\text{rec } x.m / x\} \xrightarrow{\lambda} m'}{\text{rec } x.m \xrightarrow{\lambda} m'}$	$\frac{m \xrightarrow{a} m' \quad n \xrightarrow{a} n'}{m \odot n \xrightarrow{a} m' \odot n'}$	$\frac{m \xrightarrow{(? \ell, \gamma)} m' \quad n \xrightarrow{(? \ell, \gamma)} n'}{m \odot n \xrightarrow{(? \ell, \gamma)} m' \odot n'}$	
$\frac{m \xrightarrow{\lambda} m'}{m + n \xrightarrow{\lambda} m'}$	$\frac{m \xrightarrow{(!G, \gamma)} m'}{m \odot n \xrightarrow{(!G, \gamma)} m' \odot n}$	$\frac{m \xrightarrow{(? \ell, \gamma)} m' \quad n \xrightarrow{(? \ell, \gamma)} n'}{m \odot n \xrightarrow{(? \ell, \gamma)} m' \odot n}$	

■ **Table 7** Operational semantics for communication of $M \in \text{DMon}$ (up to commutativity of \wedge , \vee).

$\frac{m \xrightarrow{(!G, \gamma)} m'}{[m]_\ell \xrightarrow{\ell: (!G, \gamma)} [m']_\ell}$	$\frac{m \xrightarrow{(? \ell', \gamma)} m' \quad \ell \in G}{[m]_\ell \xrightarrow{G: (? \ell', \gamma)} [m']_\ell}$
$\frac{m \xrightarrow{(? \ell', \gamma)} m'}{[m]_\ell \xrightarrow{G: (? \ell', \gamma)} [m]_\ell}$	$\frac{\ell \notin G}{[m]_\ell \xrightarrow{G: (? \ell', \gamma)} [m]_\ell}$
$\frac{M \xrightarrow{G: (? \ell, \gamma)} M' \quad N \xrightarrow{G: (? \ell, \gamma)} N'}{M \diamond N \xrightarrow{G: (? \ell, \gamma)} M' \diamond N'} \diamond \in \{\wedge, \vee\}$	
$\frac{M \xrightarrow{\ell: (!G, \gamma)} M' \quad N \xrightarrow{G: (? \ell, \gamma)} N'}{M \diamond N \xrightarrow{\ell: (!G, \gamma)} M' \diamond N'} \diamond \in \{\wedge, \vee\}$	

■ **Table 8** Operational semantics for actions of $M \in \text{DMon}$ (up to commutativity of \wedge , \vee).

$\frac{A(\ell) = a \quad m \xrightarrow{a} m'}{[m]_\ell \xrightarrow{A} [m']_\ell}$
$\frac{A(\ell) = a \quad m \xrightarrow{a} m' \quad m \xrightarrow{c} m'}{[m]_\ell \xrightarrow{A} [\text{end}]_\ell}$
$\frac{M \xrightarrow{A} M' \quad N \xrightarrow{A} N'}{M \diamond N \xrightarrow{A} M' \diamond N'} \diamond \in \{\wedge, \vee\}$

Monitor $[m]_\ell$ denotes that m monitors the trace located at location ℓ , so, it is ‘localized’ at ℓ (this justifies the name **LMon**). Monitors assigned to the same trace run in parallel and observe identical events; contrary to [1], monitors assigned to different traces are no longer completely isolated from each other, but can now communicate, which is the main new feature of the decentralized set-up.

The operational rules for $m \in \text{LMon}$ are given in Table 6. Notice that, when we have parallel monitors, only one of them at a time can send; by contrast, all those that can receive from some location ℓ are forced to do so.

For $M \in \text{DMon}$, the operational semantics can be found in Table 7 (the rules concerning communication) and Table 8 (the rules concerning action steps). The operational semantics in Table 7 defines multicast, where a monitor located at ℓ sends a message to group G and every monitor at a location in G that can receive from ℓ does so; every monitor that cannot, or that is not in G , does not change its state. The first four rules capture the judgment for inferring when all components of a monitor which are able to receive a certain γ sent from a location do so. Intuitively, ℓ is the location from which message γ was sent to group G , and $M \xrightarrow{G: (? \ell, \gamma)} N$ indicates that every monitor in M located at a location in G that can receive

4:10 Centralized vs Decentralized Monitors for Hyperproperties

■ **Table 9** The verdict combination rules for decentralized monitors (up to commutativity of \wedge and \vee , ranged over by \diamond).

$\frac{m \Rightarrow v}{[m]_\ell \Rightarrow v}$	$\frac{M \Rightarrow \text{end} \quad N \Rightarrow \text{end}}{M \diamond N \Rightarrow \text{end}}$
$\frac{M \Rightarrow \text{no}}{M \wedge N \Rightarrow \text{no}}$	$\frac{M \Rightarrow \text{yes} \quad N \Rightarrow v}{M \wedge N \Rightarrow v}$
$\frac{M \Rightarrow \text{yes}}{M \vee N \Rightarrow \text{yes}}$	$\frac{M \Rightarrow \text{no} \quad N \Rightarrow v}{M \vee N \Rightarrow v}$

■ **Table 10** The evolution of a decentralized monitor instrumented on a hypertrace.

$\frac{M \xrightarrow{A} M' \quad T \xrightarrow{A} T'}{M \triangleright T \mapsto M' \triangleright T'}$
$\frac{M \xrightarrow{\ell: (!G, \gamma)} M'}{M \triangleright T \mapsto M' \triangleright T}$
$\frac{M \Rightarrow v}{M \triangleright T \mapsto v}$

γ from ℓ indeed has received γ and transitioned appropriately in N . The last two rules then actually define communication. In particular, the last rule in Table 7 implements multicast by stipulating that the outcome of the synchronization between a send action $\ell : (!G, \gamma)$ and a receive one of the form $G : (? \ell, \gamma)$ is the send action itself, which can be received by other monitors at locations in G in a larger monitor of which $M \diamond N$ is a sub-term. We note, in passing, that monitors $M \in \text{DMon}$ are “input-enabled”: for each M, G, ℓ and γ , there is always some M' such that $M \xrightarrow{G: (? \ell, \gamma)} M'$. So the last rule in Table 7 (and its symmetric version) can always be applied when the send transition in its premise is available.

Monitors can also locally observe an action, as prescribed by a location-to-action function A ; the rules are given in Table 8. Monitors at the same location observe the same action. If a monitor cannot take the action prescribed by A at its location, the monitor becomes **end**, as stipulated by the second rule given in Table 8. Note that it is not sufficient to trigger that rule when m cannot exhibit action $A(\ell)$: we also require that m cannot communicate. Note that the inability of m to exhibit action $A(\ell)$ is not sufficient to trigger that rule: we also require that m cannot communicate. Intuitively, this is because monitors exhibit an “alternating” behavior in which they observe the next action produced by a system hypertrace and then embark in a sequence of communications with other monitors to inform them of what they observed. As will be made clear in our definition of a weak bisimulation relation presented in Definition 9, such communications are interpreted as internal actions in monitor behavior. Therefore, the inability of some monitor $[m]_\ell$ to perform action $A(\ell)$ can only be gauged in “stable states” – that is, monitor states in which no communication is possible. This design choice is akin to that underlying the definition of refusal testing presented in [46] and of the stable-failures model for (Timed) CSP defined in [49, 50], where the inability of a process to perform some action can only be determined in states that afford no internal computation steps.

Verdict evaluation for $M \in \text{DMon}$ is defined in Table 9 and relies on that for $m \in \text{CMon}$ provided in Table 3. Finally, given a decentralized monitor M and a hypertrace T , the instrumentation of the monitor on the trace is described by the rules of Table 10. As before, we denote with \mapsto^* the reflexive transitive closure of \mapsto .

4.1 Synthesizing Decentralized Monitors Correctly

In this section we describe how to synthesize decentralized monitors “correctly” from formulas, i.e. such that their behavior corresponds to that of the corresponding centralized monitors. The advantage of this approach is that it simplifies the proof that monitors synthesized via a “correct” decentralized synthesis function are sound and violation-complete, by utilizing

the correspondence to centralized monitors. Moreover, it identifies desirable properties of a “correct” decentralized synthesis function that can guide the development of further automated decentralized-monitor synthesis algorithms.

We first define the correspondence between centralized and decentralized monitors and show that this correspondence is sufficient to obtain soundness and violation-completeness in the decentralized setting from the corresponding results in the centralized setting (Theorems 7 and 8). In the remainder of the section, given a synthesis function which takes as inputs a formula φ and a mapping σ from location variables to locations, and outputs a monitor $\mathcal{M}_\sigma(\varphi) \in \text{DMon}$, we specify criteria that allow us to derive this correspondence.

We write $M \rightarrow M'$ to denote the existence of an integer $h > 0$ and of h monitors M_1, \dots, M_h , locations $\ell_1, \dots, \ell_{h-1}$ and communication actions c_1, \dots, c_{h-1} such that $M_1 = M$, $M_h = M'$, and $M_i \xrightarrow{\ell_i:c_i} M_{i+1}$ (for every $i = 1, \dots, h-1$). By definition of \rightarrow on communicating monitors, each c_i is $(!G_i, \gamma_i)$, for some $G_i \subseteq \mathcal{L}$ and $\gamma_i \in \text{Con}$. Similarly, at the level of local monitors we write $m \rightarrow m'$ to denote the existence of an integer $h > 0$, of local monitors m_1, \dots, m_h and of $c_1, \dots, c_h \in \{(!G, \gamma), (? \ell, \gamma) \mid G \subseteq \mathcal{L}, \ell \in \mathcal{L}, \gamma \in \text{Con}\}$ such that $m_1 = m$, $m_h = m'$ and $m_i \xrightarrow{c_i} m_{i+1}$.

The correspondence between the centralized and the decentralized monitors is characterized as a weak bisimulation:

► **Definition 9.** *A binary relation \mathcal{R} over $\text{DMon} \times \text{CMon}$ is a weak bisimulation if and only if, whenever $M \mathcal{R} m$, it holds that:*

1. $\exists M' \in \text{DMon}$ such that $M \rightarrow M'$ and $M' \Rightarrow v$ if and only if $m \Rightarrow v$.
2. If $M \xrightarrow{A} M'$ then $\exists m' \in \text{CMon}$ such that $m \xrightarrow{A} m'$ and $M' \mathcal{R} m'$.
3. If $M \xrightarrow{c} M'$ then $M' \mathcal{R} m$, where $c = \ell : (!G, \gamma)$ for some $\ell \in \mathcal{L}$, $G \subseteq \mathcal{L}$, $\gamma \in \text{Con}$.
4. If $m \xrightarrow{A} m'$ then there exist M_1, M_2, M' such that $M \rightarrow M_1 \xrightarrow{A} M_2 \rightarrow M'$ and $M' \mathcal{R} m'$.

One of the main features of weak bisimilarity is that, if $\mathcal{M}_\sigma(\varphi)$ and $\text{cm}_\sigma(\varphi)$ are weakly bisimilar, then they report the same verdict when observing any hypetrace T ; thus, we obtain violation-completeness and soundness for decentralized monitors from the corresponding results for centralized monitors:

► **Corollary 10 (Soundness).** *Let $T \in \text{HTrc}_{\mathcal{L}}$, $\varphi \in \text{Hyper-maxHML}$ be a closed formula such that $\mathcal{M}_\emptyset(\varphi)$ is defined, and \mathcal{R} a weak bisimulation such that $(\mathcal{M}_\emptyset(\varphi), \text{cm}_\emptyset(\varphi)) \in \mathcal{R}$. If $\mathcal{M}_\emptyset(\varphi) \triangleright T \rightsquigarrow^* \text{no}$, then $T \notin \llbracket \varphi \rrbracket$; if $\mathcal{M}_\emptyset(\varphi) \triangleright T \rightsquigarrow^* \text{yes}$, then $T \in \llbracket \varphi \rrbracket$.*

► **Corollary 11 (Violation Completeness).** *Let $T \in \text{HTrc}_{\mathcal{L}}$, $\varphi \in \text{Hyper-maxHML}$ be a closed formula such that $\mathcal{M}_\emptyset(\varphi)$ is defined, and \mathcal{R} a weak bisimulation such that $(\mathcal{M}_\emptyset(\varphi), \text{cm}_\emptyset(\varphi)) \in \mathcal{R}$. If $T \notin \llbracket \varphi \rrbracket$, then $\mathcal{M}_\emptyset(\varphi) \triangleright T \rightsquigarrow^* \text{no}$.*

We now describe sufficient conditions for any decentralized synthesis function such that there is a weak bisimulation between the centralized and the decentralized monitors synthesized from a formula φ and a location environment σ . Whenever we write $M \xrightarrow{c} N$ for $M, N \in \text{DMon}$, we assume that $c \in \{\ell : (!G, \gamma) \mid \ell \in \mathcal{L}, G \subseteq \mathcal{L}, \gamma \in \text{Con}\}$, as per the labeling of the communication transitions of decentralized monitors. We write $[m]_\ell \in M$, for $M \in \text{DMon}$, if $[m]_\ell$ is one of its constituents: formally, $[m]_\ell \in [m]_\ell$ and, if $[m]_\ell \in M$, then $[m]_\ell \in M \diamond N$ and $[m]_\ell \in N \diamond M$ (recall that \diamond denotes either \wedge or \vee). We start by defining when $M \in \text{DMon}$ can(not) communicate:

► **Definition 12.** *Let $M \in \text{DMon}$. We say $M \in \text{DMon}$ can communicate, if there exists $[m]_\ell \in M$ such that $m \xrightarrow{c} n$ for some $c \in \text{Com}$. Otherwise, we say M cannot communicate.*

► **Definition 13.** We say that a monitor synthesis $\mathcal{M}_-(\cdot)$ is principled when it satisfies the following conditions, for every formula φ and environment σ such that $\mathcal{M}_\sigma(\varphi)$ is defined:

Verdict Agreement: for every verdict v , $\text{Cm}_\sigma(\varphi) \Rightarrow v$ if and only if $\mathcal{M}_\sigma(\varphi) \Rightarrow v$;

Verdict Irrevocability: for every verdict v and $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M_1 \rightarrow M_2 \rightarrow M$, if $M_2 \Rightarrow v$, then $M \Rightarrow v$;

Reactivity: for every A , there exists M such that $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M$;

Bounded Communication: for every $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M'$, there exists M'' such that $M' \rightarrow M''$ and M'' cannot communicate;

Processing-Communication Alternation: for every $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M_1$,

1. $\mathcal{M}_\sigma(\varphi)$ cannot communicate, and
2. $M_1 \xrightarrow{c} M_2$ implies $M_1 \not\xrightarrow{A}$ for every c and A ;

Formula Convergence: if $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M'$, M' cannot communicate, and $\text{Cm}_\sigma(\varphi) \xrightarrow{A} \text{Cm}_{\sigma'}(\varphi')$ for some formula φ' and environment σ' , then $M' = \mathcal{M}_{\sigma'}(\varphi')$.

Let $\mathcal{M}_-(\cdot)$ be a decentralized synthesis function. We define relation $\mathcal{R}_\mathcal{M}$ as follows:

$$\begin{aligned} \mathcal{R}_\mathcal{M} &\triangleq \mathcal{R}_1 \cup \mathcal{R}_2 \\ \mathcal{R}_1 &\triangleq \{(\mathcal{M}_\sigma(\varphi), \text{Cm}_\sigma(\varphi)) \mid \text{FVloc}(\varphi) \subseteq \text{dom}(\sigma)\} \\ \mathcal{R}_2 &\triangleq \{(M', \text{Cm}_{\sigma'}(\varphi')) \mid \text{FVloc}(\varphi) \subseteq \text{dom}(\sigma) \text{ and } \mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M' \rightarrow \mathcal{M}_{\sigma'}(\varphi')\} \end{aligned}$$

The crucial property of any principled synthesis function is the following:

► **Theorem 14.** For every principled synthesis $\mathcal{M}_-(\cdot)$, $\mathcal{R}_\mathcal{M}$ is a weak bisimulation.

4.2 From Formulas to Decentralized Monitors

We now describe how to synthesize decentralized monitors for a fragment of Hyper-maxHML, and show that this synthesis function satisfies Definition 13. This allows us to apply Theorem 14 and obtain soundness and violation-completeness of these synthesized monitors.

In what follows, we consider formulas from PHyper-recHML, the subset of Hyper-recHML given by the following grammar (see Section 5 for a discussion on the choice of fragment):

$$\begin{aligned} \varphi &::= \exists \pi. \varphi \mid \forall \pi. \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \psi \\ \psi &::= \text{tt} \mid \text{ff} \mid \pi = \pi \mid \pi \neq \pi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \max x. \psi \mid \min x. \psi \mid x \mid [a_\pi] \psi \mid \langle a_\pi \rangle \psi \end{aligned}$$

We denote the class of formulas of type ψ with Qf (quantifier free). PHyper-recHML is a subset of Hyper-recHML and thus its semantics over $\text{HTrc}_\mathcal{L}$ is the one given in Table 1.

We synthesize decentralized monitors for the fragment of PHyper-recHML only containing formulas of type ψ without diamonds and least fixed-points, which we call PHyper-maxHML. In section 4.3 we also discuss how diamonds can also be added to the picture. The synthesis for decentralized monitors is given in Table 11. First, we derive a monitor belonging to LMon for formulas of type $\psi \in \text{Qf}$; this synthesis function is parametrized by a location $\ell \in \mathcal{L}$ and a partial function σ from Π to \mathcal{L} that is defined for every free location variable in ψ . Then we derive monitors belonging to DMon for formulas of type φ .

Note that, in the definition of $\text{DM}_\sigma(\psi)$, $\text{Cm}_\sigma(\psi)$ is the monitor resulting from the centralized synthesis function defined in Table 5. Intuitively $\text{DM}_\sigma(\psi)$ synthesizes a local monitor at each location relevant to ψ , which are the locations associated by σ to the free location variables in ψ . If $\sigma = \emptyset$ (and so ψ does not have any free trace variables), there is no need for communication between locations, and in fact a verdict can be obtained from ψ immediately. This verdict coincides with the one reached in the centralized synthesis.

■ **Table 11** Decentralized monitor synthesis, where ℓ_0 is any fixed element of \mathcal{L} .

$$\begin{array}{l}
 \text{Dm}_\sigma^\ell(\text{tt}) = \text{yes} \quad \text{Dm}_\sigma^\ell(\text{ff}) = \text{no} \quad \text{Dm}_\sigma^\ell(x) = x \quad \text{Dm}_\sigma^\ell(\max x.\psi) = \text{rec } x.\text{Dm}_\sigma^\ell(\psi) \\
 \text{Dm}_\sigma^\ell(\psi \wedge \psi') = \text{Dm}_\sigma^\ell(\psi) \otimes \text{Dm}_\sigma^\ell(\psi') \quad \text{Dm}_\sigma^\ell(\psi \vee \psi') = \text{Dm}_\sigma^\ell(\psi) \oplus \text{Dm}_\sigma^\ell(\psi') \\
 \text{Dm}_\sigma^\ell([a_\pi]\psi) = \begin{cases} a.(!(\text{rng}(\sigma) \setminus \{\ell\}), a).\text{Dm}_\sigma^\ell(\psi) + \sum_{b \neq a} b.(!(\text{rng}(\sigma) \setminus \{\ell\}), b).\text{yes} & \text{if } \sigma(\pi) = \ell \\ \sum_{b \in \text{Act}} b.(?\{\sigma(\pi)\}, a).\text{Dm}_\sigma^\ell(\psi) + \sum_{b \neq a} (?\{\sigma(\pi)\}, b).\text{yes} & \text{otherwise} \end{cases} \\
 \text{Dm}_\sigma^\ell(\pi = \pi') = \begin{cases} \text{yes} & \text{if } \sigma(\pi) = \sigma(\pi') \\ \text{no} & \text{otherwise} \end{cases} \quad \text{Dm}_\sigma^\ell(\pi \neq \pi') = \begin{cases} \text{yes} & \text{if } \sigma(\pi) \neq \sigma(\pi') \\ \text{no} & \text{otherwise} \end{cases} \\
 \hline
 \text{DM}_\sigma(\psi) = \begin{cases} \bigvee_{\ell \in \text{rng}(\sigma)} [\text{Dm}_\sigma^\ell(\psi)]_\ell & \text{if } \sigma \neq \emptyset \\ [v]_{\ell_0} & \text{if } \sigma = \emptyset \wedge \text{Cm}_\sigma(\psi) \Rightarrow v \end{cases} \\
 \text{DM}_\sigma(\forall \pi.\varphi) = \bigwedge_{\ell \in \mathcal{L}} \text{DM}_{\sigma[\pi \mapsto \ell]}(\varphi) \quad \text{DM}_\sigma(\exists \pi.\varphi) = \bigvee_{\ell \in \mathcal{L}} \text{DM}_{\sigma[\pi \mapsto \ell]}(\varphi) \\
 \text{DM}_\sigma(\varphi \wedge \varphi') = \text{DM}_\sigma(\varphi) \wedge \text{DM}_\sigma(\varphi') \quad \text{DM}_\sigma(\varphi \vee \varphi') = \text{DM}_\sigma(\varphi) \vee \text{DM}_\sigma(\varphi') \\
 \hline
 \end{array}$$

We observe that the case for $\sigma = \emptyset$ and $\text{Cm}_\sigma(\psi) \Rightarrow v$ only applies when ψ is a Boolean combination of tt and ff . Thus, every closed formula φ on which we apply our synthesis

1. is trivial, *i.e.* φ is logically equivalent to tt or ff , or
2. is such that every subformula $\psi \in \text{Qf}$ of φ is in the scope of a quantifier.

For non-trivial formulas, the $\sigma = \emptyset$ case for $\text{DM}_\sigma(\psi)$ never applies, and we can ignore it. The decentralized monitor for a closed formula φ is $\text{DM}_\emptyset(\varphi)$.

► **Remark 15.** In the first clause of the definition of the synthesis function for box formulas, it might seem superfluous to send a message also when the monitor observes some $b \neq a$. However, this is important to make sure monitors do not deadlock. To see this, consider a synthesis where that definition instead looks like

$$\text{Dm}_\sigma^\ell([a_\pi]\psi) = \begin{cases} a.(!(\text{rng}(\sigma) \setminus \{\ell\}), a).\text{Dm}_\sigma^\ell(\psi) + \sum_{b \neq a} b.\text{yes} & \text{if } \sigma(\pi) = \ell \\ \sum_{b \in \text{Act}} b.(?\{\sigma(\pi)\}, a).\text{Dm}_\sigma^\ell(\psi) & \text{otherwise} \end{cases}$$

Consider $\text{Act} = \{a, b\}$, $\mathcal{L} = \{\ell, \ell'\}$ and some hypertrace T such that $T(\ell) = b.t_1$ and $T(\ell') = b.t_2$ for some traces t_1 and t_2 . Now consider $m \otimes n$, where $m = \text{Dm}_\sigma^\ell([a_\pi]\psi)$, $n = \text{Dm}_\sigma^\ell([a_{\pi'}]\psi')$, $\sigma(\pi) = \ell$ and $\sigma(\pi') = \ell'$. For $A(\ell) = A(\ell') = b \neq a$, we then get $m \xrightarrow{A(\ell)} \text{yes}$ and $n \xrightarrow{A(\ell')} (?\{\sigma(\pi')\}, a).\text{Dm}_\sigma^\ell(\psi')$, and monitor $\text{yes} \otimes (?\{\sigma(\pi')\}, a).\text{Dm}_\sigma^\ell(\psi')$ is stuck because the receive action of the monitor $(?\{\sigma(\pi')\}, a).\text{Dm}_\sigma^\ell(\psi')$ has no matching send. It is precisely to avoid these scenarios that we make sure that, for each sending transition, there is a corresponding receiving transition, and a monitor always sends the last action it read to all other locations in the range of the environment σ . \square

Soundness and violation completeness for the synthesis defined in Table 11 follow from Corollary 10 and 11 by using Theorem 14, once we prove the following key result:

► **Theorem 16.** *The synthesis function DM defined in Table 11 is principled.*

4:14 Centralized vs Decentralized Monitors for Hyperproperties

► **Example 17.** In order to highlight the inter-monitor communication, we consider the following formula

$$\varphi = \exists \pi. \exists \pi'. ([a_\pi] \text{ff} \wedge [b_{\pi'}] \text{ff})$$

over $\mathcal{L} = \{1, 2\}$ and $\text{Act} = \{a, b\}$, which states that either both traces start with a , or neither does. By letting $\sigma = [\pi \mapsto \ell, \pi' \mapsto \ell']$, the synthesis for this property gives:

$$\text{DM}_\emptyset(\varphi) = \bigvee_{\ell, \ell' \in \mathcal{L}} \bigvee_{\ell'' \in \{\ell, \ell'\}} \left[\text{Dm}_\sigma^{\ell''} ([a_\pi] \text{ff} \wedge [b_{\pi'}] \text{ff}) \right]_{\ell''}, \text{ where}$$

$$\text{Dm}_\sigma^{\ell''} ([a_\pi] \text{ff} \wedge [b_{\pi'}] \text{ff}) = \begin{cases} (a.(!\emptyset, a).\text{no} + b.(!\emptyset, b).\text{yes}) \otimes & \text{if } \ell = \ell' = \ell'' \\ (b.(!\emptyset, b).\text{no} + a.(!\emptyset, a).\text{yes}) & \\ (a.(!\{\ell'\}, a).\text{no} + b.(!\{\ell'\}, b).\text{yes}) \otimes & \text{if } \ell \neq \ell' \text{ and } \ell'' = \ell \\ (a.(?\{\ell'\}, b).\text{no} + (?\{\ell'\}, a).\text{yes}) + & \\ b.(?\{\ell'\}, b).\text{no} + (?\{\ell'\}, a).\text{yes}) & \\ (a.(?\{\ell\}, a).\text{no} + (?\{\ell\}, b).\text{yes}) + & \text{if } \ell \neq \ell' \text{ and } \ell'' = \ell' \\ b.(?\{\ell\}, a).\text{no} + (?\{\ell\}, b).\text{yes}) & \\ \otimes (b.(!\{\ell\}, b).\text{no} + a.(!\{\ell\}, a).\text{yes}) & \end{cases}$$

⌋

4.3 On the Decentralized-Monitor Synthesis for Diamonds

The synthesis of decentralized monitors presented in Table 11 does not deal explicitly with formulas of the form $\langle a_\pi \rangle \psi$. However, it can be applied to those formulas using the observation that $\langle a_\pi \rangle \psi$ is logically equivalent to

$$[a_\pi] \psi \wedge \bigwedge_{b \neq a} [b_\pi] \text{ff}. \quad (3)$$

To showcase this, we present an example of the decentralized synthesis applied on Wolper's property (φ_{h_e}) from Example 3, which makes use of diamond modalities.

► **Example 18.** Recall φ_{h_e} from (2); expressed here as $\exists \pi. \psi$, with

$$\psi = \max x. (\psi_1 \wedge \psi_2) \quad \psi_1 = [a_\pi] \langle a_\pi \rangle x \quad \psi_2 = [b_\pi] \langle a_\pi \rangle x$$

Let $\mathcal{L} = \{1, 2\}$ and $\text{Act} = \{a, b\}$. The synthesis is applied thus:

$$\text{DM}_\emptyset(\varphi) = \bigvee_{\ell \in \mathcal{L}} \left[\text{rec } x. \left(m_{[\pi \mapsto \ell]}^\ell(\psi_1) \otimes m_{[\pi \mapsto \ell]}^\ell(\psi_2) \right) \right]_\ell$$

with

$$\begin{aligned} m_{[\pi \mapsto \ell]}^\ell(\psi_1) &= a.(!\emptyset, a).m_{[\pi \mapsto \ell]}^\ell(\langle a_\pi \rangle x) + b.(!\emptyset, b).\text{yes} \\ m_{[\pi \mapsto \ell]}^\ell(\psi_2) &= b.(!\emptyset, b).m_{[\pi \mapsto \ell]}^\ell(\langle a_\pi \rangle x) + a.(!\emptyset, a).\text{yes} \end{aligned}$$

and

$$m_{[\pi \mapsto \ell]}^\ell(\langle a_\pi \rangle x) = (a.(!\emptyset, a).x + b.(!\emptyset, b).\text{yes}) \otimes (b.(!\emptyset, b).\text{no} + a.(!\emptyset, a).\text{yes}) \quad (4)$$

As the monitors in Example 18 indicate, a decentralized monitor synthesis for formulas of the form $\langle a_\pi \rangle \psi$ that is based on the encoding of (3) leads to monitors with a high degree of parallelism; for simplicity, the degree in Example 18 is reduced because we assumed

to have just two actions. However, $|\text{Act}| - 1$ parallel conjunctions are required in general. Alternatively, one could define a decentralized monitor synthesis directly for formulas of the form $\langle a_\pi \rangle \psi$ as follows:

$$m_\sigma^\ell(\langle a_\pi \rangle \psi) = \begin{cases} a.(!(\text{rng}(\sigma) \setminus \{\ell\}), a).\text{Dm}_\sigma^\ell(\psi) + \sum_{b \neq a} b.(!(\text{rng}(\sigma) \setminus \{\ell\}), b).\text{no} & \text{if } \sigma(\pi) = \ell \\ \sum_{b \in \text{Act}} b.((?\{\sigma(\pi)\}, a).\text{Dm}_\sigma^\ell(\psi) + \sum_{b \neq a} (?\{\sigma(\pi)\}, b).\text{no}) & \text{otherwise} \end{cases}$$

This is essentially the synthesis for box formulas in Table 11 with `no` verdicts in place of `yes`. With this explicit rule for diamonds, (4) simply reduces to:

$$m_{[\pi \rightarrow \ell]}^\ell(\langle a_\pi \rangle x) = a.(!\emptyset, a).x + b.(!\emptyset, b).\text{no}$$

The synthesized monitor for diamond now contains no occurrence of any parallel operator.

5 Conclusion

We provided two methods to synthesize monitors for hyperproperties expressed as fragments of Hyper-`recHML`. Our first synthesis procedure constructs monitors that analyse hypertraces in a centralized manner and are guaranteed to correctly detect all violations of the respective formula, as long as it does not have a least fixed-point operator. Our second synthesis algorithm constructs monitors that operate in a decentralized manner and communicate with one another using multicast to share relevant information between them. The decentralized-monitor synthesis provides the same correctness guarantees as the centralized one, but is only defined for formulas with trace quantifiers that do not appear inside any fixed-point operator. This additional restriction, which is natural and present in many monitoring set-ups for hyperlogics, *e.g.* [10, 19, 23, 26, 30, 36], allows us to focus on examining the intricacies of monitoring in a decentralized setting with monitor communication. More precisely, it allows us to fix the σ in the synthesis function which, in turn, produces a *static* set of locations with which a monitor can communicate. Despite the restriction to PHyper-`recHML`, our synthesis algorithm still covers properties that were previously not even expressible, hence not monitorable, in state-of-the-art hyperlogics.

Of course, the picture is still incomplete: we have a centralized-monitor synthesis procedure for an expressive fragment of Hyper-`recHML`, whereas our decentralized-monitor synthesis deals with a more restricted fragment of that logic. It is not clear if this restriction is necessary; for example, a different decentralized-monitor synthesis for a larger fragment might be obtained by utilizing a different communication paradigm other than multicast, which was adopted in this study. In fact, we conjecture that broadcast communications might allow us to synthesize decentralized monitors for a larger Hyper-`recHML` fragment, including formulae that mix greatest fixed-points and quantifiers, like φ_a defined in (1); currently, monitors only send messages to the locations in the range of the specified σ . Another interesting direction is to allow monitors to infer information from communications they did not receive. A good starting point to explore such a synthesis algorithm (and prove its correctness) can be the synthesis properties in Definition 13. To fully delineate the power of decentralized monitoring, a maximality result in the spirit of those presented in [5, 7] is needed, which we intend to establish in the future.

Although we have focused on monitors that detect violations, we can also synthesize monitors that detect all satisfying hypertraces for the respective dual fragments of Hyper-`recHML`. Another direction we intend to pursue in future is the development of tools for monitoring Hyper-`recHML` specifications at runtime, based on the results of this article. We

expect that our decentralised-monitor synthesis procedure can be implemented by generating a dedicated monitor for every location in a way that is very similar to the synthesis of μ HML monitors presented in [3, 4, 11] and implemented in the tool `detectEr` available at <https://duncanatt.github.io/detector/>.

Related Work. To the best of our knowledge, Agrawal and Bonakdarpour were the first to study RV for hyperproperties expressed in HyperLTL in [10], where they investigated monitorability for k -safety hyperproperties expressed in HyperLTL. They also gave a semantic characterization of monitorable k -safety hyperproperties, which is a natural extension to hyperproperties of the “universal version” of the classic definition of monitorability presented by Pnueli-Zaks [7, 48]. In contrast to this work, we do not restrict ourselves to alternation-free formulas (see Equation (1)) and every monitorable formula considered by Agrawal and Bonakdarpour can be expressed in our monitorable fragment. Brett et al. [23] improve on the work presented in [10] by presenting an algorithm for monitoring the full alternation-free fragment of HyperLTL. They also highlight challenges that arise when monitoring arbitrary HyperLTL formulas, namely (i) quantifier alternations, (ii) inter-trace dependencies and (iii) relative ordering of events across traces. Our decentralized-monitor synthesis addresses (i) by using the number of locations as an upper bound on the number of traces, and (ii) and (iii) via synchronized multicasts.

In [30], Finkbeiner et al. investigate RV for HyperLTL [26] formulas w.r.t. three different input classes, namely the bounded sequential, the unbounded sequential and the parallel classes. They also develop the monitoring tool RVHyper [29] based on the sequential algorithms developed for those input classes. The parallel class is closest to our set-up, since it consists in a *fixed* number of system executions that are processed synchronously.

Beutner et al. [15] study runtime monitoring for $\text{HYPER}^2\text{LTL}_{\text{fp}}$, a temporal logic that is interpreted over sets of *finite* traces of *equal length*. Unlike HYPER^2LTL [14], $\text{HYPER}^2\text{LTL}_{\text{fp}}$ permits quantification under temporal operators, which is also allowed in our logic Hyper-recHML. In contrast to HyperLTL, $\text{HYPER}^2\text{LTL}_{\text{fp}}$ features second-order quantification over sets of finite traces and can express properties like common knowledge.

In [36], Gustfeld et al. study automated analysis techniques for asynchronous hyperproperties and propose a novel automata-theoretic framework, the so-called alternating asynchronous parity automata, together with the fixed-point logic H_μ for expressing asynchronous hyperproperties. The logic H_μ has commonalities with PHyper-recHML, but it only allows for prenex formulas; moreover, its semantics progresses asynchronously on each trace. Properties such as “an atomic proposition does not occur at a certain level in the tree (of traces)” are not expressible in their logic H_μ , but can be described in Hyper-recHML.

Chalupa and Henzinger [25] explore the potential of monitoring for hyperproperties using prefix transducers. They develop a transducer language, called prefix expressions, give it an operational semantics over a hypertrace (reminiscent of the semantics in Section 4) and then implement it to assess the induced overheads. They show how transducers can use the writing capabilities as a method for monitor synchronization across traces, akin to the monitor communication and verdict aggregation of Section 4. Since transducers are, in principle, more powerful than passive monitors, additional guarantees are required to ensure that they do not interfere unnecessarily with system executions.

References



- 1 Luca Aceto, Antonis Achilleos, Elli Anastasiadi, and Adrian Francalanza. Monitoring hyperproperties with circuits. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems – 42nd IFIP WG 6.1 International Conference, FORTE 2022*, volume 13273 of *LNCS*, pages 1–10. Springer, 2022. doi:10.1007/978-3-031-08679-3_1.

- 2 Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Daniele Gorla, and Jana Wagemaker. Centralized vs decentralized monitors for hyperproperties. *CoRR*, abs/2405.12882, 2024. doi:10.48550/arXiv.2405.12882.
- 3 Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, and Anna Ingólfssdóttir. A monitoring tool for linear-time μ hml. In *COORDINATION*, volume 13271 of *LNCS*, pages 200–219. Springer, 2022.
- 4 Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, and Anna Ingólfssdóttir. A monitoring tool for linear-time μ HML. *Sci. Comput. Program.*, 232:103031, 2024. doi:10.1016/j.scico.2023.103031.
- 5 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfssdóttir, and Karoliina Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proc. ACM Program. Lang. POPL*, 3(52):1–29, 2019. doi:10.1145/3290365.
- 6 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfssdóttir, and Karoliina Lehtinen. Testing equivalence vs. runtime monitoring. In *Models, Languages, and Tools for Concurrent and Distributed Programming*, volume 11665 of *LNCS*, pages 28–44. Springer, 2019.
- 7 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfssdóttir, and Karoliina Lehtinen. An operational guide to monitorability with applications to regular properties. *Softw. Syst. Model.*, 20(2):335–361, 2021.
- 8 Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfssdóttir. Runtime Instrumentation for Reactive Components. In *ECOOP*, volume 313 of *LIPICs*, pages 16:1–16:33. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 9 Luca Aceto, Ian Cassar, Adrian Francalanza, and Anna Ingólfssdóttir. On first-order runtime enforcement of branching-time properties. *Acta Informatica*, 60(4):385–451, 2023.
- 10 Shreya Agrawal and Borzoo Bonakdarpour. Runtime Verification of k-Safety Hyperproperties in HyperLTL. In *IEEE 29th Computer Security Foundations Symposium*, pages 239–252. IEEE Computer Society, 2016.
- 11 Duncan Paul Attard, Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfssdóttir, and Karoliina Lehtinen. Better late than never or: Verifying asynchronous components at runtime. In *FORTE*, volume 12719 of *LNCS*, pages 207–225. Springer, 2021.
- 12 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification – Introductory and Advanced Topics*, volume 10457 of *LNCS*, pages 1–33. Springer, 2018. doi:10.1007/978-3-319-75632-5_1.
- 13 Raven Beutner and Bernd Finkbeiner. Software verification of hyperproperties beyond k-safety. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification – 34th International Conference, CAV 2022*, volume 13371 of *LNCS*, pages 341–362. Springer, 2022. doi:10.1007/978-3-031-13185-1_17.
- 14 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Second-order hyperproperties. In *CAV (2)*, volume 13965 of *LNCS*, pages 309–332. Springer, 2023.
- 15 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Monitoring second-order hyperproperties. In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum, editors, *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024*, pages 180–188. ACM, 2024. URL: <https://dl.acm.org/doi/10.5555/3635637.3662865>.
- 16 Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *Theor. Comput. Sci.*, 669:33–58, 2017.
- 17 Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 – Concurrency Theory*, pages 162–176, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- 18 Borzoo Bonakdarpour and Bernd Finkbeiner. Runtime verification for HyperLTL. In Yliès Falcone and César Sánchez, editors, *Runtime Verification – 16th International Conference, RV 2016, Madrid, Spain, September 23–30, 2016, Proceedings*, volume 10012 of *LNCS*, pages 41–45. Springer, 2016. doi:10.1007/978-3-319-46982-9_4.



- 19 Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 162–174. IEEE Computer Society, 2018. doi:10.1109/CSF.2018.00019.
- 20 Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, David A. Rosenblueth, and Corentin Travers. Decentralized asynchronous crash-resilient runtime verification. *J. ACM*, 69(5):34:1–34:31, 2022. doi:10.1145/3550483.
- 21 Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Challenges in fault-tolerant distributed runtime verification. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications – 7th International Symposium, ISoLA 2016,*, volume 9953 of *LNCS*, pages 363–370, 2016. doi:10.1007/978-3-319-47169-3_27.
- 22 Borzoo Bonakdarpour, César Sánchez, and Gerardo Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification – 8th International Symposium, ISoLA 2018*, volume 11245 of *LNCS*, pages 8–27. Springer, 2018. doi:10.1007/978-3-030-03421-4_2.
- 23 Noel Brett, Umair Siddique, and Borzoo Bonakdarpour. Rewriting-based runtime verification for alternation-free hyperltl. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 77–93, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.
- 24 Ian Cassar, Adrian Francalanza, Claudio Antares Mezzina, and Emilio Tuosto. Reliability and fault-tolerance by choreographic design. In Adrian Francalanza and Gordon J. Pace, editors, *Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques, PrePost@iFM 2017, Torino, Italy, 19 September 2017*, volume 254 of *EPTCS*, pages 69–80, 2017. doi:10.4204/EPTCS.254.6.
- 25 Marek Chalupa and Thomas A. Henzinger. Monitoring hyperproperties with prefix transducers. In *RV*, volume 14245 of *LNCS*, pages 168–190. Springer, 2023.
- 26 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust – Third International Conference, POST 2014*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.
- 27 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
- 28 E. Allen Emerson and Joseph Y. Halpern. “Sometimes” and “Not Never” revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.
- 29 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In *TACAS (2)*, volume 10806 of *LNCS*, pages 194–200. Springer, 2018.
- 30 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods Syst. Des.*, 54(3):336–363, 2019. doi:10.1007/s10703-019-00334-z.
- 31 Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. A lower bound on the number of opinions needed for fault-tolerant decentralized run-time monitoring. *J. Appl. Comput. Topol.*, 4(1):141–179, 2020. doi:10.1007/s41468-019-00047-6.
- 32 Adrian Francalanza. Consistently-detecting monitors. In *CONCUR*, volume 85 of *LIPICs*, pages 8:1–8:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.
- 33 Adrian Francalanza. A Theory of Monitors. *Inf. Comput.*, 281:104704, 2021. doi:10.1016/j.ic.2021.104704.
- 34 Adrian Francalanza, Luca Aceto, and Anna Ingólfssdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods Syst. Des.*, 51(1):87–116, 2017. doi:10.1007/S10703-017-0273-Z.

- 35 Adrian Francalanza, Andrew Gaudi, and Gordon J. Pace. Distributed system contract monitoring. *J. Log. Algebraic Methods Program.*, 82(5-7):186–215, 2013.
- 36 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434319.
- 37 Christopher Hahn, Marvin Stenger, and Leander Tentrup. Constraint-based monitoring of hyperproperties. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 115–131, Cham, 2019. Springer International Publishing.
- 38 Jun Inoue and Yoriyuki Yamagata. Operational semantics of process monitors. In *RV*, volume 10548 of *LNCS*, pages 403–409. Springer, 2017.
- 39 Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment for higher-order session types. In *POPL*, pages 582–594. ACM, 2016.
- 40 Dexter Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. doi:10.1016/0304-3975(82)90125-6.
- 41 Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000. doi:10.1145/333979.333987.
- 42 Ruggero Lanotte, Massimo Merro, and Andrei Munteanu. A process calculus approach to detection and mitigation of PLC malware. *Theor. Comput. Sci.*, 890:125–146, 2021.
- 43 Ruggero Lanotte, Massimo Merro, and Andrei Munteanu. Industrial control systems security via runtime enforcement. *ACM Trans. Priv. Secur.*, 26(1):4:1–4:41, 2023.
- 44 Kim G. Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2):265–288, 1990. doi:10.1016/0304-3975(90)90038-J.
- 45 Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies: A monitors-as-memories approach. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*, PPDP '17, pages 127–138, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3131851.3131864.
- 46 Iain Phillips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987. doi:10.1016/0304-3975(87)90117-4.
- 47 Amir Pnueli. The temporal logic of programs. In *FOCS'77, 18th IEEE Annual Symposium on Foundations of Computer Science, Proceedings*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.
- 48 Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In *FM*, volume 4085 of *LNCS*, pages 573–586. Springer, 2006.
- 49 George M. Reed and A. W. Roscoe. The timed failures-stability model for CSP. *heoretical Computer Science*, 211(1-2):85–127, 1999. doi:10.1016/S0304-3975(98)00214-X.
- 50 A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall PTR, USA, 1997.
- 51 Moshe Y. Vardi. A temporal fixpoint calculus. In Jeanne Ferrante and Peter Mager, editors, *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 250–259. ACM Press, 1988. doi:10.1145/73560.73582.
- 52 Pierre Wolper. Temporal logic can be more expressive. *Inf. Control.*, 56(1/2):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.

MITL Model Checking via Generalized Timed Automata and a New Liveness Algorithm

S. Akshay  


Department of CSE, Indian Institute of Technology Bombay, Mumbai, India

Paul Gastin  

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
CNRS, ReLaX, IRL 2000, Siruseri, India

R. Govind  

Uppsala University, Sweden

B. Srivathsan  

Chennai Mathematical Institute, India
CNRS, ReLaX, IRL 2000, Siruseri, India

Abstract

The translation of Metric Interval Temporal Logic (MITL) to timed automata is a topic that has been extensively studied. A key challenge here is the conversion of future modalities into equivalent automata. Typical conversions equip the automata with a guess-and-check mechanism to ascertain the truth of future modalities. Guess-and-check can be naturally implemented via alternation. However, since timed automata tools do not handle alternation, existing methods perform an additional step of converting the alternating timed automata into timed automata. This “de-alternation” step proceeds by an intricate finite abstraction of the space of configurations of the alternating automaton.

Recently, a model of generalized timed automata (GTA) has been proposed. The model comes with several powerful additional features, and yet, the best known zone-based reachability algorithms for timed automata have been extended to the GTA model, with the same complexity for all the zone operations. An attractive feature of GTAs is the presence of future clocks which act like timers that guess a time to an event and stay alive until a timeout. Future clocks seem to provide another natural way to implement the guess-and-check: start the future clock with a guessed time to an event and check its occurrence using a timeout. Indeed, using this feature, we provide a new concise translation from MITL to GTA. In particular, for the timed until modality, our translation offers an exponential improvement w.r.t. the state-of-the-art.

Thanks to this conversion, MITL model checking reduces to checking liveness for GTAs. However, no liveness algorithm is known for GTAs. Due to the presence of future clocks, there is no finite time-abstract bisimulation (region equivalence) for GTAs, whereas liveness algorithms for timed automata crucially rely on the presence of the finite region equivalence. As our second contribution, we provide a new zone-based algorithm for checking Büchi non-emptiness in GTAs, which circumvents this fundamental challenge.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Quantitative automata; Theory of computation → Logic and verification

Keywords and phrases MITL model checking, timed automata, zones, liveness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.5

Related Version *Full Version:* <https://arxiv.org/abs/2407.08452> [2]

1 Introduction

The translation of Linear Temporal Logic (LTL) [32] to Büchi automata is a fundamental problem in model checking, with a long history of theoretical advances [20, 36, 18], tool implementations [25, 14, 18, 30, 12] and practical applications [33, 34, 26, 27]. In the real-time setting, Metric Interval Temporal Logic (MITL) is close to LTL, with the modalities Next



© S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 5; pp. 5:1–5:19

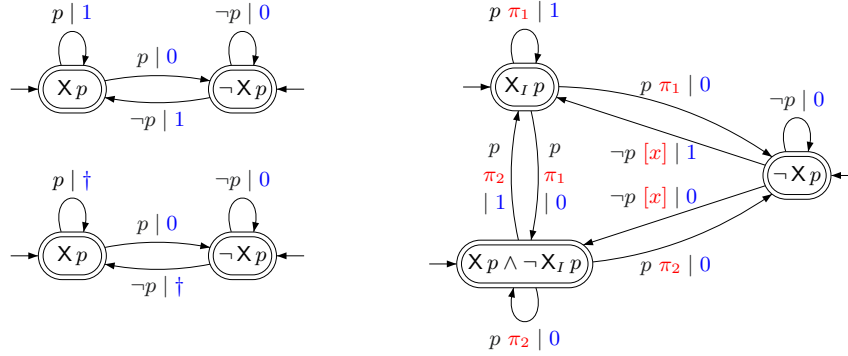


Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(X) and Until (U) extended with timing intervals – for instance, $X_{[a,b]}p$ says that the next event is a p and it occurs within a delay $\theta \in [a, b]$. Model checking for MITL is known to be EXPSPACE-complete [4]. This has led to the study of “efficient” conversions from MITL to timed automata, with each new construction aiming to make the automata more succinct. Our work is another step in this direction.

There are two ways to interpret MITL formulae: over (continuous) timed signals [4, 29, 15] or (pointwise) timed words [6, 37, 11]. Since the current timed automata tools work with timed words, we stick with the pointwise semantics. The state-of-the-art for MITL-to-TA is based on an initial translation of MITL to one-clock Alternating Timed Automata (OCATA) [31]. It has been shown that these OCATA can be converted to a network of timed automata [9, 10]. The tool MightyL [11] implements the entire MITL-to-TA translation. One of the difficulties in the MITL-to-TA translation is the inherent mismatch between the logic and the automaton in the way timing constraints are enforced. A future modality declares that a certain event takes place at a certain timing distance, *in the future*. In a timed automaton, *clocks* measure time elapsed since some event *in the past* and check constraints on these values. To implement a future modality, the automaton needs to make a prediction about the event and verify that the prediction is indeed true. Therefore, each prediction typically resets a clock and stores a new obligation in the state. The automaton needs to discharge these obligations at the right times in the future.



■ **Figure 1** (top left) Büchi transducer (with outputs) for LTL formula Xp (right) Timed transducer with clock x for MITL formula $X_I p$; $\pi_1 := x \in I; [x]$, $\pi_2 := x \notin I; [x]$ (bottom left) a hypothetical transducer with a variable θ that predicts time to next action; $\dagger := \theta \in I ? 1 : 0$.

Figure 1 (top left) shows an automaton with outputs for the LTL formula Xp . On an infinite word $w_1 w_2 \dots$ (where each w_i is a subset of atomic propositions) the automaton outputs **1** at w_i iff w_{i+1} contains p . While reading w_i , the automaton needs to guess whether $p \in w_{i+1}$ or not. Depending on the guess, it stores an appropriate obligation. This is reflected in the states and transitions: transitions with output **1** go to a state Xp which can only read p next, whereas those with output **0** go to $\neg Xp$ which can only read $\neg p$. The Xp and $\neg Xp$ can be seen as obligations that the automaton has to discharge from the state.

Now, let us consider a timed version $X_I p$ interpreted on timed words. An automaton for $X_I p$ needs to guess whether the next letter is a p and if so, whether it appears within θ time units for some $\theta \in I$. Figure 1 (bottom left) represents a hypothetical automaton that implements this idea: assuming it has access to a variable θ which contains the time to the next event, the output should depend on whether $\theta \in I$ or not. This is exactly what the if-then-else condition \dagger does: if $\theta \in I$ output **1**, else output **0**. Classical timed automata do not have direct access to θ . They implement this idea differently, by making use of extra

states. Figure 1 (right) shows a timed automaton for $X_I p$. The state $X p$ is split into two different obligations: $X_I p$ where the timing constraint is satisfied, and $X p \wedge \neg X_I p$ where it is not. The outgoing guards discharge these obligations. This example shows the convenience of having access to a variable that can predict time to future events.

This is precisely where the recently proposed model of *Generalized Timed Automata* (GTA) [1] enters the picture. This model subsumes event-clock automata [5] and automata with timers [13]. GTA come equipped with the additional resources to implement predictions better. GTA have two types of clocks: *history clocks* and *future clocks*. History clocks are similar to the usual clocks of timed automata. Future clocks are like timers, but instead of starting them at some non-negative value and making them go down to zero, they get started with some arbitrary negative value and go up until they hit zero. For example, in Figure 1 (bottom left), each transition can start a future clock, guessing the time to the next event. This immediately gives us the required θ . The exact GTA for $X_I p$ is quite close to Figure 1 (bottom left) and is given in Figure 4. Apart from the use of future clocks, the syntax of transitions in a GTA is much richer than a guard-reset pair as in timed automata. Transitions contain an “instantaneous timed program”, which consists of a sequence of guards, resets and releases (for future clocks). When difference constraints are present, the model becomes powerful enough to encode counter machines and is therefore undecidable. A safe fragment, with a careful use of diagonal constraints is known to be decidable.

GTAs are advantageous in another sense. In spite of the powerful features, the best zone-based algorithms from the timed automata literature have been shown to suitably adapt to the GTA setting, with the same complexity for zone operations, and have been implemented in the tool TChecker [21]. Therefore, an MITL to GTA conversion allows us to capitalize on the features and succinct syntax of GTA, and at the same time, lets us model check MITL directly on richer GTA models. In summary:

- We provide a translation of MITL formulae to safe GTA. The translation is compositional and implementable, and yields an *exponential improvement* in the number of locations compared to the state-of-the-art technique for pointwise semantics [11], while the number of clocks remains the same up to a constant.
- Model checking MITL against GTA requires to solve the liveness problem for (safe) GTAs, which has been open so far. We settle the liveness question in this work. Zone based algorithms for event-clock automata have been studied in [19]. A notion of weak regions has been developed and this can be used for solving both reachability and liveness using zones. The GTA model that we consider in this paper strictly subsumes event-clock automata. In particular, the presence of diagonal constraints makes the problem more challenging. Our solution to liveness for GTAs therefore gives an alternate liveness procedure for event-clock automata, and also settles liveness for event-clock automata with diagonal constraints, a model defined in [8].

We remark that the techniques used in continuous semantics do not extend to pointwise semantics. In [15] the authors simplify general MITL formulae into formulae containing only one-sided intervals (of the form $[0, c]$ or $[c, \infty)$), for which automata are considerably simpler to construct. However, this simplification at the formula level works *only* in the continuous semantics – it does not work in the pointwise-semantics (as Lemma 4.3, 4.4 of [15] do not extend to pointwise-semantics). The fundamental difference is that in the continuous semantics we can assert a formula at any time point t . However, in pointwise-semantics, we can evaluate a formula only at *action points*, i.e., points where there is an actual action. For example, in continuous semantics one can rewrite $F_{[a+c, b+c]} p$ as $F_{[0, c]} G_{[0, c]} F_{[a, b]} p$ when $c \leq b - a$ (Lemma 4.3, [15]). However, in the pointwise semantics there may be no event in the interval $[0, c]$ on which we can evaluate $G_{[0, c]} F_{[a, b]} p$. Therefore, we need a completely different approach to deal with intervals in the pointwise semantics.

Organization of the paper. We start with preliminary definitions of Generalized Timed Automata (Section 2) and provide our solution to the liveness problem in Section 3. We present our MITL to GTA translation in Section 4. Missing proofs and additional explanations can be found in the full version available at [2].

2 Preliminaries

Let $X = X_F \uplus X_H$ be a set of real-valued variables called *clocks*, which is further partitioned into *future clocks* X_F and *history clocks* X_H . Let $\Phi(X)$ denote a set of *clock constraints* generated by the grammar: $\varphi ::= x - y \triangleleft c \mid \varphi \wedge \varphi$ where $x, y \in X \cup \{0\}$, $\triangleleft \in \{<, \leq\}$ and $c \in \mathbb{Z} = \mathbb{Z} \cup \{-\infty, +\infty\}$ (the set of integers equipped with the two special values to say that a clock is “undefined”). We also allow *renamings* of clocks. Let perm_X be the set of permutations σ over $X \cup \{0\}$ mapping history (resp. future) clocks to history (resp. future) clocks ($\sigma(X_F) = X_F$ and $\sigma(X_H) = X_H$).

GTA syntax. A *Generalized Timed Automaton (GTA)* is given by $(Q, \Sigma, X, \Delta, \mathcal{I}, Q_f)$ where Q is a finite set of states, Σ is a finite alphabet of actions, $X = X_F \uplus X_H$ is a set of clocks partitioned into future clocks X_F and history clocks X_H . The initialization condition \mathcal{I} is a set of pairs (q_0, g_0) where a pair consists of an initial state $q_0 \in Q$ and an initial guard $g_0 \in \Phi(X)$, and the accepting condition is given by a set $Q_f \subseteq Q$ of Büchi states. The transition relation $\Delta \subseteq (Q \times \Sigma \times \text{Programs} \times Q)$ contains transitions of the form (q, a, prog, q') , where q is the source state, q' is the target state, a is the action triggering the transition, and prog is an *instantaneous timed program* generated by the grammar:

$$\text{prog} := \text{guard} \mid \text{change} \mid \text{rename} \mid \text{prog}; \text{prog}$$

where $\text{guard} = g \in \Phi(X)$, $\text{change} = [R]$ for an $R \subseteq X$, and $\text{rename} = [\sigma]$ for a $\sigma \in \text{perm}_X$.

Figure 4 with the blue parts removed illustrates a GTA. Both states ℓ_1 and ℓ_2 are initial, denoted by incoming arrows to each of them, and accepting, marked by the double circle. The initial guard is the trivial *true* constraint. The alphabet $\Sigma = \{0, 1\}$ (written in black). The constraint $-x \in I$ is short form for a conjunction of constraints requiring the clock to be in the interval I . For example, if $I = (4, 5]$, then $-x \in I$ is the constraint $4 < -x \wedge -x \leq 5$. During our MITL to GTA translation, we extend GTAs to include outputs (a formal definition is given in [2]). The dagger condition $(-x \in I) ? 1 : 0$ is a short form for two transitions, one which checks $-x \in I$ and outputs 1, and the other which checks $-x \notin I$ and outputs 0.

GTA semantics. A valuation of clocks is a function $v: X \cup \{0\} \mapsto \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ which maps the special clock 0 to 0, history clocks to $\mathbb{R}_{\geq 0} \cup \{+\infty\}$ and future clocks to $\mathbb{R}_{\leq 0} \cup \{-\infty\}$. We denote by $\mathbb{V}(X)$ or simply by \mathbb{V} the set of valuations over X . For a valuation $v \in \mathbb{V}$, define¹ $v \models y - x \triangleleft c$ as $v(y) - v(x) \triangleleft c$. We say that v *satisfies* a constraint φ , denoted as $v \models \varphi$, when v satisfies all the atomic constraints in φ . We denote by $v + \delta$ the *valuation* obtained from valuation v by increasing by $\delta \in \mathbb{R}_{\geq 0}$ the value of all clocks in X . Note that, from a given valuation, not all time elapses result in valuations since future clocks need to stay at most 0. We now define the *change* operation that combines the *reset* operation for history clocks (which sets history clocks to 0) and *release* operation for future clocks (which

¹ To allow evaluation of all the constraints in $\Phi(X)$, the addition and the unary minus operation on real numbers is extended [1] with the following conventions (i) $(+\infty) + \alpha = \alpha + (+\infty) = +\infty$ for all $\alpha \in \overline{\mathbb{R}}$, (ii) $(-\infty) + \beta = \beta + (-\infty) = -\infty$, as long as $\beta \neq +\infty$, and (iii) $-(+\infty) = -\infty$ and $-(-\infty) = +\infty$.

assigns a non-deterministic value to a future clock). Given a set of clocks $R \subseteq X$, we define $R_F = R \cap X_F$ as the set of future clocks in R , and $R_H = R \cap X_H$ as the set of history clocks in R . Then, $[R]v := \{v' \in \mathbb{V} \mid v'(x) = 0 \ \forall x \in R_H \text{ and } v'(x) = v(x) \ \forall x \notin R\}$. Observe that v' has no constraints for the future clocks in R , as they can take any arbitrary value in v' . For a valuation $v \in \mathbb{V}(X)$, and $\sigma \in \text{perm}_X$, we define $[\sigma]v$ as $v \circ \sigma$, i.e., $([\sigma]v)(x) = v(\sigma(x))$ for all $x \in X \cup \{0\}$.

For valuations v, v' and a guard $g \in \Phi(X)$ we write $v \xrightarrow{g} v'$ when $v' = v \models g$, and $v \xrightarrow{[R]} v'$ when $R \subseteq X$ and $v' \in [R]v$, and $v \xrightarrow{[\sigma]} v'$ when $\sigma \in \text{perm}_X$ and $v' = [\sigma]v$. When $\text{prog} = \text{prog}_1; \dots; \text{prog}_n$, we write $v \xrightarrow{\text{prog}} v'$ when there are valuations v_1, \dots, v_n such that $v \xrightarrow{\text{prog}_1} v_1 \xrightarrow{\text{prog}_2} \dots \xrightarrow{\text{prog}_n} v_n = v'$. The semantics of the GTA \mathcal{A} defined above is given by a transition system $\text{TS}_{\mathcal{T}}$ whose states are *configurations* (q, v) of \mathcal{A} , where $q \in Q$ and $v \in \mathbb{V}$ is a valuation. A configuration (q, v) is initial if $v \models g$ for some $(q, g) \in \mathcal{I}$, and it is accepting if $q \in Q_f$. Transitions of $\text{TS}_{\mathcal{T}}$ are of two forms: (1) *delay transition*: $(q, v) \xrightarrow{\delta} (q, v + \delta)$ if $v + \delta$ is a valuation, i.e., $(v + \delta) \models X_F \leq 0$, and (2) *discrete transition*: $(q, v) \xrightarrow{t} (q', v')$ if $t = (q, a, \text{prog}, q') \in \Delta$ and $v \xrightarrow{\text{prog}} v'$. A *finite (respectively infinite) run* ρ of a GTA is a finite (respectively infinite) sequence of transitions from an initial configuration of $\text{TS}_{\mathcal{A}}$: $(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \dots$.

For example, consider the run of GTA in Figure 4 on a timed word $(1, 1)(0, 2)(1, 3)(0, 4) \dots$ (1 occurs at all odd numbers, and 0 at all even numbers, starting from first timestamp 1). The program $(x = 0); [x]$ used in the transitions first checks if x is 0, and then releases it to an arbitrary non-deterministic value. The run on the above word would be: $(\ell_1, x = -1) \xrightarrow{1, t_1} (\ell_2, x = -1) \xrightarrow{1, t_2} (\ell_1, x = -1) \dots$. A transition $\xrightarrow{\delta, t}$ denotes a time elapse of δ followed by application of the program associated to transition t . At each point the value of x is released to -1 , and is checked with the guard $x = 0$ at the next event.

An infinite run is accepting if it visits accepting configurations infinitely often. The run is said to be *Zeno* if $\sum_{i \geq 0} \delta_i$ is bounded and *non-Zeno* otherwise. In this work, we will be interested in *strongly non-Zeno* GTA: these are GTA where every accepting run is non-Zeno. It is possible to convert every GTA into a strongly non-Zeno GTA using a standard construction from timed automata literature [35]. In the rest of the document, we will drop the “strongly non-Zeno” prefix and simply say GTA.

Liveness problem. The *non-emptiness or liveness problem* for a GTA asks whether the given GTA has an accepting non-Zeno run. Due to our assumption about strong non-Zenoness, the question reduces to asking if a given GTA has an accepting run. Unfortunately, the non-emptiness problem even for finite words turns out to be undecidable for general GTA [1]. Therefore, we focus our attention on a restricted sub-class of GTA’s for which non-emptiness in the finite words case is decidable, called safe GTA [1].

► **Definition 1** (Safe GTA [1]). *Given a GTA \mathcal{A} , let $X_D \subseteq X_F$ be the subset of future clocks used in diagonal guards of \mathcal{A} between future clocks, i.e., if $x - y \triangleleft c$ with $x, y \in X_F$ occurs in some guard of \mathcal{A} then $x, y \in X_D$. Then, a program prog is X_D -safe if clocks in X_D are checked for being 0 or $-\infty$ before being released and renamings $[\sigma]$ used in prog preserve X_D clocks ($\sigma(X_D) = X_D$). A GTA \mathcal{A} is safe if it only uses X_D -safe programs on its transitions and the initial guard g_0 sets each history clock to either 0 or ∞ .*

The GTA in Figure 4 is vacuously safe, since there are no diagonal constraints at all.

► **Remark 2.** Renaming operations may be considered as syntactic sugar allowing for more concise representations of GTAs. Indeed, we can transform a GTA \mathcal{A} with renamings to an equivalent GTA \mathcal{A}' without renamings by adding to the state the current permutation

of clocks (composition of the permutations applied since the initial state) and change the programs of outgoing transitions accordingly. The number of states is multiplied by the number of permutations that may occur as described above.

Zones, zone graph and simulations. Reachability for GTA proceeds by an enumeration of its reachable configurations stored as constraint systems called *zones*. A zone over a set of variables $X \cup \{0\}$ is a conjunction of difference constraints $x - y \triangleleft c$ where $x, y \in X \cup \{0\}$, $\triangleleft \in \{<, \leq\}$ and $c \in \mathbb{Z}$. For a zone Z and a valuation v , we write $v \in Z$ if the valuation v satisfies every constraint in Z . Therefore, we also interpret Z as a set of valuations, which satisfy its constraints.

A pair (q, Z) with q a control state and Z a zone represents $\{(q, v) \mid v \in Z\}$. Successors for (q, Z) can be defined based on the outgoing transitions of q . For a transition $t := (q, a, \text{prog}, q')$, we write $(q, Z) \xrightarrow{t} (q', Z_t)$ if $Z_t = \{v' \mid v' \text{ is a valuation and } (q, v) \xrightarrow{t, \delta} (q', v') \text{ for some } v \in Z \text{ and } \delta \in \mathbb{R}_{\geq 0}\}$. It was shown in [1] that the successor Z_t is also a zone. This observation is used to define the notion of the *Zone graph* of a GTA.

► **Definition 3** (GTA zone graph [1]). *Given a GTA \mathcal{A} , its GTA zone graph, denoted $\text{GZG}(\mathcal{A})$, is defined as follows: Nodes are of the form (q, Z) where q is a state and Z is a GTA zone. Initial nodes are pairs (q_0, \vec{Z}_0) where $(q_0, g_0) \in \mathcal{I}$ is an initial condition and Z_0 is given by $g_0 \wedge (X_F \leq 0) \wedge (X_H \geq 0)$ (Z_0 is the set of all valuations which satisfy the initial constraint g_0). For every node (q, Z) and every transition $t := (q, a, \text{prog}, q')$ there is an edge $(q, Z) \xrightarrow{t} (q', Z_t)$ in the GTA zone graph.*

Finally, just as is the case for zone graphs for timed automata, $\text{GZG}(\mathcal{A})$ is not guaranteed to be finite. In order to use it to check Büchi non-emptiness or reachability, we need a finite abstraction of the zone graph. The standard technique to obtain such finite abstractions is using the notion of *simulations*, that we recall next.

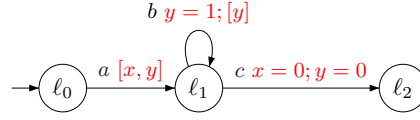
► **Definition 4** (Simulation). *A (time-abstract) simulation relation on the semantics of a GTA is a reflexive, transitive relation $(q, v) \preceq (q, v')$ relating configurations with the same control state and*

1. *for every $\delta \in \mathbb{R}_{\geq 0}$ such that $v + \delta \in \mathbb{V}$ is a valuation, there exists $\delta' \in \mathbb{R}_{\geq 0}$ such that $v' + \delta' \in \mathbb{V}$ is a valuation and $(q, v + \delta) \preceq (q, v' + \delta')$,*
2. *for every transition t , if $(q, v) \xrightarrow{t} (q_1, v_1)$ for some valuation v_1 , then $(q, v') \xrightarrow{t} (q_1, v'_1)$ for some valuation v'_1 with $(q_1, v_1) \preceq (q_1, v'_1)$,*
3. *for all future clocks $x \in X_F$, if $v(x) = -\infty$ then $v'(x) = -\infty$.*

For two GTA zones Z, Z' , we say $(q, Z) \preceq (q, Z')$ if for every $v \in Z$ there exists $v' \in Z'$ such that $(q, v) \preceq (q, v')$.

3 Liveness for GTA

In this section, we will discuss a zone-based procedure to check liveness for safe generalized timed automata. We start by explaining how the standard zone based algorithm for solving liveness in classical timed automata can be adapted to the setting of safe GTAs. The approach for timed automata crucially depends on the existence of a finite time-abstract bisimulation between valuations, namely the region-equivalence [3]. However, there exists no such finite time-abstract bisimulation for GTAs (extension of a result of [19]), as illustrated in Figure 2. The issue is that we cannot forget (abstract) the values of future clocks, unlike history clocks where values above a maximum constant are equivalent. Therefore, our approach involves a significant deviation from the standard one.



■ **Figure 2** Example to illustrate no finite bisimulation in GTA; x is a future clock, y a history clock. The initial transition releases clock x to an arbitrary value, and resets y to 0. From configuration $\langle l_1, x = -n, y = 0 \rangle$ ($n \in \mathbb{N}$), the only way to reach l_2 is by executing $b^n c$, with 1 time unit between consecutive b 's. Therefore, $\langle l_1, x = -n, y = 0 \rangle$ and $\langle l_1, x = -m, y = 0 \rangle$ are simulation incomparable, when $n \neq m$. Hence there is no finite bisimulation.

We fix a safe GTA \mathcal{A} for the rest of this section. We recall that our GTA are strongly non-Zeno, that is, every accepting run is non-Zeno. In order to focus on the main difficulties and avoid additional technicalities, we assume that the GTA \mathcal{A} is without renamings. We start by noting that non-Zeno runs have a special form: future clocks which are not ultimately $-\infty$ should be released infinitely often. If not, there is a last point where a future clock is released to a finite value, and the entire suffix of the run should fall under this finite time, which contradicts non-Zenoness.

► **Lemma 5.** *Let $\rho := (q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \dots$ be a non-Zeno run of the GTA \mathcal{A} . Then, for every future clock x of \mathcal{A} , and for every index $i \geq 0$, if $v_i(x) \neq -\infty$, there exists $j \geq i$ such that x is released in t_j .*

Overview of our solution. In classical timed automata, the liveness problem is solved by enumerating the zone graph, and using a *simulation equivalence* [23, 7, 16, 17] for termination: exploration from (q, Z) is stopped if there exists an already visited node (q, Z') such that $(q, Z) \preceq (q, Z')$ and $(q, Z') \preceq (q, Z)$ for some simulation relation \preceq . In this case a special edge is added between (q, Z) and (q, Z') to indicate a simulation equivalence. There is an (infinite) accepting run iff there is a cycle in the zone graph thus computed, containing an accepting state. The main point is that, from a cycle in the zone graph with simulation equivalences, we can conclude the existence of an infinite run over configurations.

At a high level the proof of this fact is as follows. Let us start by ignoring simulations for the moment: suppose $(q, Z) \xrightarrow{\sigma} (q, Z)$ for a sequence of transitions σ . By definition of successor computation in the zone graph, for every v in the zone Z (on the right), there exists a predecessor u in the zone Z (on the left). Repeatedly applying this argument gives a valuation $u \in Z$ from which σ can be iterated ℓ times, for any $\ell \geq 1$. When ℓ is greater than the number of Alur-Dill regions [3], we get a run $(q, u) \xrightarrow{\sigma^\ell} (q, u')$ such that u and u' are region equivalent. Since the region equivalence is a time-abstract bisimulation, this shows that we can once again do σ^ℓ from (q, u') , and so on. This leads to an infinite run from (q, u) where σ can be iterated infinitely often. Now, when simulations are involved, we need to consider sequences of the form $(q, Z) \xrightarrow{\sigma} (q, Z')$ where $(q, Z) \preceq (q, Z')$ and $(q, Z') \preceq (q, Z)$. An argument similar to the above can be adapted in this case too [28, 24, 22]. The critical underlying reason that makes such an argument possible is the presence of a finite time-abstract bisimulation, which in timed automata, is given by the region equivalence.

The same idea cannot be directly applied in the GTA setting, as there is no finite time-abstract bisimulation for GTAs, even with the safety assumption (Figure 2). However, [1] have defined a finite equivalence $v_1 \sim_M v_2$ and shown that the downward closures of the reachable zones w.r.t. a certain simulation called the **G** simulation [16, 17] are unions of \sim_M equivalence classes. Therefore, applying an argument of the above style will give us a run $(q, u) \xrightarrow{\sigma^\ell} (q, u')$ such that $u \sim_M u'$. But we cannot conclude an infinite run immediately as \sim_M is not a bisimulation.

To circumvent this problem, we will define an equivalence \approx_M which is in spirit like the region equivalence in timed automata. As expected, \approx_M will be a bisimulation. However, in accordance with the no finite timed-bisimulation result, \approx_M will have an infinite index. We make a key observation: if we have a run $(q, u) \xrightarrow{\sigma} (q, u')$ such that $u \sim_M u'$ and if σ releases every future clock, then we can get a run $(q, u) \xrightarrow{\sigma} (q, u'')$ where $u \approx_M u''$ for a suitably modified valuation u'' . Since \approx_M is a bisimulation, this will then give an infinite run where σ can be iterated infinitely often. As we have seen from Lemma 5, if we are interested in non-Zeno runs, only such cycles where all future clocks are released (or remain $-\infty$) are relevant. Therefore, in order to decide liveness for safe GTAs, it suffices to construct the zone graph with the simulation equivalence edges and look for a reachable cycle that contains an accepting state such that for every future clock x , either x is released on the cycle, or valuation $-\infty$ is possible for clock x .

This section is organized as follows: we will first define the equivalence \approx_M and show that it is a bisimulation; then we recall \sim_M , and prove the key observation mentioned above. One of the main challenges is in addressing diagonal constraints, which is exactly where the safety assumption is helpful.

A region-like equivalence for GTA. The definition of \approx_M looks like the classical region equivalence extended from $[0, +\infty)$ to $\overline{\mathbb{R}}$: all clocks which are lesser than M (which automatically includes all future clocks) have the same integral values, and the ordering of fractional parts among these clocks is preserved. To account for diagonal constraints in guards, we explicitly add a condition to say that all allowed diagonal constraints are satisfied by equivalent valuations. This new equivalence does not have a finite index, but it turns out to be a time-abstract bisimulation, similar to the classical regions.

- **Definition 6.** Let $v_1, v_2 \in \mathbb{V}$ be valuations. We say $v_1 \approx_M v_2$ if for all clocks x, y :
1. $v_1(x) \triangleleft c$ iff $v_2(x) \triangleleft c$ for all $\triangleleft \in \{<, \leq\}$ and $c \in \{-\infty, +\infty\}$ or $c \in \mathbb{Z}$ with $c \leq M$,
 2. $v_1 \models x - y \triangleleft c$ iff $v_2 \models x - y \triangleleft c$ for all $\triangleleft \in \{<, \leq\}$ and $c \in \{-\infty, +\infty\}$ or $c \in \mathbb{Z}$ with $|c| \leq M$,
 3. if $-\infty < v_1(x), v_1(y) \leq M$ then we have $\{v_1(x)\} \leq \{v_1(y)\}$ iff $\{v_2(x)\} \leq \{v_2(y)\}$.

Notice that when $v_1 \approx_M v_2$, the first condition implies $v_1(x) = +\infty$ iff $v_2(x) = +\infty$, $v_1(x) = -\infty$ iff $v_2(x) = -\infty$, $-\infty < v_1(x) \leq M$ iff $-\infty < v_2(x) \leq M$, and in this case $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$ and $\{v_1(x)\} = 0$ iff $\{v_2(x)\} = 0$.

► **Lemma 7.** \approx_M is a time-abstract bisimulation.

The equivalence \sim_M , and moving from \sim_M to \approx_M . The equivalence \sim_M is defined on the space of all valuations. Our goal in this part is to start from $v_1 \sim_M v_2$ and generate a valuation v'_2 by modifying some values of v_2 , so that we get $v_1 \approx_M v'_2$. Let us first recall the definition of \sim_M , with n be the number of clocks in the GTA.

- First, we define \sim_M on $\alpha, \beta \in \overline{\mathbb{R}}$ by $\alpha \sim_M \beta$ if $(\alpha \triangleleft c \iff \beta \triangleleft c)$ for all $\triangleleft \in \{<, \leq\}$ and $c \in \{-\infty, +\infty\}$ or $c \in \mathbb{Z}$ with $|c| \leq M$. In particular, $\alpha \sim_M \beta$ implies $\alpha = -\infty$ iff $\beta = -\infty$ and $\alpha = +\infty$ iff $\beta = +\infty$. Also, if $-M \leq \alpha \leq M$ then $\alpha \sim_M \beta$ implies $\lfloor \alpha \rfloor = \lfloor \beta \rfloor$ and $\{\alpha\} = 0$ iff $\{\beta\} = 0$.
- For valuations $v_1, v_2 \in \mathbb{V}$ we define $v_1 \sim_M v_2$ if (i) $v_1(x) \sim_{nM} v_2(x)$ for all $x \in X$, and (ii) $v_1(x) - v_1(y) \sim_{(n+1)M} v_2(x) - v_2(y)$ for all pairs of clocks $x, y \in X$.

Notice that \approx_M and \sim_M are incomparable, in the sense that neither of them is a refinement of the other. The equivalence \approx_M constrains values up to M , whereas \sim_M looks at values up to nM , i.e., between $-nM$ and nM . For instance, consider $v_1 := \langle x = M + 2, y = 1 \rangle$ and

$v_2 := \langle x = M + 3, y = 1 \rangle$ for some $M \geq 2$. We have $v_1 \approx_M v_2$, but $v_1 \not\sim_M v_2$. For the other way around, notice that \sim_M has finite index, whereas \approx_M does not. So, $v_1 \sim_M v_2$ does not imply $v_1 \approx_M v_2$. To see it more closely, $v_1 \approx_M v_2$ enforces the same integral values for all future clocks. For clocks less than $-(n + 1)M$, there is no such constraint on the actual values in \sim_M .

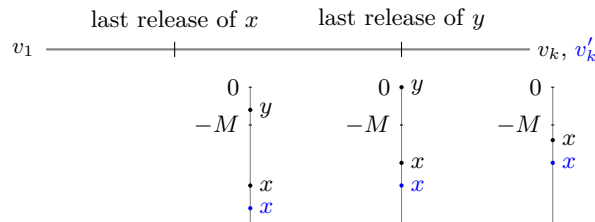
As mentioned above, our objective is to obtain \approx_M equivalent valuations starting from \sim_M equivalent ones. Lemma 8 is a first step in this direction. It essentially shows that, when restricted to clocks within $-M$ and $+M$, \sim_M entails \approx_M .

► **Lemma 8.** *Suppose $v_1 \sim_M v_2$. Let x, y be clocks such that $-M \leq v_1(x), v_1(y) \leq M$. Then, $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$, $\{v_1(x)\} = 0$ iff $\{v_2(x)\} = 0$, and $\{v_1(x)\} \leq \{v_1(y)\}$ iff $\{v_2(x)\} \leq \{v_2(y)\}$.*

Lemma 8 considers clocks within $-M$ and $+M$. What about clocks above M ? Directly from $v_1 \sim_M v_2$, we have $M < v_1(x)$ iff $M < v_2(x)$, and moreover diagonal constraints up to M are already preserved by \sim_M . Therefore, together with Lemma 8, $v_1 \sim_M v_2$ implies $v_1 \approx_M v_2$ when restricted to clocks greater than $-M$. We cannot say the same for clocks lesser than $-M$, in particular we may have $v_1(x) = -nM - 1$ and $v_2(x) = -nM - 2$. However, as shown in the lemma below, we can choose suitable values for clocks lesser than $-M$ to get a \approx_M -equivalent valuation from a \sim_M -equivalent one.

► **Lemma 9.** *Suppose $v_1 \sim_M v_2$, and let $L = \{x \mid -M \leq v_1(x)\}$. There is a valuation v'_2 such that $v'_2 \downarrow_L = v_2 \downarrow_L$ and $v_1 \approx_M v'_2$.*

Finally, we show that if we have a run between \sim_M equivalent valuations, we can extract a run between \approx_M equivalent valuations, simply by changing the last released values of future clocks. Suppose there is a run ρ from configuration (q, v_1) to configuration (q, v_k) such that $v_1 \sim_M v_k$, and all future clocks are released in ρ . By Lemma 9, there is a v'_k satisfying $v_1 \approx_M v'_k$ that differs from v_k only in the clocks that are less than $-M$. In order to reach v'_k from v_1 using the same sequence of transitions as in ρ , it is enough to choose a suitable shifted value during the last release of the clocks that were modified. This gives a new run ρ' . The non-trivial part is to show that ρ' is indeed a run, that is: all guards are satisfied by the new values. We depict this situation in Figure 3. The modified clocks are those that are less than $-M$ in v_k . Clock x is one such. The black dot represents its value in v_k , and the blue dot is its value in v'_k . Its new value is still $< -M$. In the run ρ' , clock x is released to a suitably shifted value at its last release point. Notice that from this last release point till k , clock x stays below $-M$ in both ρ and ρ' . Therefore, all non-diagonal constraints $x \triangleleft c$ that were originally satisfied in ρ continue to get satisfied in ρ' . Showing that all diagonal constraints are still satisfied is not as easy. Here, we make use of the safety assumption. Let us look at a diagonal constraint $x - y$, and a situation as in Figure 3 where the last release of y happens after the last release of x . For simplicity, let us assume there is no release of y in between these two points.



■ **Figure 3** An illustration for the proof of Lemma 10.

We divide the run into three parts: the left part is the one before the last release of x , the middle part is the one between the two release points, and the right part is the rest of the run, to the right of the release of y . In the left part, the values of x and y are the same in both ρ and ρ' , and so the diagonal constraints continue to get satisfied. In the right part, the value of $x - y$ equals $v'_k(x) - v'_k(y)$. Using $v'_k \approx_M v_1$ and $v_1 \sim_M v_k$, we can argue that v'_k and v_k satisfy the same diagonal constraints up to constant M . This takes care of the right part. The middle part is the trickiest. In this part, we know that x remains less than $-M$ in both ρ and ρ' . The value of y is the same in both ρ and ρ' . But what about the difference $x - y$? Can it be, say -1 in ρ and -2 in ρ' ? Here is where we use the safety assumption to infer the value of $x - y$. Before y is released, its value should be 0. At that point, x is still less than $-M$ (in both the runs). Therefore $x - y < -M$ just before y is last released. As the differences do not change, we see that $x - y < -M$ in the middle part, for both runs. Hence the diagonal constraints continue to hold in ρ' . We formalize these observations in Lemma 10, where we exhaustively argue about all the different cases.

► **Lemma 10.** *Consider a safe GTA \mathcal{A} . Let $\rho : (q_1, v_1) \xrightarrow{\delta_1, t_1} (q_2, v_2) \xrightarrow{\delta_2, t_2} \dots (q_k, v_k)$ be a run of \mathcal{A} such that $v_1 \sim_M v_k$ and for every future clock x , either x is released in the transition sequence $t_1 \dots t_{k-1}$ or $v_1(x) = -\infty$. Let $L = \{x \mid -M \leq v_1(x)\}$. Let v'_k be a valuation such that $v'_k \downarrow_L = v_k \downarrow_L$ and $v_1 \approx_M v'_k$. Then, there exists a run of the form $\rho' : (q_1, v_1) = (q_1, v'_1) \xrightarrow{\delta_1, t_1} (q_2, v'_2) \xrightarrow{\delta_2, t_2} \dots (q_k, v'_k)$ in \mathcal{A} , leading to (q_k, v'_k) from (q_1, v_1) .*

We lift this argument to the level of zones, to obtain one of the main results of this paper.

► **Theorem 11.** *Let $(q, Z) = (q_1, Z_1) \xrightarrow{t_1} (q_2, Z_2) \xrightarrow{t_2} \dots \xrightarrow{t_{k-1}} (q_k, Z_k) = (q, Z')$ be a run in the zone graph such that $(q, Z) \preceq (q, Z')$, $(q, Z') \preceq (q, Z)$ and for every future clock x , either x is released in the sequence $t_1 \dots t_{k-1}$, or there is a valuation $v_x \in Z'$ with $v_x(x) = -\infty$. Then, there is a valuation $v \in Z$ and an infinite run starting from (q, v) over the sequence of transitions $(t_1 \dots t_{k-1})^\omega$.*

Finally, combining Theorem 11 and Lemma 5, we get an algorithm for liveness: we construct the zone graph with simulation equivalence and check for a reachable cycle that contains an accepting state and where every future clock x which is not released during the cycle may take value $-\infty$ in some valuations of the zones in the cycle.

4 Translating MITL to GTA

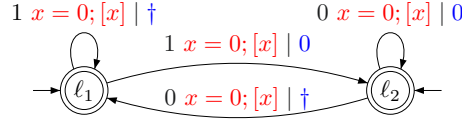
We first introduce the preliminaries for Metric Interval Temporal Logic. Let **Prop** be a finite nonempty set of atomic propositions. The alphabet Σ that we consider is the set of subsets of **Prop**. The set of MITL formulae over the set of atomic propositions **Prop** is defined as

$$\varphi := p \mid \varphi \wedge \varphi \mid \neg \varphi \mid \mathbf{X}_I \varphi \mid \varphi \mathbf{U}_I \varphi$$

where $p \in \mathbf{Prop}$, and I is either $[0, 0]$, or a non-singleton (open, or closed) interval whose end-points come from $\mathbb{N} \cup \{\infty\}$. In other words, if the end-points of the interval are a and b respectively, then either $a = b = 0$, or $a, b \in \mathbb{N} \cup \{\infty\}$ and $a < b$.

We will now define the *pointwise semantics* of MITL formulae inductively as follows. A timed word $w = (a_0, \tau_0)(a_1, \tau_1)(a_2, \tau_2) \dots$ is said to satisfy the MITL formula φ at position $i \geq 0$, denoted as $(w, i) \models \varphi$ if (omitting the classical Boolean connectives)

- $(w, i) \models p$ if $p \in a_i$
- $(w, i) \models \mathbf{X}_I \varphi$ if $(w, i + 1) \models \varphi$ and $\tau_{i+1} - \tau_i \in I$.
- $(w, i) \models \varphi_1 \mathbf{U}_I \varphi_2$ if there exists $j \geq i$ s.t. $(w, j) \models \varphi_2$, $(w, k) \models \varphi_1$ for all $i \leq k < j$, and $\tau_j - \tau_i \in I$.



■ **Figure 4** GTT for X_I using a future clock x . The output is 0 for transitions to location ℓ_2 and is the if-then-else $\dagger = (-x \in I) ? 1 : 0$ for transitions to location ℓ_1 , where I is some interval.

Our goal is to construct a GTA with outputs for an MITL formula φ , which reads the timed word and outputs 1 at position i iff $(w, i) \models \varphi$. More precisely, there is a unique run of the GTA on w : $(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \dots$, where the output of each transition t_i equals 1 iff $(w, i) \models \varphi$. We refer to GTA with outputs as Generalized Timed Transducers (GTT) (discussed in detail in the full version [2]). At a high level, our construction can be viewed as structural induction on the *parse tree* of the MITL formula, where we build a GTT for atomic propositions, and then for each Boolean and temporal operator, and finally we compose these GTT bottom up to obtain the GTT for each subformula, which by structural induction finally gives us the GTT for the full formula. A detailed discussion of this compositional approach can be found in the full version [2]. We describe the transducers for X_I and U_I in this section.

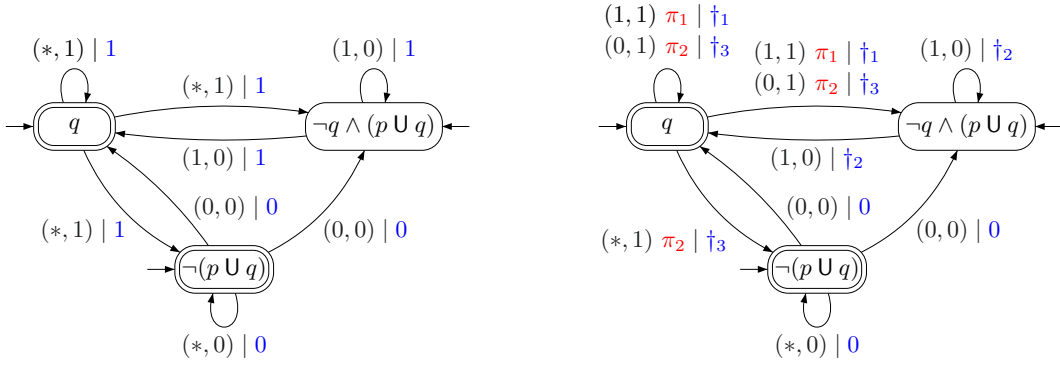
Next operator. The transducer for $X_I p$ is given in Figure 4. It is obtained by extending the untimed variant of the Next-transducer with a future clock x that predicts the time to the next event. The idea is the same as explained in Figure 1 of the Introduction. The prediction of the next event is verified, by having the guard $x = 0$ in every transition. Notice the use of the program syntax in this example: a transition first checks if $x = 0$ (satisfying a previous obligation), and then releases x to a non-deterministic value guessing the time to the next event, and then asks for a guard, either $-x \in I$ or $-x \notin I$.

Until operator. We start by describing the transducer for the untimed U modality $p U q$ (in other words, $p U_I q$ with $I = [0, \infty)$). This is shown in Figure 5. For simplicity, we have assumed $\text{Prop} = \{p, q\}$ and the alphabet is represented as $(0, 0), (1, 0), (0, 1), (1, 1)$ corresponding to $\{\}, \{p\}, \{q\}$ and $\{p, q\}$. On the word w , if s_i is the state that reads a_i , then the following invariants hold:

- $s_i = q$ iff $q \in a_i$,
- $s_i = \neg q \wedge (p U q)$ iff $q \notin a_i$ and $(w, i) \models p U q$,
- $s_i = \neg(p U q)$ iff $(w, i) \not\models p U q$.

At the initial state the automaton makes a guess about position 0, and then subsequently on reading every a_i , it makes a guess about position $i + 1$ and moves to the corresponding state. The transitions implement this guessing protocol. For instance, transitions out of state q read letters with $q = 1$, and also output 1; transitions out of state $\neg(p U q)$ have output 0. A noteworthy point is that state $q \wedge \neg(p U q)$ is non-accepting, preventing the automaton to stay in that state forever. For every word, the transducer has a unique accepting run and the output at position i is 1 iff $(w, i) \models p U q$.

Let us move on to the timed until U_I . Let us forget the specific interval I for the moment. We will come up with a generic construction, on which the outputs can be appropriately modified for specific intervals. To start the construction, we need the following notion.



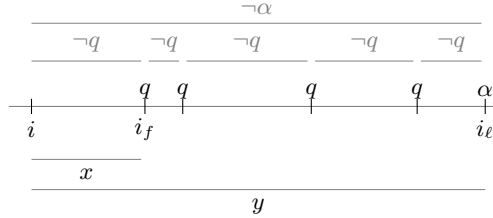
■ **Figure 5** (Left) Transducer for the untimed LTL operator $p \text{ U } q$. (Right) Transducer \mathcal{A} tracking the earliest and last q witnesses for $p \text{ U } q$. Program π_1 is $x = 0; [x]$ and program π_2 is $x = 0; y = 0; [x, y]$. The outputs \dagger_i depend on interval I in the timed until $p \text{ U }_I q$.

▶ **Definition 12.** Let $w = (a_0, t_0)(a_1, t_1) \dots$ be a timed word and let $i \geq 0$. The earliest q -witness at position i is the least position $j > i$ such that $q \in a_j$, if it exists. We denote this position j giving the earliest q -witness at i as i_f . The last q -witness is the least position $j > i$ that satisfies

$$\alpha = q \wedge \neg(p \wedge X(p \text{ U } q)) \equiv (q \wedge \neg p) \vee (q \wedge X \neg(p \text{ U } q))$$

We denote this position j giving the last q -witness at i as i_ℓ .

The earliest and last q -witnesses provide a convenient mechanism to check $p \text{ U }_I q$ which, in many cases, can be deduced by knowing the time to the earliest and last witnesses. Figure 6 illustrates the interpretation of x and y .



■ **Figure 6** Division of q events, and interpretation of x, y .

Our next task is to extend the U transducer of Figure 5 to include two future clocks x and y that predict at each i , the time to i_f and i_ℓ , respectively. Figure 5 describes the transducer \mathcal{A} . For clock x to maintain time to i_f at each position i , we can do the following: at every transition that reads q , the transducer checks for $x = 0$ as guard and releases x (with the time to the next q). If there is no such q , then x needs to be released to $-\infty$ in order to continue the run, as our timed words are non-Zeno. Transitions satisfying $\neg q$ do not check for a guard on x or release x . Therefore, in any run, the value of x determines the time to the next q event.

In Figure 6, the last witness (property α) can be identified by transitions of the form $(0, 1)$ (signifying $q \wedge \neg p$) and transitions $(*, 1)$ going to state $\neg(p \text{ U } q)$ (for $q \wedge X \neg(p \text{ U } q)$). Similar to the previous case of the earliest witness, every time we see such a transition we check for $y = 0$ as a guard and release y . No other transition checks or updates y . Notice

that only the transitions with q have been changed. All transitions $(0, 1)$ check and release both clocks (program π_2). Transitions $(1, 1)$ that do not go to $\neg(p \cup q)$ check and release only x (program π_1), whereas the $(*, 1)$ transition that goes to $\neg(p \cup q)$ does π_2 .

► **Lemma 13.** *For every timed word $w = (a_0, \tau_0)(a_1, \tau_1) \cdots$, there is a unique run of \mathcal{A} of the form: $(s_0, v_0) \xrightarrow{\tau_0, \theta_0} (s_1, v_1) \xrightarrow{\tau_1 - \tau_0, \theta_1} \cdots$ such that for every position $i \geq 0$: (1) s_i is state q of \mathcal{A} iff $w, i \models q$, (2) s_i is state $\neg q \wedge (p \cup q)$ iff $w, i \models \neg q \wedge (p \cup q)$, (3) s_i is state $\neg(p \cup q)$ iff $w, i \models \neg(p \cup q)$, (4) $v_i(x) = \tau_i - \tau_{i_f}$ and $v_i(y) = \tau_i - \tau_{i_\ell}$.*

Using \mathcal{A} we can already answer $p \cup_I q$ for one-sided intervals: $[0, c]$, $[0, c)$, $[b, +\infty)$, $(b, +\infty)$, for natural numbers b, c .

- if $0 \in I$: $\dagger_1 = \dagger_3 = 1$ (current position is a witness), and $\dagger_2 = (-x \in I \vee -y \in I) ? 1 : 0$,
- if $0 \notin I$: $\dagger_3 = 0$, and $\dagger_1 = \dagger_2 = (-x \in I \vee -y \in I) ? 1 : 0$.

This is because in one-sided intervals, if at all there is a witness, the earliest or the last is one of them.

Until with a non-singular interval. We will now deal with the case of intervals $I = [b, c]$ with $0 < b < c < \infty$. Firstly, using x and y , some easy cases of $p \cup_I q$ can be deduced. Output remains 0 for transitions starting from $\neg(p \cup q)$. For other transitions, here are some extra checks:

- if $-x \in I$ or $-y \in I$, output 1 (one of the earliest or last witness is also a witness for $p \cup_I q$),
- else, if $-y < b$ or $c < -x$, output 0 (the time to the last witness is too small or the time to the earliest witness is too large, so there is no witness within I).

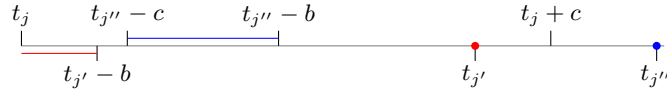
If neither of the above cases hold, then we need guess a potential witness within $[b, c]$ and verify it. This requires substantial book-keeping which we will now explain. Assume we are given a timed word $w = (a_0, t_0)(a_1, t_1) \cdots$. Let us call $j \geq 0$ a *difficult point* if:

$$w, j \models p \cup q \text{ and } t_{j_f} < t_j + b \text{ and } t_j + c < t_{j_\ell}$$

This leaves the possibility for a q -witness within $[b, c]$. So, for difficult points, we need to make a prediction whether we have a q -witness within $[t_j + b, t_j + c]$: guess a time to a witness within $[t_j + b, t_j + c]$ and check it. We cannot keep making such predictions for every difficult point as we have only finitely many clocks. Therefore, we will guess some special witnesses. First we state a useful property.

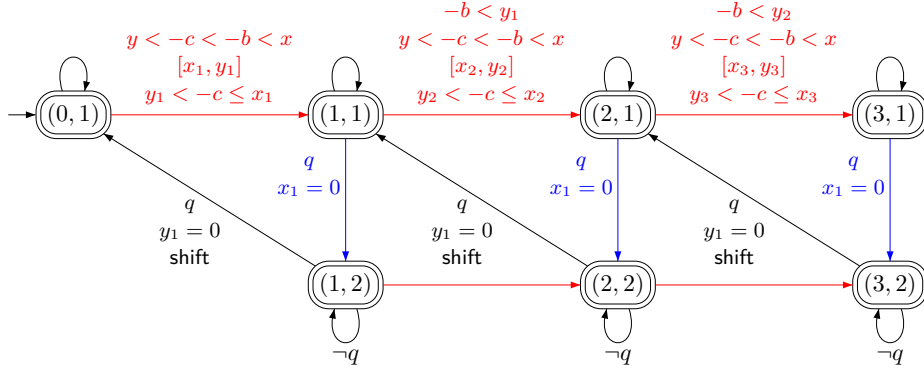
► **Lemma 14.** *Let j be a difficult point. Then, for all k such that $j \leq k \leq j_\ell$, we have $w, k \models p \cup q$.*

Therefore, automaton \mathcal{A} stays in the top two states, while reading j upto j_ℓ .



■ **Figure 7** Illustration of a point j . The point j' is the last q -witness before $t_j + c$, and j'' is the first q -witness after $t_j + c$.

We will now come back to the idea of choosing special witnesses. This is illustrated in Figure 7. For a point j , we let $j' \geq j$ be the greatest position containing q such that $t_{j'} \leq t_j + c$. Let $j'' > j$ be the least position containing q such that $t_j + c < t_{j''}$. So, no



■ **Figure 8** The automaton \mathcal{B} for predicting q -witnesses which are not given by the earliest and latest. For clarity, not every transition is indicated. All clocks are future clocks.

position $j' < k < j''$ contains q . While reading a difficult point j , let us make use of fresh clocks x_1 and y_1 to predict these two witnesses:

$$x_1 = t_j - t_{j'} \quad y_1 = t_j - t_{j''}$$

For the next important observation, we will once again take the help of Figure 7. Notice that for all points i with $t_i \in [t_j, t_{j'} - b]$, the point j' is also a witness for w , $i \models p \mathbf{U}_{[b,c]} q$. Similarly, j'' is a witness for all i such that $t_i \in [t_j, t_{j''} - c]$. Therefore for all i such that $t_i \in [t_j, t_{j'} - b]$, we have a way to determine the output: it is 1 iff while reading a_i we have $-x_1 \in I$ or $-y_1 \in I$ (recall we have predicted x_1 and y_1 while reading a_j as explained above). So, we do not have to make new guesses at the difficult points in $[t_j, t_{j'} - b]$. After $t_{j'} - b$ (which can be identified with the constraint $-b < y_1$), we need to make new such guesses, using fresh clocks, say x_2, y_2 . We will call the difficult points where we start new guesses as *special difficult points*. Notice that the distance between two special difficult points is at least $c - b$ (which is ≥ 1 , as we consider non-singular intervals with bounds in \mathbb{N}). In the figure, if j is a special point, a new special point will be opened later than $t_{j''} - b$.

This gives a bound on the number of special points that can be open between j and j'' . Suppose $j < \ell_1 < \ell_2 < \dots < \ell_i < j''$ be the sequence of special points between j and j'' . Since ℓ_1 is opened when time to j'' is at most b , we get the inequality: $t_{\ell_i} - t_{\ell_1} < b$. Since consecutive special points are at least $c - b$ apart, we have $(i - 1)(c - b) < b$. This entails $i < 1 + \lceil \frac{b}{c - b} \rceil$. By the time we reach j'' , we need to have opened at most $k = 1 + \lceil \frac{b}{c - b} \rceil$ special points, and hence we can work with the extra clocks $x_1, y_1, x_2, y_2, \dots, x_k, y_k$.

All these ideas culminate in a book-keeping automaton \mathcal{B} to handle difficult points. Its set of states is $\{0, 1, \dots, N\} \times \{1, 2\}$ where $N = 1 + \lceil \frac{b}{c - b} \rceil$ (state $(0, 2)$ is not reachable). All states are accepting. This is shown in Figure 8 for $N = 3$. The automaton \mathcal{B} synchronizes with \mathcal{A} (via a usual cross-product synchronized on transitions). All transitions of \mathcal{B} other than the self loop on state $(0, 1)$ satisfy p . Transitions which satisfy q , and $\neg q$ are specifically marked in the figure.

The automaton \mathcal{B} starts in the initial state $(0, 1)$. It moves to $(1, 1)$ on the first difficult point j (which will become special) and comes back to $(0, 1)$ when there are no active special difficult points waiting for witnesses (a special difficult point j is active at positions $j \leq i \leq j''$). States $(i, 1)$ in the top indicate that there are i active special difficult points currently. A state $(i, 2)$ indicates that the j' witness for the oldest active point has been seen, and we are waiting for its j'' witness (the space between $t_{j'}$ and $t_{j''}$ in Figure 7).

The red transitions $(i, *) \rightarrow (i+1, *)$ open new special difficult points, and contain the program as illustrated in the figure. At $(i, 1)$, suppose $\ell_1 < \ell_2 < \dots < \ell_i$ are the active special difficult points where we have predicted $x_1, y_1, \dots, x_i, y_i$ respectively. We have the invariant:

$$y_i < x_i \leq y_{i-1} < x_{i-1} \leq \dots \leq y_2 < x_2 \leq y_1 < x_1$$

Notice that we may have $x_i = y_{i-1}$: the “first” witness of the i^{th} special point (ℓ'_i) could coincide with the “second” witness of the $(i-1)^{\text{th}}$ point (ℓ'_{i-1}). This leads to certain subtleties, which we will come to later.

The blue transitions read the witness for the oldest active special point (that is, we have reached ℓ'_1). Observe that $x_1 = 0$ does not immediately identify ℓ'_1 , since there could be a sequence of positions at the same time, and ℓ'_1 is the last of them. Therefore, we make a non-deterministic choice whether to take the blue transition (implying that ℓ'_1 has been found), or we remain in the same state. The blue transitions read a q , check $x_1 = 0$, and then releases x_1 to $-\infty$ (not shown in Figure 7). The black (diagonal) transitions witness ℓ''_1 . When this happens, x_1, y_1 are no longer useful, and therefore all the higher clocks are shifted using the permutation shift which maps $x_2, y_2, \dots, x_k, y_k, x_1, y_1$ to $x_1, y_1, \dots, x_k, y_k$ and keeps the other clocks unchanged.

There are some subtleties which arise when special points coincide with witness points, or when the second witness of a special point coincides with the first witness of the consecutive special point.

Subtleties. The first subtlety arises when we have $\ell''_j = \ell'_{j+1}$ for consecutive special points. This will imply $y_j = x_{j+1}$. The reverse direction is not true, as there could be a sequence of positions with the same time, but let us assume we have dealt with it by the non-deterministic choice. When we actually witness these points, the clock values would have shifted to lower indices. This situation will be manifested as $y_1 = x_2 = 0$. Suppose we are in $(i, 2)$ and see a point ℓ''_j ($y_1 = 0$). The diagonal transition takes the automaton to $(i-1, 1)$ and shifts x_2 to x_1 . Now, $x_1 = 0$ (as $\ell'_{j+1} = \ell''_j$). Therefore, we will have to combine the black-diagonal-left with the downward-blue to get the combined effect. This leads to these two divisions:

$$(i, 2) \xrightarrow{y_1=0} (i-1, 1) \qquad (i, 2) \xrightarrow{y_1=0 \wedge x_2=0} (i-1, 2)$$

The second subtlety is that one of either ℓ'_j or ℓ''_j witnesses be a new special point (notice that the red transitions are independent of the blue and black transitions). In such cases, we can combine the two effects in any order: first discharge x_1 or y_1 verification, and then open a new special point or vice-versa. This leads to some additional divisions of the form:

$$(i, 1) \xrightarrow{x_1=0 \wedge -b < y_i} (i+1, 2) \qquad (i, 2) \xrightarrow{y_1=0 \wedge -b < y_i} (i, 1)$$

In the first transition, we have combined a blue and a red (in any order); whereas in the second, we have combined a red and a black-diagonal, in any order.

The third subtlety is that the first and second subtleties may occur together! A point could be ℓ''_j, ℓ'_{j+1} and also a new special point. We illustrate this on a specific state $(i, 2)$. We provide only the “guards”. The full program is obtained by suitably combining the effects of the individual transitions:

$$\begin{aligned} (i, 2) &\xrightarrow{y_1=0 \wedge y_i \leq -b} (i-1, 1) & (i, 2) &\xrightarrow{y_1=0 \wedge -b < y_i} (i, 1) \\ (i, 2) &\xrightarrow{y_1=0 \wedge x_2=0 \wedge y_i \leq -b} (i-1, 2) & (i, 2) &\xrightarrow{y_1=0 \wedge x_2=0 \wedge -b < y_i} (i, 2) \end{aligned}$$

This concludes the description of the automaton \mathcal{B} . The product $\mathcal{A} \times \mathcal{B}$ gives the required transducer for $p \text{ U}_I q$. The full construction of \mathcal{B} , taking into account all these subtleties, is described as Algorithm 1. In comments, we use the terminology introduced before and we also refer to the color of transitions in Figure 8. Parsing the pseudo-code from a current state (k, m) results in a sequence of guards and releases, an output of a Boolean value (**output** value), and the next state (**goto** (k', m')). The most difficult case is for states $(k, 2)$ with $k \geq 2$, where we could generate transitions to states $(k-1, 1), (k-1, 2), (k, 1), (k, 2), (k+1, 2)$.

Complexity and comparison with the MightyL approach. The final automaton $\mathcal{A} \times \mathcal{B}$ has at most $6k$ states, where $k = 1 + \lceil \frac{b}{c-b} \rceil$ as defined above: automaton \mathcal{A} has 3 states, and automaton \mathcal{B} has $2k - 1$ states (see Figure 8). In terms of clocks, \mathcal{A} has 2 future clocks x, y , and \mathcal{B} has $2k$ future clocks $x_1, y_1, \dots, x_k, y_k$. We have used a permutation operation **shift**. As we mention in Remark 2, renamings can be eliminated by maintaining in the current state the composition of permutations applied since the initial state. Since each permutation does a cyclic shift, in any composition, the clocks $x_1, y_1, \dots, x_k, y_k$ are renamed to some $x_i, y_i, \dots, x_k, y_k, x_1, y_1, \dots, x_{i-1}, y_{i-1}$. Therefore, there are at most k renamings. Maintaining them in states gives rise to at most $\mathcal{O}(k^2)$ states.

In contrast, the state-of-the-art approach [11] starts with a 1-clock alternating timed automata for U_I . After reading a timed word, the 1-ATA reaches a configuration containing several state-valuation pairs (q, v) . A finite abstraction of this set of configurations, called the interval semantics, has been proposed [9, 10, 11]. This abstraction is maintained in the states. Overall, the number of locations for $p \text{ U}_I q$ is exponential in k , and the number of clocks is $2k + 2$.

Due to the presence of future clocks, we are able to make predictions, as in Figure 7 and the GTA syntax enables concisely checking these predictions in the transitions. Therefore, we are able to give a direct construction to the final automaton, instead of going via an alternating automaton and then abstracting it.

5 Conclusion

In this paper, we have answered two problems: (1) liveness of GTA and (2) MITL model checking using GTA. The solution to the first problem required to bypass the technical difficulty of having no finite time-abstract bisimulation for GTAs. The presence of diagonal constraints adds additional challenges. For MITL model checking using GTA, we have described the GTA for the X_I and U_I modalities. Indeed, the presence of future clocks allows to make predictions better and we see an exponential gain over the state-of-the-art, in the number of states of the final automaton produced. Moreover, our construction is direct, without having to go via alternation.

The next logical step would be to implement these ideas and see how they perform in practice, and compare them with existing well-engineered tools (e.g., [11]). This will require a considerable implementation effort, needing several optimizations and incorporating of many practical considerations before it can become scalable. This provides tremendous scope for future work on these lines.

■ **Algorithm 1** Automaton \mathcal{B} (synchronized with \mathcal{A}).

```

1: State  $(0, 1)$ : ▷ initial state of  $\mathcal{B}$ 
2:   if  $\mathcal{A}$  at state  $\neg(p \cup q)$  then output 0; goto  $(0, 1)$  end if
3:   if  $\neg x \in I$  or  $\neg y \in I$  then output 1; goto  $(0, 1)$  end if
4:   if  $x < -c$  or  $-b < y$  then output 0; goto  $(0, 1)$  end if
5:   Release  $[x_1, y_1]$  ▷ Special difficult point
6:   Check  $y_1 < -c \leq x_1$ 
7:   output  $(x_1 \leq -b)$  ▷ Boolean value
8:   goto  $(1, 1)$  ▷ red transition
9: State  $(k, 1)$  with  $k > 0$ : ▷ waiting for the event predicted by  $x_1$ 
10:   $k' \leftarrow k$ 
11:  if  $y_k \leq -b$  then
12:    output  $(x_k \in I) \vee (y_k \in I)$  ▷ Boolean value
13:  else
14:    if  $-c \leq y$  then ▷ not a difficult point
15:      output  $(y \leq -b)$  ▷ Boolean value ( $y \in I$ )
16:    else ▷ new special difficult point, red transition,
17:      ▷ possibly combined with a blue transition below
18:       $k' \leftarrow k + 1$ ;
19:      Release  $[x_k, y_k]$ ; Check  $y_k < -c \leq x_k$ 
20:      output  $(x_k \leq -b)$  ▷ Boolean value
21:    end if
22:  end if
23:  choose non-deterministically
24:    when True do goto  $(k', 1)$  ▷ not the event predicted by  $x_1$ 
25:    when  $q \wedge (x_1 = 0)$  do Release  $[x_1]$ ;  $x_1 = -\infty$ ; goto  $(k', 2)$  ▷ blue transition
26:  end choose
27: State  $(k, 2)$  with  $k > 0$ : ▷ waiting for the event predicted by  $y_1$ 
28:   $k' \leftarrow k$ 
29:  if  $y_k \leq -b$  then
30:    output  $(x_k \in I) \vee (y_k \in I)$  ▷ Boolean value
31:  else
32:    if  $-c \leq y$  then ▷ not a difficult point
33:      output  $(y \leq -b)$  ▷ Boolean value ( $y \in I$ )
34:    else ▷ new special difficult point, red transition,
35:      ▷ possibly combined with a black transition below
36:       $k' \leftarrow k + 1$ ;
37:      Release  $[x_k, y_k]$ ; Check  $y_k < -c \leq x_k$ 
38:      output  $(x_k \leq -b)$  ▷ Boolean value
39:    end if
40:  end if
41:  if  $\neg q$  then ▷ not the event predicted by  $y_1$ 
42:    goto  $(k', 2)$ 
43:  else ▷ event predicted by  $y_1$ , black transition
44:    ▷ possibly combined with a blue transition below
45:    Check  $y_1 = 0$ ; Release  $[y_1]$ ;  $y_1 = -\infty$ 
46:    if  $k' = 1$  then
47:      goto  $(0, 1)$ 
48:    else
49:      Shift  $x_2, y_2, \dots, x_k, y_k, x_1, y_1$  to  $x_1, y_1, \dots, x_k, y_k$ 
50:    end if
51:    choose non-deterministically
52:      when True do goto  $(k' - 1, 1)$  ▷ not the event predicted by the new  $x_1$ 
53:      when  $(x_1 = 0)$  do Release  $[x_1]$ ;  $x_1 = -\infty$ ; goto  $(k' - 1, 2)$  ▷ blue transition
54:    end choose
55:  end if

```

References

- 1 S. Akshay, Paul Gastin, R. Govind, Aniruddha R. Joshi, and B. Srivathsan. A unified model for real-time systems: Symbolic techniques and implementation. In *CAV (1)*, volume 13964 of *Lecture Notes in Computer Science*, pages 266–288. Springer, 2023.
- 2 S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan. MITL model checking via generalized timed automata and a new liveness algorithm, 2024. [arXiv:2407.08452](https://arxiv.org/abs/2407.08452).
- 3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- 4 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
- 5 Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.
- 6 Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. In *LICS*, pages 390–401. IEEE Computer Society, 1990.
- 7 Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.
- 8 Laura Bozzelli, Angelo Montanari, and Adriano Peron. Complexity issues for timeline-based planning over dense time under future and minimal semantics. *Theor. Comput. Sci.*, 901:87–113, 2022.
- 9 Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata. In *FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2013.
- 10 Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata over infinite words. In *FORMATS*, volume 8711 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2014.
- 11 Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. MightyL: A compositional translation from MITL to timed automata. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 421–440. Springer, 2017.
- 12 Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model verifier. In *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 495–499. Springer, 1999.
- 13 David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.
- 14 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and ω -automata manipulation. In *ATVA*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129, 2016.
- 15 Thomas Ferrère, Oded Maler, Dejan Nickovic, and Amir Pnueli. From real-time logic to timed automata. *J. ACM*, 66(3):19:1–19:31, 2019.
- 16 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. In *CONCUR*, volume 118 of *LIPICs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 17 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2019.
- 18 Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.
- 19 Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. On regions and zones for event-clock automata. *Formal Methods Syst. Des.*, 45(3):330–380, 2014.

- 20 Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995.
- 21 F. Herbretreau and G. Point. TChecker. <https://github.com/fredher/tchecker>, v0.2 - April 2019.
- 22 Frédéric Herbretreau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. Why liveness for timed automata is hard, and what we can do about it. *ACM Trans. Comput. Log.*, 21(3):17:1–17:28, 2020. doi:10.1145/3372310.
- 23 Frédéric Herbretreau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. In *LICS*, pages 375–384. IEEE Computer Society, 2012.
- 24 Frédéric Herbretreau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. *Formal Methods Syst. Des.*, 40(2):122–146, 2012.
- 25 Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5):279–295, 1997.
- 26 Henrik Ejersbo Jensen, Kim G. Larsen, and Arne Skou. Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL. In *The Spin Verification System*, volume 32 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 33–49. DIMACS/AMS, 1996.
- 27 Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.
- 28 Guangyuan Li. Checking timed Büchi automata emptiness using LU-abstractions. In Joël Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009. doi:10.1007/978-3-642-04368-0_18.
- 29 Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2006.
- 30 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018.
- 31 Joël Ouaknine and James Worrell. On metric temporal logic and faulty turing machines. In *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.
- 32 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.
- 33 Amir Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 510–584. Springer, 1986.
- 34 Amir Pnueli and Eyal Harel. Applications of temporal logic to the specification of real-time systems. In *FTRTFT*, volume 331 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1988.
- 35 Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods Syst. Des.*, 26(3):267–292, 2005.
- 36 Moshe Y Vardi. An automata-theoretic approach to linear temporal logic. *Lecture Notes in Computer Science*, 1043:238–266, 1996.
- 37 Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *FTRTFT*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer, 1994.

Causally Deterministic Markov Decision Processes

S. Akshay ✉ 

Indian Institute of Technology Bombay, Mumbai, India

Tobias Meggendorfer ✉ 

Lancaster University Leipzig, Germany

P. S. Thiagarajan ✉

The University of North Carolina at Chapel Hill, NC, USA

Chennai Mathematical Institute, India

Abstract

Probabilistic systems are often modeled using *factored* versions of Markov decision processes (MDPs), where the states are composed out of the local states of components and each transition involves only a small subset of the components. Concurrency arises naturally in such systems. Our goal is to exploit concurrency when analyzing factored MDPs (FMDPs). To do so, we first formulate FMDPs in a way that aids this goal and port several notions from concurrency theory to the probabilistic setting of MDPs. In particular, we provide a concurrent semantics for FMDPs based on the classical notion of event structures, thereby cleanly separating causality, concurrency, and conflicts that arise from stochastic choices. We further identify the subclass of *causally deterministic* FMDPs (CMDPs), where non-determinism arises solely due to concurrency. Using our event structure semantics, we show that in CMDPs, *local reachability* properties can be computed using a “greedy” strategy. Finally, we implement our ideas in a prototype and apply it to four models, confirming the potential for substantial improvements over state-of-the-art methods.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases MDPs, distribution, causal determinism

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.6

Supplementary Material *Software (Artifact)*: <https://zenodo.org/records/12657579> [23]

Funding *S. Akshay*: Partially supported by Google India Research Award 2023 and SBI Foundation Hub for Data and Analytics.

1 Introduction

Factored versions of systems often constitute an important subclass. Two typical, well known examples – among very many – are Petri nets (and related models of concurrency) [26] and dynamic Bayesian networks [17]. A common key feature is that a state of the system is a *vector* of local component states. Further, a transition only involves a small subset of the components and hence can be specified succinctly; so much so, the size of the induced global system will often be exponential in the size of the factored presentation. This allows to model *large* systems without having to enumerate the set of global states and transitions explicitly.

Here, we explore this idea in the probabilistic setting of Markov decision processes (MDPs). Our starting point is a variant of factored MDPs (FMDPs). These are made up of several individual components, and a vector of local states constitutes the global state. Moreover, each action is associated with a fixed set of components named its *locations*. The availability of an action at a global state only depends on the local states of its locations and the stochastic changes that take place when an action occurs only involve the states of its locations. The resulting transition relation can be easily converted into the usual presentation of factored MDPs in the literature [4, 14]. Notably, our version of FMDPs includes models



© S. Akshay, Tobias Meggendorfer, and P. S. Thiagarajan;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 6; pp. 6:1–6:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

specified in the established PRISM language [18] and JANI [5]. When handling systems with a large number of components, a key challenge is to analyze the global behavior in terms of the factored presentation instead of first explicitly constructing the global behavior. In the case of factored MDPs, this is particularly difficult due to the complex interplay between non-deterministic, stochastic and concurrent features of the dynamics.

As a first step toward addressing this challenge, we focus on the analysis of a subclass of FMDPs, called *causally deterministic* FMDPs (CMDPs). The defining feature of CMDPs is that any two actions that are available at a global state will have disjoint sets of locations. As a result, the two actions will be *causally independent*: executing one of them will not affect the availability of the other or its outcomes. Consequently, CMDPs admit a powerful partial order based analysis technique for verifying certain “robust” probabilistic properties. In the current paper, we focus on local reachability properties.

As a key tool to analyzing CMDPs, we identify the notion of *complete* strategies, which can be explained as follows. In a CMDP, the role of a strategy is to resolve the non-determinism that arises in the dynamics due to causally independent actions. This means a strategy linearizes a partially ordered set of action occurrences. Hence, if an action a is enabled at a state s , and is not chosen along a finite sequence of moves leading to the state s' , then a will still be available at s' . Accordingly, a complete strategy is defined to be one in which the set of trajectories along which an available action is ignored forever has probability measure 0. Based on this notion, our main technical results for CMDPs are that (i) complete strategies suffice to obtain the optimal (maximal) probability of a local reachability property and (ii) *all* complete strategies will yield the same maximal probability value. Consequently we can choose a greedy complete strategy which avoids visiting many “useless” states. As the experimental results in Sec. 6 show, for CMDPs, our method vastly outperforms established, highly optimized tools such as Storm [8].

We establish these properties by exploiting fundamental objects drawn from concurrency theory, namely Mazurkiewicz traces [9] and prime event structures [24]. In particular, we develop an event structure semantics for *all* FMDPs. Since they arise in the context of FMDPs, the events in the event structure will have probability values assigned to them in a natural manner. We then use these probabilistic events to show that all complete strategies yield the same maximal probability values for local reachability properties. We view the present work as a first step towards developing partial-order reductions for FMDPs in general. Specifically, via the event structure semantics, based on Mazurkiewicz traces, a variety of techniques such as finite prefixes of event structures [11], and partial order reduction notions such as ample sets [13] and stubborn sets [15] can be brought to bear when analyzing FMDPs.

To summarize, our contributions are:

1. A novel class of factored MDPs, called CMDPs, in which the non-determinism between actions arises solely due to their causal independence.
2. An event structure semantics for FMDPs that cleanly separates causality, concurrency, and (stochastic) conflicts arising in the global behavior of an FMDP.
3. The identification of complete strategies for CMDPs which have the crucial properties; (i) they suffice to attain the optimal probability values for local reachability properties and (ii) all of them yield the same optimal value.
4. A prototypical implementation of (i) a syntactical over-approximation for checking that the input MDP is a CMDP and (ii) a greedy complete strategy accompanied by an experimental evaluation on four models. Comparison with existing state-of-the-art tools, e.g., Storm [8], shows a vast performance improvement for the evaluated models, highlighting the potential benefits of our approach.

Structure. In the rest of this section we review related work. We then present basic material concerning Markov chains and Markov decision processes. In Sec. 3, we introduce our class of FMDPs and the subclass of causally deterministic FMDPs (CMDPs). In the subsequent section we construct the event structure representation of our FMDPs which then leads to the main results developed in Sec. 5. The greedy strategy, its implementation, and the experimental results are presented in Sec. 6. The paper concludes with Sec. 7.

Related Work. Factored MDPs have been long studied in the literature [4, 14], where the transition relation is usually presented using a two layer dynamic Bayesian network. With an eye toward learning applications, a reward function is also included. Our formulation of FMDPs is geared towards capturing the distributed dynamics of FMDPs and hence is based on the notion of locations. Further, reward functions play no role in the present setting.

In the verification setting, several works have considered compositional methods to reason about large MDPs that are “factorized” via compositional operations. While some approaches use bisimulation based equivalences [12], others use abstractions [16], and yet others use a category-theoretical view of MDPs [31]. In a sense, these represent an approach which is orthogonal to ours, which is grounded in FMDPs and focused on solving quantitative behavioral properties. There have also been works adapting partial-order reduction techniques to the probabilistic setting using ample sets [13] and stubborn sets [15]. Variants of these approaches are incorporated in state-of-the-art tools such as Storm [8] and PRISM [18]. However, these works deal with MDPs viewed as monolithic objects presented in terms of global states and transitions. Thus, it will be difficult – if not impossible – to deal with the large MDPs that are presented succinctly as FMDPs. Furthermore, the focus in these works is on model checking linear time and branching time (probabilistic) properties using a semantically defined notion of commutability of actions along an execution sequence. These techniques do not enable one to compute optimal values of local reachability properties that we achieve using the event structure semantics. It will however be interesting to explore these methods in the context of CMDPs and, more generally, FMDPs.

Similarly, [7] exploit a model consisting of purely probabilistic components, however they use these components only to obtain a compact symbolic representation of the global MDP; in the end, they still work with the entire global MDP. In contrast, our analysis method directly works with the factored representation of the global MDP.

Several studies start with event structures, adjoin probabilities to events and study the resulting objects from a theoretical standpoint [1, 30]. However, in these studies probabilities are introduced in an ad-hoc manner and no attempt is made to establish a verification framework for an associated system model. In sharp contrast, the probabilities attached to the events in our event structures arise naturally from the associated MDPs. Furthermore, our use of event structures is firmly grounded in a verification framework for CMDPs.

Generalized stochastic Petri nets (GSPN) [20], despite being based on Petri nets, do not exploit concurrency and instead focus on their interleaved global behaviors in terms of (continuous time) Markov chains. A variant called Markov decision Petri nets is proposed in [3] as a high level modeling formalism. Their global behaviors are captured by MDPs and analyzed using symbolic representations. Here again concurrency essentially plays no role.

Finally, distributed Markov chains (DMCs) studied in [28, 29] have a similar flavour to CMDPs. DMCs consist of a network of probabilistic transition systems that synchronize on common actions with a sufficiently strong *syntactic* restriction ensuring that if two actions are enabled at a global state then they must involve disjoint sets of components. In addition,

they focus on statistical model checking of properties specified in a variant of bounded linear temporal logic. In contrast, CMDPs are a natural *behavioral* subclass of MDPs and our focus is determining the *exact* maximal probability of (unbounded) local reachability properties.

2 Preliminaries

A *Markov chain (MC)* (e.g., [2]), is a tuple $M = (S, \hat{s}, P)$, where S is a (countable) set of states, $\hat{s} \in S$ is the *initial state*, and $P : S \rightarrow \mathcal{D}(S)$ is a *transition function* that for each state s yields a probability distribution over successor states, where $\mathcal{D}(S)$ is the set of distributions over S . A *Markov decision process (MDP)* (e.g., [25]) is a tuple $\mathcal{M} = (S, \hat{s}, Act, P)$, where S is a (finite) set of states, $\hat{s} \in S$ is the initial state, Act a finite set of actions, overloaded as $Act(s) \subseteq Act$ specifying available actions at a state s , and $P : S \times Act \rightarrow \mathcal{D}(S)$ yielding a distribution over successors for each $s \in S$ and $a \in Act(s)$. For simplicity, we write $P(s, s')$ instead of $P(s)(s')$ for a MC and $P(s, a, s')$ instead of $P(s, a)(s')$ for an MDP.

Paths. An infinite path in an MC M is an infinite sequence $\rho = s_1 s_2 \dots$ where $s_1 = \hat{s}$ and $P(s_i, s_{i+1}) > 0$ for all i . A finite path ρ is a finite prefix of an infinite path. A Markov chain $M = (S, \hat{s}, P)$ naturally induces a unique probability measure \Pr_M over the σ -algebra generated by the cylinder sets induced by the finite paths [2, Sec. 10.1]. Similarly, for an MDP \mathcal{M} , an infinite path is a sequence $\rho = s_1 a_1 s_2 a_2 \dots$ such that $s_1 = \hat{s}$ and for all i we have $a_i \in Act(s_i)$ and $P(s_i, a_i, s_{i+1}) > 0$. A finite path is a finite prefix of an infinite path *ending in a state*. We write $FPaths_{\mathcal{M}}$ to denote the set of finite paths in \mathcal{M} . Moreover, $|\rho| = k$ denotes the length of a path (setting it to ∞ for infinite paths) and we define it to be the number of actions (transitions) that appear in the path. For $i \leq |\rho|$ we write ρ_i to refer to the i -th state in a path. Finally, $last(\rho) = \rho_{|\rho|}$ denotes the last state in a finite path.

Strategies. Intuitively, in every state s , an action a from $Act(s)$ is chosen and the system advances to a successor state s' according to the probability distribution given by $P(s, a)$. Starting from the initial state \hat{s} and repeating this process indefinitely yields an infinite path. The way actions are chosen along an infinite path is captured by *strategies*. Specifically, a strategy is function mapping each finite path to one of the actions, say a , available in the last state, say s , of the path. This leads to new states chosen according to the distribution $P(s, a)$. We let Π refer to the set of all strategies. To support our technical constructions arising later, our strategies are thus deterministic but not necessarily memoryless. A strategy is *memoryless* (or *positional*) if it only depends on the current state, i.e. $\pi(\rho) = \pi(\rho')$ whenever $last(\rho) = last(\rho')$. As usual, a strategy π induces the Markov chain $\mathcal{M}^\pi = (FPaths_{\mathcal{M}}, \hat{s}, P^\pi)$, where for $\rho \in FPaths_{\mathcal{M}}$ with $s = last(\rho)$ and $a = \pi(\rho) \in Act(s)$ we set $P^\pi(\rho, \rho a s') = P(s, a, s')$. We write $\Pr_{\mathcal{M}, \hat{s}}^\pi = \Pr_{\mathcal{M}^\pi, \hat{s}}$ for the induced probability measure.

Reachability. Fix an MDP $\mathcal{M} = (S, \hat{s}, Act, P)$. Then, (*unbounded*) *reachability* for a set of target states $T \subseteq S$ is the set of all (infinite) paths which eventually visit one of the target states, i.e. $\diamond T = \{\rho \mid \exists i. \rho_i \in T\}$, which is measurable [2, Sec. 10.1.1]. For a strategy π , the probability of reaching T according to π is the probability assigned to this set of infinite paths $\diamond T$ in \mathcal{M}^π , i.e. $\Pr_{\mathcal{M}}^\pi[\diamond T]$. However, different strategies will in general yield different probabilities and one is often interested in the *maximum* of these probabilities. In other words, the goal is to determine $\sup_{\pi \in \Pi} \Pr_{\mathcal{M}}^\pi[\diamond T]$ (also called the *value*). For MDPs, a well-known result states that it suffices to consider memoryless deterministic strategies for this maximization (see, e.g., [2, Lem. 10.102]).

3 Factored MDPs and Causal Determinacy

Often, an MDP comprises interacting components (or agents, processes). In particular, many modelling formalisms used in practice, e.g. the PRISM language [18] or JANI [5], define MDPs in this manner. Consequently, a state of the MDP will consist of a tuple of local states of the component processes. Further, an action will often involve only a fixed subset of the components leading to a stochastic transformation of the states of these components while the states of the other components are left untouched. We propose to use *factored* MDPs to study such systems.

Accordingly, let $Proc$ denote a finite set of components. Each component $p \in Proc$ has a set of *local states* denoted as S_p . This gives rise to the set of *global states* $\mathbf{S} = \prod_{p \in Proc} S_p$. To capture the idea that an action involves only a fixed subset of components, we use the *location map* $loc : Act \rightarrow 2^{Proc} \setminus \emptyset$ to specify for each action a the set of components that participate in a . For convenience, we also identify a global state \mathbf{s} with the map $\mathbf{s} : Proc \rightarrow \bigcup_{p \in Proc} S_p$ such that $\mathbf{s}(p) \in S_p$ for $p \in Proc$. Then, for a set of components $P \subseteq Proc$, we let $\mathbf{s}[P]$ denote the map \mathbf{s} restricted to P . In other words, $\mathbf{s}[P] \in \prod_{p \in P} S_p$ and $\mathbf{s}[P](p) = \mathbf{s}(p)$ for $p \in P$. For $a \in Act$, we often write $\mathbf{s}[a]$ instead of $\mathbf{s}[loc(a)]$ and call it the a -state induced by \mathbf{s} . This leads to $\mathbf{S}[a] = \{\mathbf{s}[a] \mid \mathbf{s} \in \mathbf{S}\}$, the set of a -states. With these notations at hand, we introduce factored MDPs (FMDPs).

► **Definition 1.** A factored MDP \mathcal{M} is a tuple $(\{S_p\}_{p \in Proc}, \{\hat{s}_p\}_{p \in Proc}, Act, loc, \{P_a\}_{a \in Act})$ where (i) $Proc$ is a finite, non-empty set of components, (ii) S_p is a finite, non-empty set of states for each $p \in Proc$, (iii) $\hat{s}_p \in S_p$ is the initial state of component p , inducing the global initial state $\hat{\mathbf{s}}$ with $\hat{\mathbf{s}}(p) = \hat{s}_p$ for each $p \in Proc$, (iv) Act is a finite, non-empty set of actions, (v) $loc : Act \rightarrow 2^{Proc} \setminus \{\emptyset\}$ is the locations map, and (vi) for each $a \in Act$, $P_a : \mathbf{S}[a] \rightarrow \mathcal{D}(\mathbf{S}[a])$ is a (partial) transition function.

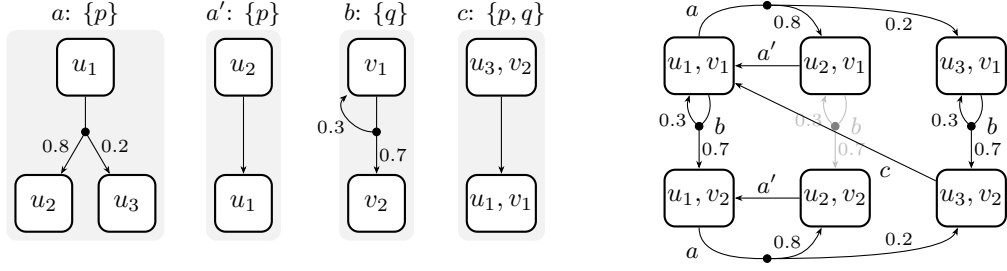
Similar to MDPs, we write $P_a(\mathbf{u}, \mathbf{v})$ instead of $P_a(\mathbf{u})(\mathbf{v})$. The FMDP \mathcal{M} induces an MDP called its global MDP defined as follows.

► **Definition 2.** Let \mathcal{M} be an FMDP as above. Then its global MDP is given by $\widehat{\mathcal{M}} = (\mathbf{S}, \hat{\mathbf{s}}, Act, P)$ where (i) $a \in Act(\mathbf{s})$ iff $P_a(\mathbf{s}[a])$ is defined, and (ii) for every $\mathbf{s}' \in \mathbf{S}$ and $a \in Act(\mathbf{s})$, we have $P(\mathbf{s}, a, \mathbf{s}') = v > 0$ iff $P_a(\mathbf{s}[a], \mathbf{s}'[a]) = v$ and $\mathbf{s}[Proc \setminus loc(a)] = \mathbf{s}'[Proc \setminus loc(a)]$.

We can immediately verify that $\widehat{\mathcal{M}}$ is indeed an MDP. Moreover, $\widehat{\mathcal{M}}$ has two important properties, namely: (F1) The availability of an action a at a state \mathbf{s} depends only on $\mathbf{s}[a]$. Further, when an action a occurs at a state \mathbf{s} , the changes it produces involve only the components in $loc(a)$; the local states of components in $Proc \setminus loc(a)$ remain unchanged. (F2) When action a occurs at a global state \mathbf{s} , the changes it produces (to the states of participating components) depends only on the a -state $\mathbf{s}[a]$. In particular, suppose \mathbf{s}_1 and \mathbf{s}_2 are global states and $a \in Act$ is an action where $\mathbf{s}_1[a] = \mathbf{s}_2[a]$. Then, if $P(\mathbf{s}_1, a, \mathbf{s}'_1) = v > 0$ there exists a unique global state \mathbf{s}'_2 such that $P(\mathbf{s}_2, a, \mathbf{s}'_2) = v$ and $\mathbf{s}'_1[a] = \mathbf{s}'_2[a]$.

Before presenting an illustrative example, we briefly remark on this defining way of defining an FMDP and how it relates to established notions.

► **Remark 3.** Traditionally, FMDPs are defined using a transition relation represented by a two-layer dynamic Bayesian network [4, 14]. We have chosen to use a slightly different definition, aligned with concurrency theory, so that the distributed nature of the dynamics can be clearly brought out, as we shall see below. However, our theory is neutral to *how* the dynamics of the individual components are represented as long as the global transitions



■ **Figure 1** This figure illustrates a two-component FMDP where $Proc = \{p, q\}$, $S_p = \{u_1, u_2, u_3\}$, $S_q = \{v_1, v_2\}$, and $Act = \{a, a', b, c\}$. On the left, for each action a both $loc(a)$ and P_a are depicted. On the right, the induced global MDP is shown. The middle b action is greyed out solely for readability, it is not special in any way. We omit the probability label if it is 1.

are factored in terms of the components participating in the actions. In particular, once the properties (F1) and (F2) stated above are satisfied by the resulting global MDP, our theory is applicable to any model which exhibits such behaviour, e.g. the DBN-based definitions.

► **Example 4.** In Fig. 1, an example of an FMDP (on the left) and its induced global MDP (on the right) is shown. To explain the relation between FMDP and global MDP, we write $\langle u_1, u_2 \rangle$ and similar to denote global states and c -states as tuples of local states, as the correspondence with the local states of the components is clear. The a -transition from $\langle u_1 \rangle$ to $\langle u_2 \rangle$ in the FMDP implies a is available at the global state $\langle u_1, v_1 \rangle$ since $\langle u_1, v_1 \rangle[a] = \langle u_1 \rangle$ and $P_a(\langle u_1 \rangle)$ is defined in the FMDP. Further, $P(\langle u_1, v_1 \rangle, a, \langle u_2, v_1 \rangle) = 0.8$ as $P_a(\langle u_1 \rangle, \langle u_2 \rangle) = 0.8$. In particular, this transition does not modify the state of the q -component since $loc(a) = \{p\}$. The other transitions shown in the global MDP can be inferred using similar reasoning.

By slight abuse of notation, in the following we write \mathbf{S} to denote the set of *reachable* states, defined in the obvious way. We also identify the FMDP with its induced global MDP and freely go back and forth between the two notions and the associated notations. Finally, for simplicity we assume that the FMDPs we deal with are free of *deadlocks*, i.e. if $\mathbf{s} \in \mathbf{S}$ then $Act(\mathbf{s}) \neq \emptyset$. (Since our focus is on reachability, this can be ensured by adding a new component d with a single state s_d , a new action a_d with $loc(a_d) = \{d\}$, and $P_{a_d}(\langle s_d \rangle, a_d, \langle s_d \rangle) = 1$.)

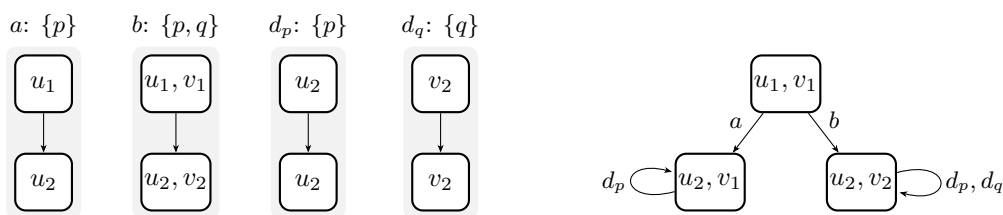
3.1 Local Reachability

Let \mathcal{M} be an FMDP. Then, a *local* reachability problem is specified by $T \subseteq S_p$ for some component p . Let $\mathbf{T} = \{\mathbf{s} \mid \mathbf{s}(p) \in T\}$ the corresponding global reachability set. The goal is to determine the probability $\sup_{\pi \in \Pi} \Pr_{\mathcal{M}}^{\pi}[\diamond \mathbf{T}]$.

Local reachability for an FMDP can be solved by ignoring its factored nature and instead treat it as a “global” reachability problem on the induced global MDP. In this case, classical approaches as employed by PRISM [18] and Storm [8] can be used. This problem is in PTIME [2, Cor. 10.107], but *in the size of the global MDP*, which can easily be exponential in the size of the FMDP. Our goal is to mitigate this state-space explosion by exploiting the partially ordered nature of the dynamics of the model.

3.2 Causal Determinacy and Complete Strategies

As a first step, we shall tackle the state explosion problem by considering the subclass of FMDPs in which the sole source of non-determinism is from the *causal independence* of actions. This idea can be captured through a natural restriction.



■ **Figure 2** This figure illustrates an FMDP which is not CD. We have $Proc = \{p, q\}$, $S_p = \{u_1, u_2\}$, $S_q = \{v_1, v_2\}$, and $Act = \{a, b, d_p, d_q\}$. On the left, we depict the transition function P_a for all $a \in Act$ and to the right the induced global MDP.

► **Definition 5.** An FMDP \mathcal{M} is causally deterministic (CD) if for every (reachable) state \mathbf{s} and $a, b \in Act(\mathbf{s})$ with $a \neq b$ we have $loc(a) \cap loc(b) = \emptyset$. We call such an FMDP a CMDP.

► **Example 6.** The FMDP shown in Fig. 1 is causally deterministic: In any global state, the available set of actions is $\{a, b\}$, $\{a', b\}$ or $\{c\}$. In contrast, the FMDP of Fig. 2 is not CD. In $\langle u_1, v_1 \rangle$ both a and b are available, but $loc(a) \cap loc(b) = \{p\} \neq \emptyset$. And indeed, it is relevant whether we choose a or b . For example, a leads to a state in which b is not enabled anymore, and, in particular, v_2 is not reachable, while b reaches v_2 with probability 1.

► **Remark 7.** Causal determinacy is intrinsically a concurrency based notion. If $a, b \in Act(\mathbf{s})$ with $a \neq b$ then a and b can occur independent of each other at \mathbf{s} . In fact, suppose $\mathbf{s}_0 a_1 \mathbf{s}_1 \cdots \mathbf{s}_{n-1} a_n \mathbf{s}_n$ is a finite path, $b \in Act(\mathbf{s}_0)$ and $a_i \neq b$ for $1 \leq i \leq n$. Then $loc(a_i) \cap loc(b) = \emptyset$ for all $1 \leq i \leq n$ and $b \in Act(\mathbf{s}_n)$. This follows from the fact that a CMDP is an FMDP and hence $\mathbf{s}_0[b] = \mathbf{s}_n[b]$. This basic feature of a CMDP leads to a partial-order based technique using which one can often efficiently verify many behavioral properties that are “robust” with respect to interleavings of partially ordered behaviors, such as local reachability. Due to space considerations, we will not pause to formalize the notion of robust properties since it is not needed to establish our results.

► **Remark 8.** Deciding whether the global MDP induced by an FMDP (encoded in a standard manner) is CD is in PSPACE. The idea is to convert the probabilistic transitions to non-deterministic ones, and reduce the CD property to a reachability property of the resulting 1-safe Petri net, known to be in PSPACE [10]. However, given our main goals, establishing this result in detail would be a digression and hence we do not do so. That said, for practical purposes, we later discuss a simple, sufficient syntactic condition allowing us to over-approximate CD in our case studies.

As a central tool to exploit causal determinacy, we introduce *complete* strategies.

► **Definition 9.** Let ρ be an infinite path in a Markov chain \mathcal{M}^π induced by a strategy π on a CMDP \mathcal{M} . Then, ρ is a complete path iff for every $i \geq 0$, if $a \in Act(\rho_i)$ then there exists $j \geq i$ such that $\pi(\rho_i \rho_{i+1} \dots \rho_j) = a$. In other words, if a is available at ρ_i then it is eventually chosen by the strategy along the path (where it will remain available due to CD).

Let Υ be the set of complete paths in \mathcal{M}^π . A strategy π is complete iff $\Pr_{\mathcal{M}}^\pi[\Upsilon] = 1$.

Thus, incomplete paths may be present in \mathcal{M}^π , but the collection of such paths has measure 0 and does not contribute to the reachability probabilities of interest. Note that Υ is measurable as it can be written as countable intersections and unions of cylinder sets.

First, we show here that it suffices to consider only complete strategies for local reachability. Later we will show that *all* complete strategies will yield the same, maximal probability value. Consequently, we can freely choose a “greedy” strategy with which the maximal probabilities can be computed in an efficient manner.

► **Lemma 10.** *There exists a deterministic, complete strategy $\pi \in \Pi$ which achieves the optimal value, i.e. $\Pr_{\mathcal{M}}^{\pi}[\diamond \mathbf{T}] = \sup_{\pi' \in \Pi} \Pr_{\mathcal{M}}^{\pi'}[\diamond \mathbf{T}]$.*

Proof Sketch. We delegate the (rather routine) proof to App. B. For a sketch, we show that any (optimal, memoryless) strategy can be extended to a complete strategy without reducing the reachability probability it achieves. Intuitively, the modified strategy waits until the original strategy has visited all the states it will ever visit (thus any goal it might reach is already reached), which happens with probability 1, and then switches to a “complete” mode in which it plays all the actions that have not been played since they became available. ◀

In the next section, we develop the event structure semantics for FMDPs. Using this, we show in Sec. 5 that any two complete strategies achieve the *same* value.

4 An Event Structure Semantics for FMDPs

4.1 Mazurkiewicz Trace Languages

We first associate a Mazurkiewicz trace language with an FMDP. Then, using a standard construction, we obtain the event structure representation. We recall from [21] a Mazurkiewicz trace alphabet is a pair (Σ, I) where Σ is a finite non-empty alphabet and $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric relation called the *independence* relation. When describing the executions of a distributed system, Σ is the set of actions and $a I b$ asserts that the actions a and b are “causally” independent. In other words, they can be executed in any order when they are both enabled. We define $D = (\Sigma \times \Sigma) \setminus I$ to be the *dependency* relation. The relation I induces in a natural way the equivalence relation $\approx_I \Sigma^* \times \Sigma^*$. It is the least equivalence satisfying $\sigma a b \sigma' \approx_I \sigma b a \sigma'$ for $a I b$. For $\sigma \in \Sigma^*$, $[\sigma]$ denotes the \approx_I -equivalence class containing σ , often called a *Mazurkiewicz trace*. It corresponds to the set of all possible interleavings of a unique partially ordered set of actions. A Mazurkiewicz trace language is a subset of $\{[\sigma] \mid \sigma \in \Sigma^*\}$, i.e. a set of Mazurkiewicz traces. For convenience, we abbreviate Mazurkiewicz traces (Mazurkiewicz trace languages) as traces (trace languages).

4.2 The Mazurkiewicz Trace Language of an FMDP

To define the trace language of an FMDP we start with \mathcal{M} -events.

► **Definition 11.** *Let $\mathcal{M} = (\{S_p\}_{p \in Proc}, \{\hat{s}_p\}_{p \in Proc}, Act, loc, \{P_a\}_{a \in Act})$ be an FMDP. Then $\alpha = (\mathbf{u}, a, \mathbf{v})$ is an \mathcal{M} -event if $P_a(\mathbf{u}, \mathbf{v}) > 0$. We define the probability of α as $Pr(\alpha) = P_a(\mathbf{u}, a, \mathbf{v})$. Furthermore, we set $act(\alpha) = a$ and $loc(\alpha) = loc(a)$.*

The \mathcal{M} -event $\alpha = (\mathbf{u}, a, \mathbf{v})$ comprises the a -state that must hold at a state \mathbf{s} for it to occur (i.e. $\mathbf{s}[a] = \mathbf{u}$). It also reports the a -state that is chosen with probability $Pr(\alpha)$ resulting in the global state \mathbf{s}' (i.e. $\mathbf{s}'[a] = \mathbf{v}$ and $\mathbf{s}[Proc \setminus loc(a)] = \mathbf{s}'[Proc \setminus loc(a)]$). For instance, $\alpha = (\langle u_1 \rangle, a, \langle u_2 \rangle)$ is an \mathcal{M} -event in the FMDP shown in Fig. 1 with $Pr(\alpha) = 0.8$.

\mathcal{M} -events naturally give rise to the transition relation $\longrightarrow_{\mathcal{M}}$ over \mathbf{S} , defined as follows. Suppose $\alpha = (\mathbf{u}, a, \mathbf{v})$, and $\mathbf{s}, \mathbf{s}' \in \mathbf{S}$. Then $\mathbf{s} \xrightarrow{\alpha}_{\mathcal{M}} \mathbf{s}'$ if $\mathbf{s}[a] = \mathbf{u}$, $\mathbf{s}'[a] = \mathbf{v}$, and $\mathbf{s}[Proc \setminus loc(a)] = \mathbf{s}'[Proc \setminus loc(a)]$. As usual, we write \longrightarrow instead of $\longrightarrow_{\mathcal{M}}$. We now define an \mathcal{M} -path to be a sequence $\mathbf{s}_0 \alpha_1 \mathbf{s}_1 \alpha_2 \cdots \mathbf{s}_{n-1} \alpha_n \mathbf{s}_n$ such that (i) $\mathbf{s}_0 = \hat{\mathbf{s}}$ and (ii) $\mathbf{s}_{i-1} \xrightarrow{\alpha_i} \mathbf{s}_i$ for $1 \leq i \leq n$. In essence, \mathcal{M} -paths correspond to finite paths in the global MDP. Since we only deal with \mathcal{M} -paths from now on, we say “path” instead of \mathcal{M} -path henceforth.

Let $\Sigma_{\mathcal{M}}$ denote the set of \mathcal{M} -events. In the following, we instead write Σ , as \mathcal{M} will be clear from the context. Moreover, we set $\Sigma_p = \{\alpha \mid \alpha \in \Sigma, p \in \text{loc}(\alpha)\}$ for each component p . We define the independence relation $I \subseteq \Sigma \times \Sigma$ as $I = \{(\alpha, \beta) \mid \text{loc}(\alpha) \cap \text{loc}(\beta) = \emptyset\}$. Clearly, I is irreflexive and symmetric, and hence (Σ, I) is a trace alphabet. Next, for $\Sigma' \subseteq \Sigma$ let $\text{prj}_{\Sigma'} : \Sigma^* \rightarrow \Sigma'^*$ be the projection which from sequences in Σ^* erases all appearances of letters that are not in Σ' . We abbreviate prj_{Σ_p} as prj_p . This leads to the equivalence relation \approx_I over Σ^* given by $\sigma \approx_I \sigma'$ iff for every $p \in \text{Proc}$ we have $\text{prj}_p(\sigma) = \text{prj}_p(\sigma')$. Effectively, two traces are equivalent if no single component can differentiate between them. We define \approx_I in this way instead of using the usual partial commutative relation, as it extends smoothly to infinite \mathcal{M} -event sequences. For convenience, we write from now on \approx instead of \approx_I .

Let $\sigma = \alpha_1 \alpha_2 \cdots \alpha_n \in \Sigma^*$. Then σ is an \mathcal{M} -event sequence of \mathcal{M} if there exists states s_0, s_1, \dots, s_n such that $s_0 \alpha_1 s_1 \cdots s_{n-1} \alpha_n s_n$ is an \mathcal{M} -path. We let $L_{\mathcal{M}}^{\text{seq}}$ denote the set of \mathcal{M} -event sequences of \mathcal{M} . This leads to the trace language of \mathcal{M} given by $L_{\mathcal{M}} = \{[\sigma] \mid \sigma \in L_{\mathcal{M}}^{\text{seq}}\}$.

We next introduce some terminology to aid in the construction of the event structure representation of \mathcal{M} . These notions are generic to the theory of Mazurkiewicz trace languages. However, for convenience, we introduce them in the context of $L_{\mathcal{M}}$. First, $\sqsubseteq \subseteq L_{\mathcal{M}} \times L_{\mathcal{M}}$ is given by $[\sigma] \sqsubseteq [\sigma']$ iff $\text{prj}_p(\sigma)$ is a prefix of $\text{prj}_p(\sigma')$ for every $p \in \text{Proc}$. Clearly, \sqsubseteq is a well-defined partial ordering relation. Next, suppose $[\sigma], [\sigma'] \in L_{\mathcal{M}}$. Then $[\sigma] \uparrow [\sigma']$ iff there exists $[\sigma''] \in L_{\mathcal{M}}$ such that $[\sigma] \sqsubseteq [\sigma'']$ and $[\sigma'] \sqsubseteq [\sigma'']$. Finally, $[\sigma] \in L_{\mathcal{M}}$ is a *prime trace* iff there exists an \mathcal{M} -event α such that $\text{last}(\sigma') = \alpha$ for every $\sigma' \in [\sigma]$ where $\text{last}(\tau)$ is the last letter of the non-null sequence τ .

There is a rich theory of Mazurkiewicz trace languages available, see e.g. [9]. Here we only use basic facts of the theory which we state below. The proofs are standard and can be assembled from [9,27] and hence we omit them.

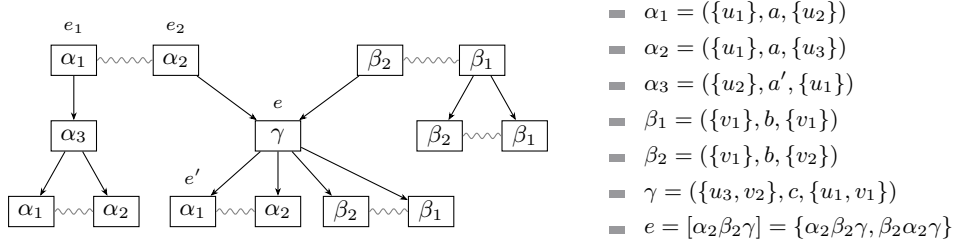
► **Proposition 12.** *It holds that (i) if $\sigma \approx \sigma'$ then $|\sigma| = |\sigma'|$, and (ii) $[\sigma] \uparrow [\sigma']$ iff there exist sequences σ'', σ_1 and σ'_1 such that (a) $\sigma \approx \sigma'' \sigma_1$ and $\sigma' \approx \sigma'' \sigma'_1$ and (b) $a I b$ for every letter a that appears in σ_1 and every letter b that appears in σ'_1 .*

4.3 The Event Structure Representation of FMDPs

We begin by recalling from [24] that a *prime event structure* is a tuple $ES = (E, \leq, \#)$ where (i) E is a countable set of events, (ii) $\leq \subseteq E \times E$ is a partial ordering relation called the *causality relation*, and (iii) $\# \subseteq E \times E$ is an irreflexive and symmetric relation called the *conflict relation*. It is required that if $e \# e'$ and $e' \leq e''$ then $e \# e''$. Usually, a prime event structure is accompanied by a labelling function that relates a system to its event structure representation. In our case, there will be two such functions.

► **Definition 13.** *Let $\mathcal{M} = (\{S_p\}_{p \in \text{Proc}}, \{\hat{s}_p\}_{p \in \text{Proc}}, \text{Act}, \text{loc}, \{P_a\}_{a \in \text{Act}})$ be an FMDP. Its event structure is a tuple $ES_{\mathcal{M}} = (E, \leq, \#, \lambda, \mu)$ where $(E, \leq, \#)$ is a prime event structure where (i) $E = \{[\sigma] \in L_{\mathcal{M}} \mid [\sigma] \text{ is a prime trace}\}$, (ii) \leq is \sqsubseteq restricted to $E \times E$, (iii) $[\sigma] \# [\sigma']$ iff it is not the case that $[\sigma] \uparrow [\sigma']$, (iv) $\lambda : E \rightarrow \Sigma$ is the labelling function satisfying $\lambda([\sigma]) = \text{last}(\sigma)$, and (v) $\mu : E \rightarrow [0, 1]$ assigns to $e = [\alpha_1 \alpha_2 \cdots \alpha_n] \in E$ the probability $\mu(e) = \prod_{1 \leq j \leq n} \text{Pr}(\alpha_j)$ (i.e. the probability of a prime trace is the product of the probabilities of the \mathcal{M} -events encountered along a sequence in the prime trace).*

In what follows we often write \leq instead of \sqsubseteq when viewing events as elements of E and not as traces. The “states” of an event structure are called *configurations* and the dynamics of $ES_{\mathcal{M}}$ is captured via a transition relation over its configurations.



■ **Figure 3** This figure illustrates the initial fragment of the event structure representation for the FMDP depicted in Fig. 1.

► **Definition 14.** For $c \subseteq E$, define $\downarrow c = \{y \mid \exists x \in c \text{ s.t. } y \leq x\}$. Then $c \subseteq E$ is a configuration iff $c = \downarrow c$ and $(c \times c) \cap \# = \emptyset$.

We define $C_{\mathcal{M}}$ to be the set of *finite* configurations of $ES_{\mathcal{M}}$ and note that \emptyset is a configuration. Let $c, c' \in C_{\mathcal{M}}$ and $\alpha \in \Sigma$. Then $c \xrightarrow{\alpha}_{ES} c'$ iff there exists $e \in E \setminus c$ such that $c \cup \{e\} = c'$ and $\lambda(e) = \alpha$. This basically says that an event e which is not in the configuration c can be added to it to obtain a larger configuration provided the past of e (under $<$) is contained in c . For simplicity, we write $\downarrow e$ instead of $\downarrow \{e\}$ for $e \in E$. Clearly, $\downarrow e$ is a configuration for every e in E .

In Fig. 3 we show the initial fragment of the event structure representation of the FMDP in Fig. 1. In order to minimize clutter, we have named the \mathcal{M} -events as α_1, α_2 , etc. We note that $Pr(\alpha_1) = 0.8$, $Pr(\alpha_2) = 0.2$, $Pr(\beta_1) = 0.3$, and $Pr(\beta_2) = 0.7$. Further, $Pr(\alpha_3) = 1 = Pr(\gamma)$. In the diagram, the directed arrows represent the *immediate* causality relation $<$ where $e < e'$ iff $e < e'$ and for every e'' , $e \leq e'' \leq e'$ implies $e = e''$ or $e'' = e'$. The remaining members of the causality relation are obtained by taking the reflexive transitive closure of this relation. Similarly, the squiggly lines represent the *minimal* conflict relation $\#$ defined as $e \# e'$ iff $e \# e'$ and $(\downarrow e \times \downarrow e') \cap \# = \{(e, e'), (e', e)\}$. Using the conflict inheritance requirement of an event structure, we can deduce all other members of the conflict relation. For example, in the event structure shown in Fig. 3, $e_3 \# e_4$ since $e_1 \# e_2 \leq e_4$ implies $e_1 \# e_4$ and since $e_1 \leq e_3$ and $\#$ is symmetric, we get $e_3 \# e_4$. In addition, we have listed the members of just one of the prime traces named e whose label is α_2 . For the remaining events, we have just indicated their labels.

The behavior of \mathcal{M} can be related to the behavior of $ES_{\mathcal{M}}$ as follows.

► **Proposition 15.** Let \mathcal{M} and $ES_{\mathcal{M}}$ be defined as above. Then the following statements hold.

1. Let $c = \{e_1, e_2, \dots, e_n\}$ be a configuration such that $e_1 e_2 \dots e_n$ is a linearization of the partial order (c, \leq) where, by abuse of notation, \leq also denotes the restriction of \leq to $c \times c$. Then there exists $\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_n \in \mathbf{S}$ such that $\mathbf{s}_0 = \hat{\mathbf{s}}$ and $\mathbf{s}_0 \lambda(e_1) \mathbf{s}_1 \lambda(e_2) \mathbf{s}_2 \dots \mathbf{s}_{n-1} \lambda(e_n) \mathbf{s}_n$ is a finite path in \mathcal{M} , which we shall call a c -path (in \mathcal{M}).
2. Let the function $state : C \rightarrow \mathbf{S}$ be given by (i) $state(\emptyset) = \hat{\mathbf{s}}$ and (ii) for a non-empty configuration c and c -path $\rho = \mathbf{s}_0 \alpha_1 \mathbf{s}_1 \dots \mathbf{s}_n$ in \mathcal{M} , we define $state(c) = \mathbf{s}_n$. Then, $state$ is a well-defined map from C onto the set of reachable states of \mathcal{M} .
3. Let $c, c' \in C$ and $\alpha = (\mathbf{u}, a, \mathbf{v})$. Then $c \xrightarrow{\alpha}_{ES} c'$ iff $P(state(c), \alpha, state(c')) = Pr(\alpha) > 0$.
4. Let $tr : C \rightarrow L_{\mathcal{M}}$ be the map given by (i) $tr(\emptyset) = \{\varepsilon\}$ and (ii) for a non-empty configuration c and c -path $\mathbf{s}_0 \alpha_1 \mathbf{s}_1 \dots \mathbf{s}_n$ in \mathcal{M} it is the case that $tr(c) = [\alpha_1 \alpha_2 \dots \alpha_n]$. Then, tr is well defined and a bijection.

Most of these observations are standard [27] and directly carry over to our setting. The third part is specific to FMDPs but follows directly from the definition of an \mathcal{M} -event.

We close out this section with a useful result which will be needed in the next section. Let $\sigma = \alpha_1\alpha_2 \cdots \alpha_n$ be an \mathcal{M} -event sequence in \mathcal{M} . Then $dg(\sigma)$, the subsequence of σ , is defined inductively by (i) $dg(\alpha_n) = \alpha_n$ and (ii) $dg(\alpha_{i-1}\alpha_i \cdots \alpha_n) = \alpha_{i-1}dg(\alpha_i\alpha_{i+1} \cdots \alpha_n)$ if there exists an \mathcal{M} -event β in $dg(\alpha_i\alpha_{i+1} \cdots \alpha_n)$ such that $\alpha_i D \beta$ and $dg(\alpha_i\alpha_{i+1} \cdots \alpha_n)$ otherwise. Basically, dg is the so called dependency graph that captures the causal past α_n in σ . We now define $ev(\alpha_1\alpha_2 \cdots \alpha_n)$ to be the trace $= [dg(\alpha_1\alpha_2 \cdots \alpha_n)]$. Finally, the relation $co \subseteq E \times E$ for the event structure $ES_{\mathcal{M}}$ is given by, $co = E \times E \setminus (\leq \cup \geq \cup \#)$. If $e co e'$ this can be interpreted as e and e' being causally independent.

► **Lemma 16.** *Let $\sigma = \alpha_1\alpha_2 \cdots \alpha_n \in L_{\mathcal{M}}^{seq}$ a non-null \mathcal{M} -event sequence in $L_{\mathcal{M}}^{seq}$.*

1. *Then $ev(\sigma)$ is a prime trace and hence is an event in $ES_{\mathcal{M}}$.*
2. *Suppose that $e' \leq e$ in $ES_{\mathcal{M}}$. Then there exists a unique $i \in \{1, 2, \dots, n-1\}$ such that $ev(\alpha_1\alpha_2 \cdots \alpha_i) = e'$.*
3. *Suppose that $e' = ev(\alpha_1\alpha_2 \cdots \alpha_i)$ for some $1 \leq i < n$. Then $e' \leq e$ or $e co e'$ in $ES_{\mathcal{M}}$.*
4. *Suppose that $\alpha_n = (\mathbf{u}, a, \mathbf{v})$ and $\sigma' = \alpha_1\alpha_2 \cdots \alpha_{n-1}\alpha'_n$ such that $\alpha'_n = (\mathbf{u}, a, \mathbf{v}')$ and $\mathbf{v} \neq \mathbf{v}'$. Then $ev(\sigma) \# ev(\sigma')$ in $ES_{\mathcal{M}}$.*

The proof follows from [32]. The first part says that along a path in \mathcal{M} every \mathcal{M} -event corresponds to the occurrence of an event in $ES_{\mathcal{M}}$. The second part says that every event e' that lies in the past of the event e represented by the \mathcal{M} -event sequence σ will appear as the event corresponding to a unique prefix of σ . The third part says if e corresponds to the \mathcal{M} -event sequence σ then every event that corresponds to a strict prefix of σ will either be causally earlier than e or will be causally independent of e in $ES_{\mathcal{M}}$. The last part says that two different stochastic choices made at a state along an \mathcal{M} -path will correspond to conflicting events in $ES_{\mathcal{M}}$.

► **Remark 17.** We conclude by noting that an event $e = [\alpha_1\alpha_2 \dots \alpha_n]$ in $ES_{\mathcal{M}} = (E, \leq, \#, \lambda, \mu)$ gets assigned a probability value via $\mu(e) = \prod_{1 \leq i \leq n} Pr(\alpha_i)$. It is not difficult to provide a measure theoretic justification for this probability value by constructing a σ -algebra generated by the family of cylinder sets $\{CS(e)\}_{e \in E}$ where $CS(e) = \{c \in C_{\max}^{\infty} \mid \downarrow e \subseteq c\}$. Here, C^{∞} is the set of infinite configurations $ES_{\mathcal{M}}$ and $c \in C^{\infty}$ is maximal (i.e. $c \in C_{\max}^{\infty}$) iff $c \subseteq c' \in C^{\infty}$ implies $c = c'$. In other words, c cannot be extended to a larger (infinite) configuration. This distinction between infinite and maximal infinite configurations arises due to concurrency and corresponds to the distinction between complete and incomplete paths. We can define $Pr_{ES}(CS(e)) = \mu(e)$ and show that Pr_{ES} extends canonically to a probability measure over the σ -algebra generated by the above family of cylinder sets. We leave this construction for future work, since we merely need the probability values assigned to the events as common reference points to establish the main result of the next section, namely, all complete strategies determine the same probability values for local reachability properties.

5 The Key Result for CMDPs

Recall that we are given $T \subseteq S_p$ for some component p and aim to determine $\sup_{\pi \in \Pi} Pr_{\mathcal{M}}^{\pi}[\diamond \mathbf{T}]$, where $\mathbf{T} = \{\mathbf{s} \mid \mathbf{s}(p) \in T\}$. In Lem. 10, we argued that it suffices to consider complete strategies to achieve this. Here, we shall show that *all* complete strategies compute the same probability value for $\diamond \mathbf{T}$. This allows us to choose a complete strategy greedily, which in turn enables us to efficiently compute the (optimal) probability of a local reachable set.

We first identify the set of events $E_{\diamond T}$ in the event structure $ES_{\mathcal{M}}$, corresponding to paths in \mathcal{M} reaching T . Let $e = [\alpha_1\alpha_2 \cdots \alpha_n] \in E$ with $\alpha_j = (\mathbf{u}_j, a_j, \mathbf{v}_j)$ for $1 \leq j \leq n$. Then $e \in E_{\diamond T}$ if $\mathbf{v}_n(p) \in T$ and $\mathbf{v}_i(p) \notin T$ for $1 \leq i < n$, in other words, when its last \mathcal{M} -event reaches a member of T and no earlier \mathcal{M} -event in the sequence representing e does so.

To establish the main goal of this section we proceed as follows. For the complete strategy π , we let $Paths_{comp}^\pi$ denote the set of complete paths of the Markov chain \mathcal{M}^π . We then identify, for a given $e \in E_{\diamond T}$, the set of finite paths $\text{PathReach}(\mathcal{M}^\pi, e)$ in \mathcal{M}^π which are prefixes of complete paths and “reach” e . Specifically, suppose $\xi = \rho_0\alpha_1\rho_1\alpha_2 \cdots \rho_{n-1}\alpha_n\rho_n$ is a path in \mathcal{M}^π . Then $\xi \in \text{PathReach}(\mathcal{M}^\pi, e)$ if (i) it is a prefix of a path in $Paths_{comp}^\pi$, (ii) $ev(\alpha_1\alpha_2 \cdots \alpha_n) = e$ and (iii) no strict prefix of ξ satisfies (ii).

We first show that for each $e \in E_{\diamond T}$ it is the case that $\mu(e) = \Pr_{\mathcal{M}}^\pi[\bigcup_{\sigma \in \text{PathReach}(\mathcal{M}^\pi, e)} \sigma]$. (Recall that $\mu(e)$ is the probability value assigned to e in $ES_{\mathcal{M}}$.) We then lift this result to $E_{\diamond T}$ and show that $\sum_{e \in E_{\diamond T}} \mu(e) = \sum_{e \in E_{\diamond T}} \Pr_{\mathcal{M}}^\pi[\text{PathReach}(\mathcal{M}^\pi, e)] = \Pr_{\mathcal{M}}^\pi[\diamond \mathbf{T}]$. Since these results apply to *every* complete strategy π , we are done.

5.1 Relating the Probability of e to the Probability of $\text{PathReach}(\mathcal{M}^\pi, e)$

Through this subsection, fix $e \in E_{\diamond T}$ and a complete strategy π . We wish to prove that $\mu(e) = \Pr_{\mathcal{M}}^\pi[\text{PathReach}(\mathcal{M}^\pi, e)]$. Our proof consists of three steps. First, we represent \mathcal{M}^π as a transition system TS^π by labelling the transitions of the Markov chain with \mathcal{M} -events. Second, we represent $\text{PathReach}(\mathcal{M}^\pi, e)$ as a *finite* prefix of TS^π . Third, we use this finite prefix to establish that $\mu(e) = \Pr_{\mathcal{M}}^\pi[\text{PathReach}(\mathcal{M}^\pi, e)]$.

We begin by deriving the transition system TS^π . The states of TS^π are the states of \mathcal{M}^π (i.e. finite paths in \mathcal{M}). To avoid confusion, we write ρ for these states and ξ for paths in TS^π . Moreover, there is a transition $\rho \xrightarrow{\alpha} \rho'$ iff (i) $\mathcal{M}^\pi(\rho, \rho') > 0$ and (ii) $\alpha = (\mathbf{u}, a, \mathbf{v})$ is the unique \mathcal{M} -event that satisfies $\text{last}(\rho)[a] = \mathbf{u}$ and $\mathbf{s}'[a] = \mathbf{v}$ where $\rho' = \rho a \mathbf{s}'$. In effect, TS^π is obtained from \mathcal{M}^π by replacing the probability “labels” of transitions by the \mathcal{M} -event corresponding to that transition. In particular, note that for a state ρ of TS^π , $a \in \text{Act}(\text{last}(\rho))$ iff there exists an \mathcal{M} -event $\alpha = (\mathbf{u}, a, \mathbf{v})$ such that $\text{last}(\rho)[a] = \mathbf{u}$. Based on this, we can directly transfer the definition of complete paths to TS^π . We define the set of successor states in the obvious way, i.e. $\text{succ}(\rho) = \{\rho' \mid \exists \alpha. \rho \xrightarrow{\alpha} \rho'\}$. Observe that if $\text{succ}(\rho) = \{\rho_1, \rho_2, \dots, \rho_k\}$ and $\rho \xrightarrow{\alpha_i} \rho_i$ for $1 \leq i \leq k$ then there exists an a such that $\text{Act}(\alpha_i) = a$ for every $i \in \{1, 2, \dots, k\}$ and $\sum_{1 \leq i \leq k} Pr(\alpha_i) = 1$. For the rest of this subsection, we work with this transition system.

We now turn to representing $\text{PathReach}(\mathcal{M}^\pi, e)$ as a finite prefix of TS^π . First we introduce some useful terminology. We set $c_0 = \downarrow e$. Next, suppose $\xi = \rho_0\alpha_1\rho_1 \cdots \alpha_n\rho_n$ is a path in TS^π . Then, $EV(\xi) = \{ev(\alpha_1\alpha_2 \cdots \alpha_i) \mid 1 \leq i \leq n\}$ denotes the set of events encountered along the path ξ . Naturally, $EV(\xi) = \emptyset$ if $\xi = \hat{\mathbf{s}}$. We write $G_e = (V, \implies)$ to denote the finite prefix of TS_{comp}^π we are after. We construct G_e inductively by starting with $\hat{\mathbf{s}} \in V$ and mark it as unprocessed. We define ε to be a path in V and $\hat{\mathbf{s}} = \text{last}(\varepsilon)$. We note that $EV(\hat{\mathbf{s}}) = \emptyset \subset c_0$ (as usual, \subset denotes a strict subset).

Suppose $\xi = \rho_0\alpha_1\rho_1 \cdots \rho_{n-1}\alpha_n\rho_n$ is a path in V with ρ_n marked as unprocessed and all the other nodes preceding it in ξ marked as processed. Furthermore, assume that $EV(\xi) \subset c_0$. Let $\text{succ}(\rho_n) = \{\rho'_1, \rho'_2, \dots, \rho'_k\}$ and $\beta_1, \beta_2, \dots, \beta_k$ such that $\rho \xrightarrow{\beta_i} \rho'_i$ for $1 \leq i \leq k$. We now extend G_e by adding the nodes $\rho'_1, \rho'_2, \dots, \rho'_k$ to V and the transitions (ρ, β_i, ρ'_i) for $1 \leq i \leq k$ to \implies . We mark ρ_n as processed. To define the status of the new nodes that have been added, we consider two cases after setting $e_i = ev(\alpha_1\alpha_2 \cdots \alpha_n\beta_i)$ for $1 \leq i \leq k$.

Case 1. Suppose there exists i with $e_i \leq e$. Then $e_i \in c_0 \setminus EV(\xi)$ and hence $EV(\xi\beta_i\rho'_i) = EV(\xi) \cup \{e_i\}$. If $EV(\xi\beta_i\rho'_i) = c_0$ we mark ρ'_i as a *live leaf node* and do not process it any further. This is so since $ev(\xi\beta_i\rho'_i) = e$ and e has been hence reached. We also note that $\alpha_1\alpha_2 \cdots \alpha_n\beta_i \in \text{PathReach}(\mathcal{M}^\pi, e)$. On the other hand, if $EV(\xi\beta_i\rho'_i) \subset c_0$, we mark ρ'_i as unprocessed.

In addition we mark, for each $l \in \{1, 2, \dots, k\} \setminus \{i\}$, the node ρ'_l to be a *dead leaf node* and do not process it any further. To justify this, let $e_l = ev(\xi\beta_l\rho'_l)$ for $l \in \{1, 2, \dots, k\} \setminus \{i\}$. Then clearly $e_i \leq e$ and hence by the last part of Lem. 16, we must have $e_i \# e_l$ for every $l \in \{1, 2, \dots, k\} \setminus \{e_i\}$. But then $e_l \# e_i \leq e$ implies $e_l \# e$ since conflict is inherited via the causality relation in an event structure. Hence e_l and e can not together belong to any configuration and we can never “reach” e by exploring ρ'_l any further.

Case 2. Suppose $e_i \not\leq e$ for each i . Then, by the third part of Lem. 16, we must have e_i *co* e . This implies that $EV(\xi\beta_i\rho'_i) = EV(\xi)$ for each i and we mark each node ρ'_i as unprocessed. The idea is that the chosen action a at ρ_n does not contribute to uncovering any of the events in c_0 and hence all the successors of this node must be further explored.

Starting from the root node we repeatedly apply the above rules until there are no unprocessed nodes left. It remains to be shown that G_e is a *finite* prefix of TS^π and consequently the construction procedure for G_e always terminates. To this end, we require some terminology. Let $\xi^\omega = \rho_0\alpha_1\rho_1\alpha_2 \cdots$ be a complete path in TS^π . For $n \geq 0$, let $\xi^n = \rho_0\alpha_1\rho_1 \cdots \rho_n$ denote the finite prefix of ξ^ω of length n . We say that $\rho_n \xrightarrow{\alpha_{n+1}} \rho_{n+1}$ is a *useful* transition if there exists $e' \in c_0 \setminus EV(\xi^n)$ such that $\pi(\rho_n) = act(e')$. Otherwise it is a *useless* transition. Moreover, we set $EV_e(\xi^n) = EV(\xi^n) \cap c_0$. Finally, we say that $\xi^n = \rho_0\alpha_1\rho_1 \cdots \rho_n$ is a *live* path if (i) ρ_i is not a dead leaf node for $1 \leq i \leq n$ and (ii) $EV_e(\xi^n) \subset c_0$.

► **Lemma 18.** *Suppose $\xi^n = \rho_0\alpha_1\rho_1 \cdots \rho_n$ is a live path.*

1. *If $e' \in \min(c_0) \setminus EV(\xi^n)$ then $act(e') \in Act(\rho_n)$*
2. *$\rho_i \in V$ for $0 \leq i \leq n+1$ and $\rho_j \xrightarrow{\alpha_{j+1}} \rho_{j+1}$ for $0 \leq j < n+1$.*
3. *If $\rho_n \xrightarrow{\alpha_{n+1}} \rho_{n+1}$ is a useful transition, then ρ_{n+1} is a dead leaf node or $|EV_e(\xi^{n+1})| = |EV_e(\xi^n)| + 1$. Further, ρ_{n+1} is a live leaf node if $EV_e(\xi^{n+1}) = c_0$*
4. *If $\rho_n \xrightarrow{\alpha_{n+1}} \rho_{n+1}$ is a useless transition then $EV_e(\xi^{n+1}) = EV_e(\xi^n)$ and ξ^{n+1} is a live path.*

Proof. For the first part, let $e' \in \min(c_0 \setminus EV(\xi^n))$. If $e'' < e'$ then $e'' \in EV(\xi^n)$. Otherwise $e'' \in c_0 \setminus EV(\xi^n)$ which contradicts $e' \in \min(c_0 \setminus EV(\xi^n))$. Thus $c' = EV(\xi^n) \cup \{e'\}$ is a configuration and $EV_e(\xi^n) \xrightarrow{e'}_{ES} c'$. From the first part of Prop. 15 we get $act(e') \in Act(\rho_n)$. The rest follows from the construction rules for G_e and their explanations. ◀

► **Lemma 19.** *The following assertions hold.*

1. *Let $\xi^\omega \in Paths_{comp}^\pi$ with $\xi^n = \rho_0\alpha_1\rho_1 \cdots \rho_n$. Then there exists $k > 0$ such that ρ_k is a live or dead leaf node.*
2. *G_e is a finite tree.*

Proof. From the third part of Lem. 18, it follows that there can be at most $|c_0|$ useful transitions along ξ^ω before a dead or live leaf node is encountered. We now claim that there can be only a finite number of *consecutive* useless moves along ξ^ω . This follows from the first and fourth parts of Lem. 18 and the definition of a complete path. Hence ξ^ω will eventually hit a dead or live node. The second part of the lemma now follows from the first part and König’s lemma since TS^π is finitely branching. ◀

Since G_e is a finite tree it is immediate that its construction procedure always terminates. It is also easy to see that the set of live branches, i.e. paths from the root node to the live leaf nodes in G_e , correspond to $PathReach(\mathcal{M}^\pi, e)$.

For the event e of Fig. 3, our construction produces the tree shown in the left of Fig. 4. The boxes denote dead leaf nodes and the circle is the lone live leaf node. On the other hand, for the event e' , the resulting tree can be arbitrarily large. After the γ event, the strategy

First, we observe that TS^π naturally inherits a probability measure from \mathcal{M}^π . To see this, by the definition of TS^π we are assured that $\rho_0\rho_1\cdots\rho_n$ is a path in \mathcal{M}^π iff $\rho_0 \xrightarrow{\alpha_1} \rho_1 \cdots \rho_{n-1} \xrightarrow{\alpha_n} \rho_n$ is a path in TS^π where the sequence of \mathcal{M} -events $\alpha_1\alpha_2\cdots\alpha_n$ is uniquely determined by the sequence $\rho_1\rho_2\cdots\rho_n$. As a result, the σ -algebra generated by the (cylinder set of) finite paths of TS^π will be in a bijective relation with the usual σ -algebra generated by the finite paths of \mathcal{M}^π . Consequently, we can transfer the probability measure $\Pr_{\mathcal{M}}^\pi$ to a probability measure over the σ -algebra of TS^π . By abuse of notation, we shall denote this measure too as $\Pr_{\mathcal{M}}^\pi$ in what follows.

Now consider $e \in E_{\diamond T}$ and G_e , the finite tree constructed in the previous subsection. Let $Paths_e$ be the set of branches from the root node to the live leaf nodes in G_e . Further, let $CS(\xi)$ be the cylinder set of the finite path ξ in TS^π . Then from the proof of Lem. 20 it follows that $\Pr_{\mathcal{M}}^\pi[\text{PathReach}(\mathcal{M}^\pi, e)] = \Pr_{\mathcal{M}}^\pi[\bigcup_{\xi \in Paths_e} CS(\xi)]$. Consequently, $\Pr_{\mathcal{M}}^\pi[\diamond \mathbf{T}] = \bigcup_{e \in E_{\diamond T}} \bigcup_{\xi \in Paths_e} CS(\xi)$. Since E is a countable set, this probability value is well-defined. To show that this value is the same for all complete strategies, we establish that $\bigcup_{e \in E_{\diamond T}} \bigcup_{\xi \in Paths_e} CS(\xi) = \sum_{e \in E_{\diamond T}} \mu(e)$. The key to doing this is the next result.

► **Lemma 21.** *Let $e_1, e_2 \in E_{\diamond T}$ such that $e_1 \neq e_2$. Then $e_1 \# e_2$.*

Proof. Let $e_1 = [\alpha_1\alpha_2\cdots\alpha_n]$ and $e_2 = [\beta_1\beta_2\cdots\beta_m]$. If $e_1 < e_2$, then there exists $i < n$ such that $ev(\beta_1\beta_2\cdots\beta_i) = e_1$. But this contradicts the requirement that α_n is the *first* \mathcal{M} -event in the sequence $\alpha_1\alpha_2\cdots\alpha_n$ with $\mathbf{v}_n(p) \in T$ where $\alpha_j = (\mathbf{u}_j, a_j, \mathbf{v}_j)$ for $1 \leq j \leq n$. Thus $e_1 \not\prec e_2$ and similarly $e_2 \not\prec e_1$. Next suppose $e_1 \text{ co } e_2$.

Then $c_{12} = \downarrow e_1 \cup \downarrow e_2$ is a configuration. To see this, let x and y be events such that $x \in c_{12}$ and $y \leq x$. Suppose $x \in \downarrow e_1$. Then $y \in \downarrow e_1 \subseteq c_{12}$. Similarly, $x \in \downarrow e_2$ implies that $y \in c_{12}$. Next, suppose that $x \# y$. Then, it can not be the case that x, y are both in $\downarrow e_1$ or $\downarrow e_2$ since both $\downarrow e_1$ and $\downarrow e_2$ are configurations and hence conflict-free. Hence, assume that $x \in \downarrow e_1$ and $y \in \downarrow e_2$. Then $x \leq e_1$ and $y \leq e_2$, which implies that $e_1 \# e_2$, contradicting $e_1 \text{ co } e_2$. Thus c_{12} indeed is a configuration.

This implies that $\downarrow e_1 \uparrow \downarrow e_2$. Hence by the last part of Prop. 12, there exist \mathcal{M} -event sequences $\gamma_1\gamma_2\cdots\gamma_l$, $\alpha'_1\alpha'_2\cdots\alpha'_{n'}$, and $\beta'_1\beta'_2\cdots\beta'_{m'}$ such that (i) $\gamma_1\gamma_2\cdots\gamma_l\alpha'_1\alpha'_2\cdots\alpha'_{n'} \approx \alpha_1\alpha_2\cdots\alpha_n$, (ii) $\gamma_1\gamma_2\cdots\gamma_l\beta'_1\beta'_2\cdots\beta'_{m'} \approx \beta_1\beta_2\cdots\beta_m$, and (iii) $\alpha'_i \perp \beta'_j$ for $1 \leq i \leq n'$ and $1 \leq j \leq m'$. Since $[\alpha_1\alpha_2\cdots\alpha_n]$ and $[\beta_1\beta_2\cdots\beta_m]$ are both prime traces we must have $\alpha'_{n'} = \alpha_n$ and $\beta'_{m'} = \beta_m$. This leads to $\alpha_n \perp \beta_m$, which is a contradiction since $p \in \text{loc}(\text{act}(\alpha_n)) \cap \text{loc}(\text{act}(\beta_m))$ and hence $\alpha_n \text{ D } \beta_m$. ◀

► **Lemma 22.** $\Pr_{\mathcal{M}}^\pi[\diamond \mathbf{T}] = \sum_{e \in E_{\diamond T}} \mu(e)$.

Proof. We have $\Pr_{\mathcal{M}}^\pi[\diamond \mathbf{T}] = \Pr_{\mathcal{M}}^\pi[\bigcup_{e \in E_{\diamond T}} \bigcup_{\xi \in Paths_e} CS(\xi)]$ from the remarks preceding Lem. 20, where $Paths_e$ is the set of branches from the root node to live leaf nodes in G_e , the finite tree constructed in the proof of Lem. 20. Let $e_1, e_2 \in E_{\diamond T}$ such that $e_1 \neq e_2$. Then $e_1 \# e_2$ by Lem. 21. Let $\xi_1 \in Paths_{e_1}$ and $\xi_2 \in Paths_{e_2}$. Then from the definition of $Paths_e$ it follows directly that $CS(\xi_1) \cap CS(\xi_2) = \emptyset$. This implies that $\Pr_{\mathcal{M}}^\pi[\diamond \mathbf{T}] = \sum_{e \in E_{\diamond T}} \Pr_{\mathcal{M}}^\pi[\bigcup_{\xi \in Paths_e} CS(\xi)]$. From Lem. 20 we get $\Pr_{\mathcal{M}}^\pi[\diamond \mathbf{T}] = \sum_{e \in E_{\diamond T}} \mu(e)$. ◀

This at once leads to our main result.

► **Theorem 23.** *Let π and π' be two deterministic complete strategies for the CMDP \mathcal{M} . Then $\Pr_{\mathcal{M}}^\pi[\diamond \mathbf{T}] = \Pr_{\mathcal{M}}^{\pi'}[\diamond \mathbf{T}]$.*

Combined with Lem. 10, we have that in order to compute the optimal local reachability value, we can confine ourselves to complete strategies and from among them, greedily choose one.

6 Implementation and Experimental Evaluation

We implemented a prototype tool and evaluated it on a few models, as we describe in the following. The tool is written in Java and based on PET [22]. It uses PRISM [18] to parse models. We used the pure-Java library `oj! Algorithms` to solve linear programs. We empirically validated the soundness of our implementation by comparing its output on about 20 models to the results of Storm [8] in its sound, exact mode. The tool, its source code, all used models, and further models can be obtained from [23].

6.1 Algorithm Description

Our tool (i) provides a syntactic over-approximation for checking the CD property, and (ii) computes the maximal reachability probability of a local reachability set, assuming that the input MDP is a CMDP. In the interest of space, we only sketch the computation procedure here. More details and a formal description can be found in App. A. Intuitively, the goal is to construct only the part of the system that is reached by one specific complete strategy, chosen as follows. First, we heuristically fix a priority over the set of actions. Then, we begin exploring the global FMDP by starting in the initial state, picking the available action with the highest priority, and determine all its stochastic successors. We repeat this process for the discovered successors until a fixpoint of states is reached. One must however ensure that this greedy prioritization avoids neglecting an available action forever. To this end, we check in each *bottom maximal end component* whether any available action is never chosen. If so, we pick, for each bottom component, the constantly omitted action with highest priority and explore as above. Eventually, this process will terminate with no bottom MECs having any omitted action. Then, we determine the maximal reachability probability of the target local state on the constructed subsystem. In our implementation we use the standard linear program for reachability (see, e.g., [2, Thm. 10.105]). We note that for the computation, CD is only required for correctness, not for termination.

Our implementation is quite simplistic and can be optimized in multiple ways. In particular, the priority order of actions will have a large influence on the size of the resulting subsystem, and this could be significantly improved by intelligent adaptive techniques and learning-based approaches. However, our current heuristical ordering already provides convincing results. Hence we did not explore this issue further.

6.2 Setup and Results

We consider four types of models, each of which was either constructed from scratch or obtained by adapting an existing model to fit into our framework. Unfortunately, most models of the PRISM benchmark suite [19] are not immediately CD and one needs to examine which ones can be adapted to fit into our framework, which we leave for future work. We provide a brief intuitive description of the models we used. The concrete specification in the PRISM modelling language can be found in [23]. The **sync** model consists of 20 processes running in parallel, each repeatedly tossing a (biased) coin and progressing when head is obtained, and finally synchronizing on a common action with all other processes to reach their final states. We next consider a variant of the classical **dining philosophers**, where philosophers alternate between eating and thinking. In our variant, the thinking process of each philosopher has several (probabilistic) steps with each philosopher initially “musing” and eventually becoming “enlightened” or “bewildered”, and we seek the probability of one philosopher achieving

■ **Table 1** Overview of results for our four models. From left to right, we list the model name, its overall size (as reported by Storm), the runtime of Storm with sparse and symbolic engine, respectively, the size of the reduced model constructed by our tool, and the overall runtime of our tool. T/O denotes a runtime of over 5 minutes. We also carried out a comparison to PRISM, however Storm was faster in all cases.

Model	Size	Storm-sparse	Storm-symbolic	Reduced Size	Our Tool
sync	$2.1 \cdot 10^6$	17s	2s	22	2s
philosophers	$8.6 \cdot 10^6$	T/O	T/O	3264	4s
production	$6.6 \cdot 10^7$	T/O	T/O	11669	8s
scheduling	$2.8 \cdot 10^{10}$	T/O	T/O	111	<1s
scheduling (large)	??	T/O	T/O	1021	3s

enlightenment. The **production** model comprises a production network where resources are used to assemble (through several steps) a final product. Resources have a chance of becoming exhausted every time they are mined and we are interested in the probability of producing a given quantity of the final product. Finally, **scheduling** models a central process C which proceeds in ten stages. In stage i , the process needs to synchronize with the process p_i to proceed to stage $i + 1$. The sub-processes are independent, but may fail to complete. We are interested in the probability of the central process finishing the final stage. For scalability analysis, we also consider a “large” variant where C has 20 stages and each p_i has 50 sequential steps.

We executed our tool on standard hardware and compared our results with those obtained using the model checker Storm. We considered both the default sparse as well as symbolic engines of Storm and otherwise let Storm run in its default configuration. Notably, we did not require exact or sound results (i.e. Storm could decide to use classical, unsound value iteration), while our tool computed correct, exact results using linear programming (up to floating point precision). We summarize our findings in Table 1. One can see that our (basic, unoptimized) approach significantly outperforms existing approaches on the chosen models. This improvement is due to our method being able to avoid visiting a lot number of “useless” states by not exploring every interleaving. On the “large” variant of **scheduling**, Storm fails to even output a state count, which we estimate to be of the order of 50^{20} ($\approx 10^{34}$).

7 Conclusion

We introduced a class of factored MDPs where through the notion of locations, we cleanly separate the causality, concurrency, and conflict relations between the stochastic events in the system. This leads to an event structure semantics for our FMDPs. We mainly used this representation to provide the basis for a powerful partial order based quantitative analysis technique for CMDPs, a natural subclass of FMDPs.

In the future, we plan to study the class of CMDPs from the standpoint of expressiveness. In particular it will be interesting to separate CMDPs from FMDPs that *inherently* do not have the CD property but are unavoidable in practice. Here we suspect that the property called confusion-freeness will play an important role [30]. We also wish to emphasize that the class of FMDPs we identify and their event structure semantics are of independent interest. In particular, it opens up the possibility of using techniques such as finite prefixes of event structures [11] and stubborn sets [15] to analyze FMDPs. These techniques can be applied for model checking the FMDPs for probabilistic temporal logical specifications. To secure the foundations for doing so, the probability measure for events structures that was alluded to at the end of Sec. 4 will need to be fleshed out.

Our experiments suggest that the presented method has significant potential for practical applicability, especially in light of the fact that the method itself can be improved and extended in multiple ways; for instance, by considering reachability properties for a small number of components or by formulating weaker versions of the CD property.

References

- 1 Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Information and Computation*, 204(2):231–274, 2006. doi:10.1016/j.ic.2005.10.001.
- 2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 3 Marco Beccuti, Giuliana Franceschinis, and Serge Haddad. Markov decision petri net and markov decision well-formed net formalisms. In Jetty Kleijn and Alexandre Yakovlev, editors, *Petri Nets and Other Models of Concurrency – ICATPN 2007, 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings*, volume 4546 of *Lecture Notes in Computer Science*, pages 43–62. Springer, 2007. doi:10.1007/978-3-540-73094-1_6.
- 4 Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1104–1113. Morgan Kaufmann, 1995. URL: <http://ijcai.org/Proceedings/95-2/Papers/012.pdf>.
- 5 Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. JANI: quantitative model and tool interaction. In *Tools and Algorithms for the Construction and Analysis of Systems – 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II*, volume 10206 of *Lecture Notes in Computer Science*, pages 151–168, 2017. doi:10.1007/978-3-662-54580-5_9.
- 6 Luca de Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford University, USA, 1997. URL: <https://searchworks.stanford.edu/view/3910936>.
- 7 Luca de Alfaro, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of probabilistic processes using mtbdds and the kronecker representation. In Susanne Graf and Michael I. Schwartzbach, editors, *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 – April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2000. doi:10.1007/3-540-46419-0_27.
- 8 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *Computer Aided Verification – 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600. Springer, 2017. doi:10.1007/978-3-319-63390-9_31.
- 9 Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, 1995. doi:10.1142/2563.
- 10 Javier Esparza. Decidability and complexity of Petri net problems – an introduction. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer, 1996. doi:10.1007/3-540-65306-6_20.

- 11 Javier Esparza and Keijo Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008. doi:10.1007/978-3-540-77426-6.
- 12 Robert Givan, Thomas L. Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artif. Intell.*, 147(1-2):163–223, 2003. doi:10.1016/S0004-3702(02)00376-4.
- 13 Marcus Größer and Christel Baier. Partial order reduction for Markov decision processes: A survey. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 408–427. Springer, 2005. doi:10.1007/11804192_19.
- 14 Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *J. Artif. Intell. Res.*, 19:399–468, 2003. doi:10.1613/JAIR.1000.
- 15 Henri Hansen, Marta Z. Kwiatkowska, and Hongyang Qu. Partial order reduction for model checking Markov decision processes under unconditional fairness. In *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*, pages 203–212. IEEE Computer Society, 2011. doi:10.1109/QEST.2011.35.
- 16 Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods Syst. Des.*, 36(3):246–280, 2010. doi:10.1007/S10703-010-0097-6.
- 17 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models – Principles and Techniques*. MIT Press, 2009. URL: <http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11886>.
- 18 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification – 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011. doi:10.1007/978-3-642-22110-1_47.
- 19 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012*, pages 203–204. IEEE Computer Society, 2012. doi:10.1109/QEST.2012.14.
- 20 Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. Modelling with generalized stochastic petri nets. *SIGMETRICS Perform. Evaluation Rev.*, 26(2):2, 1998. doi:10.1145/288197.581193.
- 21 Antoni W. Mazurkiewicz. Introduction to trace theory. In Volker Diekert and Grzegorz Rozenberg, editors, *The Book of Traces*, pages 3–41. World Scientific, 1995. doi:10.1142/9789814261456_0001.
- 22 Tobias Meggendorfer. PET – A partial exploration tool for probabilistic verification. In *Automated Technology for Verification and Analysis – 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings*, volume 13505 of *Lecture Notes in Computer Science*, pages 320–326. Springer, 2022. doi:10.1007/978-3-031-19992-9_20.
- 23 Tobias Meggendorfer. Causally deterministic markov decision processes, July 2024. Software (visited on 2024-08-20). doi:10.5281/zenodo.12657579.
- 24 Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theor. Comput. Sci.*, 13:85–108, 1981. doi:10.1016/0304-3975(81)90112-2.
- 25 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. doi:10.1002/9780470316887.
- 26 Wolfgang Reisig. *Understanding Petri Nets – Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. doi:10.1007/978-3-642-33278-4.
- 27 Brigitte Rozoy and P. S. Thiagarajan. Event structures and trace monoids. *Theor. Comput. Sci.*, 91(2):285–313, 1991. doi:10.1016/0304-3975(91)90087-I.

- 28 Ratul Saha, Javier Esparza, Sumit Kumar Jha, Madhavan Mukund, and P. S. Thiagarajan. Distributed Markov chains. In Deepak D’Souza, Akash Lal, and Kim Guldstrand Larsen, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 117–134, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-46081-8_7.
- 29 P. S. Thiagarajan and Shaofa Yang. A theory of distributed Markov chains. *Fundam. Informaticae*, 175(1-4):301–325, 2020. doi:10.3233/FI-2020-1958.
- 30 Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. *Theoretical Computer Science*, 358(2):173–199, 2006. Concurrency Theory (CONCUR 2004). doi:10.1016/j.tcs.2006.01.015.
- 31 Kazuki Watanabe, Clovis Eberhart, Kazuyuki Asada, and Ichiro Hasuo. Compositional probabilistic model checking with string diagrams of MDPs. In *Computer Aided Verification – 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 40–61. Springer, 2023. doi:10.1007/978-3-031-37709-9_3.
- 32 Glynn Winskel and Mogens Nielsen. *Models for concurrency*, pages 1–148. Oxford University Press, Inc., USA, 1995.

A Algorithm Description

In this section, we provide a more detailed description of our algorithmic approach. We assume that we are given the description of each process in an MDP network. As mentioned above, our tool reads models given in the PRISM language, which introduces additional modes of synchronization. For example, guards and updates can read the value of other processes’ states without explicitly synchronizing with them.

Checking Causal Determinacy

To syntactically check whether a given model is CD, we check for every local state of every process and every pair of actions available for that process that the intersection of the action guards is empty. This directly implies that the model is CD. However, this is also an over-approximation, since a potential violation might not be reachable in the actual system. All models except the **philosophers** model directly satisfy this simple syntactic property. For the model, **philosophers** model we verified the CD property by manual inspection.

Constructing a Complete Strategy

As mentioned in the main body, our first goal is to heuristically fix a priority order on the available actions. To this end we first record all “dependencies” between processes, i.e. whenever a process reads from or synchronizes with another process, we add an edge in the module dependency graph. Then, starting from the process for which we have the local reachability query, we explore this dependency graph in a breadth-first fashion and order the processes according to this search. We then derive the action priority as follows: We iterate over the processes in the above order, and consider each action this process is involved in which has not yet been processed (i.e. all actions a for which the current module has the highest priority among all processes in $\text{loc}(a)$). These actions are then sorted according to the process with the lowest priority among all of those involved with the action, i.e. $\text{loc}(a)$. This then gives us the overall priority ordering over all actions appearing in the system.

The second part then is to construct a sub-system of the global FMDP which contains at least one complete strategy. By computing the maximal reachability probability on this sub-system, we obtain the overall maximal reachability probability, as any complete strategy

is optimal under CD. To this end, we start in the global initial state and explore the graph induced by the following rule: (i) In a state \mathbf{s} , compute the set of available actions $Act(\mathbf{s})$. (ii) Among those actions, pick the action with the highest priority according to the determined order. (iii) Return the set of successors under this action. We fully explore the system induced by this transition relation using BFS. In other words, we explore the sub-system induced by greedily following actions according to our priority order.

As already mentioned, this alone does not guarantee that we get a complete strategy: For example, it might be the case that the highest priority action a available in some state \mathbf{s} simply self-loops, but another action b (with lower priority) would lead to a new successor \mathbf{s}' . To ensure this, we determine the set of bottom maximal end components, i.e. all regions where the strategy we are following is “looping”. Let R be a set of states forming such an end component in the explored sub-system and for every state \mathbf{s} let $\mathbf{A}(\mathbf{s})$ the action we chose according to our greedy rule. We then compute $\mathcal{A}(R) = \bigcup_{\mathbf{s} \in R} Act(\mathbf{s}) \setminus \bigcup_{\mathbf{s} \in R} \mathbf{A}(\mathbf{s})$. When $\mathcal{A}(R) = \emptyset$, we are finished with the end component R . If not, we pick for each bottom end component R with $\mathcal{A}(R) \neq \emptyset$ the action with the highest priority from $\mathcal{A}(R)$ according to our priority rule and again apply the exploration rule from above.

Correctness

We argue that the subsystem explored in this way contains a complete strategy, independent of the action priority used, by explicitly constructing one. Let \mathcal{B} the set of states in bottom maximal end components in the explored sub-system. Let π a strategy that (i) reaches \mathcal{B} with probability 1 and (ii) uses each action available in \mathcal{B} infinitely often with probability 1 (e.g., by using round-robin memory). Such a strategy exists due to standard results on the properties of end components [2, Chapter 10], [6]. We claim that this strategy is complete.

Assume for contradiction that it is not, i.e. the set of incomplete paths under this strategy has non-zero measure. Since the set of state-action pairs is finite, there exists at least one pair (\mathbf{s}, a) which is “responsible” for the incompleteness. In other words, under the strategy we reach (after a finite number of steps) a state \mathbf{s} where a is available, but from that point onward we never see a with some non-zero probability. Formally, there exists (\mathbf{s}, a) and index i such that $\mathcal{P} = \{\varrho \mid \mathbf{s} = \varrho_i \wedge \forall j \geq i. A(\varrho, j) \neq a\}$ has non-zero measure (where $A(\varrho, j)$ denotes the action in path ϱ at step j). Observe that by the CD condition, for the paths in \mathcal{P} the action a is available at all subsequent states after i .

Next, let $\text{Inf}(\varrho) \subseteq \mathbf{S}$ the set of states visited infinitely often by path ϱ . Consider the (finite) partitioning of \mathcal{P} by Inf , i.e. grouping runs that visit the same set of states infinitely often. By additivity of $\text{Pr}_{\mathcal{M}}^\pi$, there exists at least one partition S_∞ that has non-zero measure. Thus, by the definition of π , S_∞ is a subset of \mathcal{B} : Almost all paths under π end up in \mathcal{B} , so there can be no non-zero measure set that does not.

To conclude, recall that a is available on all states of all paths in \mathcal{P} , including all paths in S_∞ . Let R a maximal bottom end component in the explored subsystem (i.e. $R \subseteq \mathcal{B}$) with a non-empty intersection with S_∞ . By the definition of π , almost all paths of \mathcal{P} that end up in R visit all states of R infinitely often. Together, a must be available in all states of R , but is never chosen by the strategy π . However, by construction, we would have explored a , as it is an available action in a bottom end component of the subsystem. Concretely, we have that $\mathcal{A}(R)$ is not empty, hence we would explore further, contradicting that R is a bottom end component. This concludes the proof.

B Proof of Lemma 10

► **Lemma 10.** *There exists a deterministic, complete strategy $\pi \in \Pi$ which achieves the optimal value, i.e. $\Pr_{\mathcal{M}}^{\pi}[\diamond \mathbf{T}] = \sup_{\pi' \in \Pi} \Pr_{\mathcal{M}}^{\pi'}[\diamond \mathbf{T}]$.*

Proof. We show this by arguing that an optimal, possibly non-complete strategy can be modified into a complete one without losing any reachability probability. To this end, let π a *memoryless* deterministic strategy that achieves the optimal value. Assume this strategy is incomplete. We now show how to extend it to a complete strategy. Consider the bottom strongly connected components $\mathcal{B} = \{B_1, \dots, B_n\}$ in the induced Markov chain \mathcal{M}^{π} . With probability 1, these are eventually reached (i.e. $\Pr_{\mathcal{M}}^{\pi}[\diamond \bigcup B_i] = 1$), and, likewise, once in a BSCC B_i , every state within it is reached with probability 1 [2, Chp. 10]. Consider the following strategy π' : Follow π , waiting until one of the BSCCs B_i is reached. Meanwhile, track a set of actions A . At each state s , add all actions $Act(s)$ to A and then remove $\pi(s)$. In other words, A tracks all actions that were available but have not been played since they became available. Then, wait until every state in B_i was seen at least once. Until now, π' has behaved exactly as π and has only stored a bounded amount of information.

At this stage π' switches to a different behaviour. Store the set of actions A which have not been played to A' and clear A . By CD, all actions in A are still available. So, π' chooses the actions in A' one by one, and, in the meantime, keeps updating A as before. Once A' is empty, again A is copied to A' , A is cleared and the whole process is repeated. (If A is empty at this stage, π simply picks any action.)

This strategy clearly reaches every state that π reaches with at least the same probability, since π' only deviates from π once all states that π can see have been encountered. In addition, this strategy is complete since every action that is available is played within a finite number of steps with probability 1. ◀



Fairness and Consensus in an Asynchronous Opinion Model for Social Networks

Jesús Aranda  

Universidad del Valle, Colombia

Sebastián Betancourt  

Universidad del Valle, Colombia

Juan Fco. Díaz  

Universidad del Valle, Colombia

Frank Valencia 

CNRS LIX, École Polytechnique de Paris, France

Pontificia Universidad Javeriana Cali, Colombia

Abstract

We introduce a DeGroot-based model for opinion dynamics in social networks. A community of agents is represented as a weighted directed graph whose edges indicate how much agents influence one another. The model is formalized using labeled transition systems, henceforth called *opinion transition systems (OTS)*, whose states represent the agents' opinions and whose actions are the edges of the influence graph. If a transition labeled (i, j) is performed, agent j updates their opinion taking into account the opinion of agent i and the influence i has over j . We study (*convergence to*) *opinion consensus* among the agents of strongly-connected graphs with influence values in the interval $(0, 1)$. We show that consensus cannot be guaranteed under the standard *strong fairness* assumption on transition systems. We derive that consensus is guaranteed under a stronger notion from the literature of concurrent systems; *bounded fairness*. We argue that bounded-fairness is too strong of a notion for consensus as it almost surely rules out random runs and it is not a constructive liveness property. We introduce a weaker fairness notion, called *m-bounded fairness*, and show that it guarantees consensus. The new notion includes almost surely all random runs and it is a constructive liveness property. Finally, we consider OTS with *dynamic influence* and show convergence to consensus holds under *m-bounded fairness* if the influence changes within a fixed interval $[L, U]$ with $0 < L < U < 1$. We illustrate OTS with examples and simulations, offering insights into opinion formation under fairness and dynamic influence.

2012 ACM Subject Classification Theory of computation → Social networks

Keywords and phrases Social networks, fairness, DeGroot, consensus, asynchrony

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.7

Related Version *Technical Report with proofs*: <https://arxiv.org/abs/2312.12251> [7]

Supplementary Material *Software (Source python code for simulations)*: <https://github.com/promueva/Fairness-and-Consensus-in-Opinion-Models>

archived at `swh:1:dir:3d1d063e991e22e10ce933f8b060dcb8f1703702`

Funding This work is partly supported by the Colombian Minciencias project PROMUEVA, BPIN 2021000100160.

1 Introduction

Social networks have a strong impact on *opinion formation*, often resulting in polarization. Broadly, the dynamics of opinion formation in social networks involve users expressing their opinions, being exposed to the opinions of others, and potentially adapting their own views based on these interactions. Modeling these dynamics enables us to glean insights into how opinions form and spread within social networks.



© Jesús Aranda, Sebastián Betancourt, Juan Fco. Díaz, and Frank Valencia; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 7; pp. 7:1–7:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The models of social learning aim to capture opinion dynamics in social networks [36]. The DeGroot model [14] is one of the most prominent formalisms for social learning and opinion formation dynamics, and it remains a continuous focus of study in social network theory [21]. A given community is represented as a weighted directed graph, known as the *influence graph*, whose edges indicate how much individuals (*agents*) influence one another. Each agent has an opinion represented as a value in $[0, 1]$, indicating the strength of their agreement with an underlying proposition (e.g., “*AI poses a threat to humanity*”). Agents repetitively revise their opinions by averaging them with those of their contacts, taking into account the influence each contact holds. (There is empirical evidence validating the opinion formation through averaging of the model in controlled sociological experiments, e.g., [10].) A fundamental theoretical result of the model states that the agents will *converge to consensus* if the influence graph is *strongly connected* and the agents have non-zero self-influence (*puppet freedom*) [21]. The significance of this result lies in the fact that consensus is a central problem in social learning. Indeed, the inability to reach consensus is a sign of a polarized community.

Nevertheless, the DeGroot model makes at least two assumptions that could be overly constraining within social network contexts. Firstly, it assumes that all the agents update their opinions simultaneously (*full synchrony*), and secondly, it assumes that the influence of agents remains the same throughout opinion evolution (*static influence*). These assumptions may hold in some controlled scenarios and render the model tractable but in many real-world scenarios individuals do not update their opinions simultaneously [29]. Instead, opinion updating often occurs *asynchronously*, with different agents updating their opinions at different times. Furthermore, individuals may gain or lose influence through various factors, such as expressing contrarian or extreme opinions [20].

In this paper, we introduce an *asynchronous* DeGroot-based model with *dynamic influence* to reason about opinion formation, building upon notions from concurrency theory. The model is presented by means of labeled transition systems, here called *opinion transition systems (OTS)*. The states of an OTS represent the agents’ opinions, and the actions (labels) are the edges of the influence graph. All actions are *always* enabled. If a transition labeled with an edge (i, j) is chosen, agent j updates their opinion by averaging it with the opinion of agent i weighted by the influence that this agent carries over j . A *run* of an OTS is an infinite sequence of (chosen) transitions.

We shall focus on the problem of convergence to opinion consensus in runs of the OTS, *assuming* strong connectivity of the influence graph and puppet freedom. For consensus to make sense, all agents should have the chance to update their opinions. Therefore, we need to make *fairness* assumptions about the runs. In concurrency theory, this means requiring that some actions be performed sufficiently often.

We first show that contrary to the DeGroot model, consensus *cannot* be guaranteed for runs of OTS even under the standard *strong fairness* assumption (i.e., that each action occurs infinitely often in the run) [22, 27]. This highlights the impact of asynchronous behavior on opinion formation.

We then consider the well-known notion of *bounded fairness* in the literature on verification of concurrent systems [16]. This notion requires that every action must be performed not just eventually but within some bounded period of time. We show that bounded-fairness guarantees convergence to consensus. This also gives us insight into opinion formation through averaging, i.e., preventing unbounded delays of actions (opinion updates) is sufficient for convergence to consensus.

Nevertheless, bounded fairness does not have some properties one may wish in a fairness notion. In particular, it is not a *constructive liveness* property in the sense of [34, 33]. Roughly speaking, a fairness notion is a constructive liveness property if, while it may require that a particular action is taken sufficiently often, it should not prevent any other action from being taken sufficiently often. Indeed, we will show that preventing unbounded delays implies preventing some actions from occurring sufficiently often.

Furthermore, bounded-fairness is not *random inclusive*. A fairness notion is random inclusive if any random run (i.e., a run where each action is chosen independently with non-zero probability) is *almost surely* fair under the notion. We find this property relevant because we wish to apply our results to other asynchronous randomized models whose runs are random and whose opinion dynamics can be captured as an OTS.

We therefore introduce a new weaker fairness notion, called *m-bounded fairness*, and show that it guarantees consensus. The new notion is shown to be a constructive liveness property and random inclusive. We also show that consensus is guaranteed under *m-bounded fairness* even if we allow for *dynamic influence* as long as all the changes of influence are within a fixed interval $[L, U]$ with $0 < L < U < 1$.

All in all, we believe that asynchronous opinion updates and dynamic influence provide us with a model more faithful to reality than the original DeGroot model. The fairness assumptions and consensus results presented in this paper show that the model is also tractable and that it brings new insights into opinion formation in social networks. To the best of our knowledge, this is the first work using fairness notions from concurrency theory in the context of opinion dynamics in social networks.

Furthermore, since *m-bounded fairness* is random inclusive, our result extends with dynamic influence the consensus result in [17] for distributed averaging with randomized gossip algorithms. Distributed averaging is a central problem in other application areas, such as decentralized computation, sensor networks and clock synchronization.

Organization. The paper is organized as follows: In Section 2, we introduce OTS and the consensus problem. Initially, to isolate the challenges of asynchronous communication in achieving consensus, we assume static influence. In Section 3, we identify counter-examples, graph conditions, and fairness notions for consensus to give some insight into opinion dynamics. In Section 4, we introduce a new notion of fairness and state our first consensus theorem. Finally, in Section 5, we add dynamic influence and give the second consensus theorem.

The detailed proofs are included in a related technical report [7]. The Python code used to produce OTS examples and simulations in this paper can be found in the following repository: <https://github.com/promueva/Fairness-and-Consensus-in-Opinion-Models>.

2 The Model

In the standard DeGroot model [14], agents update their opinion *synchronously* in the following sense: at each time unit, all the agents (individuals) update simultaneously their current opinion by listening to the current opinion values of those who influence them. This notion of updating may be unrealistic in some social network scenarios, as individuals may listen to (or read) others' opinions at different points in time.

In this section, we introduce an opinion model where individuals update their beliefs asynchronously; one agent at a time updates their opinion by listening to the opinion of one of their influencers.

2.1 Opinion Transition Systems

In social learning models, a *community* is typically represented as a directed weighted graph with edges between individuals (agents) representing the direction and strength of the influence that one has over the other. This graph is referred to as the *Influence Graph*.

► **Definition 1** (Influence Graph). *An influence graph is a directed weighted graph $G = (A, E, I)$ with $A = \{1, \dots, \mathbf{n}\}$, $\mathbf{n} > 1$, the vertices, $E \subseteq A^2 - Id_A$ the edges (where Id_A is the identity relation on A) and $I : E \rightarrow (0, 1]$ the weight function.*

The vertices in A represent \mathbf{n} agents of a given community or network. The set of edges E represents the (direct) influence relation between agents; i.e., $(i, j) \in E$ means that agent i influences agent j . The value $I(i, j)$, for simplicity written $I_{(i,j)}$ or I_{ij} , denotes the strength of the influence: a higher value means stronger influence.

Similar to the DeGroot-like models in [21], we model the evolution of agents' opinions about some underlying *statement* or *proposition*, such as, for example, “*human activity has little impact on climate change*” or “*AI poses a threat to humanity*”.

The *state of opinion* (or *belief state*) of all the agents is represented as a *vector* in $[0, 1]^{|A|}$. If \mathbf{B} is a state of opinion, $\mathbf{B}[i]$ denotes the *opinion* (*belief*, or *agreement*) value of agent $i \in A$ regarding the underlying proposition: the higher the value of $\mathbf{B}[i]$, the stronger the agreement with such a proposition. If $\mathbf{B}[i] = 0$, agent i completely *disagrees* with the underlying proposition; if $\mathbf{B}[i] = 1$, agent i completely *agrees* with the underlying proposition.

The opinion state is updated as follows: Starting from an initial state, at each time unit, one of the agents, say j , updates their opinion taking into account the influence and the opinion of one of their contacts, say i . Intuitively, in social network scenarios, this can be thought of as having an agent j read or listen to the opinion of one of their influencers i and adjusting their opinion $\mathbf{B}[j]$ accordingly.

The above intuition can be realized as a *Labelled Transition System* (LTS) whose set of states is $S = [0, 1]^{|A|}$ and set of *actions* is E .

► **Definition 2** (OTS). *An Opinion Transition System (OTS) is a tuple $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ where $G = (A, E, I)$ is an influence graph, $\mathbf{B}_{\text{init}} \in S = [0, 1]^{|A|}$ is the initial opinion state, and $\rightarrow \subseteq S \times E \times S$ is a (labelled) transition relation defined thus: $(\mathbf{B}, (i, j), \mathbf{B}') \in \rightarrow$, written $\mathbf{B} \xrightarrow{(i,j)} \mathbf{B}'$, iff for every $k \in A$,*

$$\mathbf{B}'[k] = \begin{cases} \mathbf{B}[j] + (\mathbf{B}[i] - \mathbf{B}[j])I_{ij} & \text{if } k = j \\ \mathbf{B}[k] & \text{otherwise} \end{cases} \quad (1)$$

If $\mathbf{B} \xrightarrow{e} \mathbf{B}'$ we say that \mathbf{B} evolves into \mathbf{B}' by performing (choosing or executing) the action e .

A labeled transition $\mathbf{B} \xrightarrow{(i,j)} \mathbf{B}'$ represents the opinion evolution from \mathbf{B} to \mathbf{B}' when choosing an action represented by the edge (i, j) . As a result of this action, agent j updates their opinion as $\mathbf{B}[j] + (\mathbf{B}[i] - \mathbf{B}[j])I_{ij}$, thereby moving closer to the opinion of agent i . Alternatively, think of agent i as pulling the opinion of agent j towards $\mathbf{B}[i]$. The higher the influence of i over j , I_{ij} , the closer it gets. Intuitively, if $I_{ij} < 1$, it means that agent j is *receptive* to agent i but offers certain *resistance* to fully adopting their opinion. If $I_{ij} = 1$, agent j may be viewed as a *puppet* of i who disregards (or forgets) their own opinion to adopt that of i .

► **Remark 3.** In Def. 1, we do not allow edges of the form (j, j) . In fact, allowing them would *not* present us with any additional technical issues, and the results in this paper would still hold. The reason for this design choice, however, has to do with clarity about

the intended intuitive meaning of a transition. Suppose that $\mathbf{B} \xrightarrow{(i,j)} \mathbf{B}'$. Since $\mathbf{B}'[j] = \mathbf{B}[j] + (\mathbf{B}[i] - \mathbf{B}[j])I_{ij} = \mathbf{B}[j](1 - I_{ij}) + \mathbf{B}[i]I_{ij}$, agent j gives a weight of I_{ij} to the opinion of i and of $(1 - I_{ij})$ to *their own opinion*. Therefore, the weight that j gives to their opinion may change depending on the agent i . Thus, allowing also a fixed weight I_{jj} of agent j to their own opinion may seem somewhat confusing to some readers. Furthermore, for any $\mathbf{B} \in S$ we would have $\mathbf{B} \xrightarrow{(j,j)} \mathbf{B}$ regardless of the value I_{jj} thus making the actual value irrelevant. Notice also we do not require the sum of the influences over an agent to be 1.

2.2 Runs and Consensus

We are interested in properties of opinion systems, such as convergence to consensus and fairness, which are inherent properties of infinite runs of these systems.

► **Definition 4** (e-path, runs and words). *An execution path (e-path) of an OTS $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$, where $G = (A, E, I)$, is an infinite sequence $\pi = \mathbf{B}_0 e_0 \mathbf{B}_1 e_1 \dots$ (also written $\mathbf{B}_0 \xrightarrow{e_0} \mathbf{B}_1 \xrightarrow{e_1} \dots$) such that $\mathbf{B}_t \xrightarrow{e_t} \mathbf{B}_{t+1}$ for each $t \in \mathbb{N}$. We say that e_t is the action performed at time t and that \mathbf{B}_t is the state of opinion at time t . Furthermore, if $\mathbf{B}_0 = \mathbf{B}_{\text{init}}$ then the e-path π is said to be a run of M .*

An ω -word of M is an infinite sequence of edges (i.e., an element of E^ω). The sequence $w_\pi = e_0 e_1 \dots$ is the ω -word generated by π . Conversely, given an ω -word $w = e'_0 e'_1 \dots$ the (unique) run that corresponds to it is $\pi_w = \mathbf{B}_{\text{init}} \xrightarrow{e'_0} \mathbf{B}_1 \xrightarrow{e'_1} \dots$

► **Remark 5.** The uniqueness of the run that corresponds to a given ω -word is derived from the fact that an OTS is a deterministic transition system¹. This gives us a one-to-one correspondence between ω -words and runs, which allows us to abstract away from opinion states when they are irrelevant or clear from the context. In fact, throughout the paper, *we will use the terms ω -words and runs of an OTS interchangeably when no confusion arises*. It is also worth noting that in OTS, any action (edge) can be chosen at any point in an execution path; that is, *all actions are enabled*.

Consensus is a property of central interest in social learning models [21]. Indeed, failure to reach a consensus is often an indicator of polarization in a community.

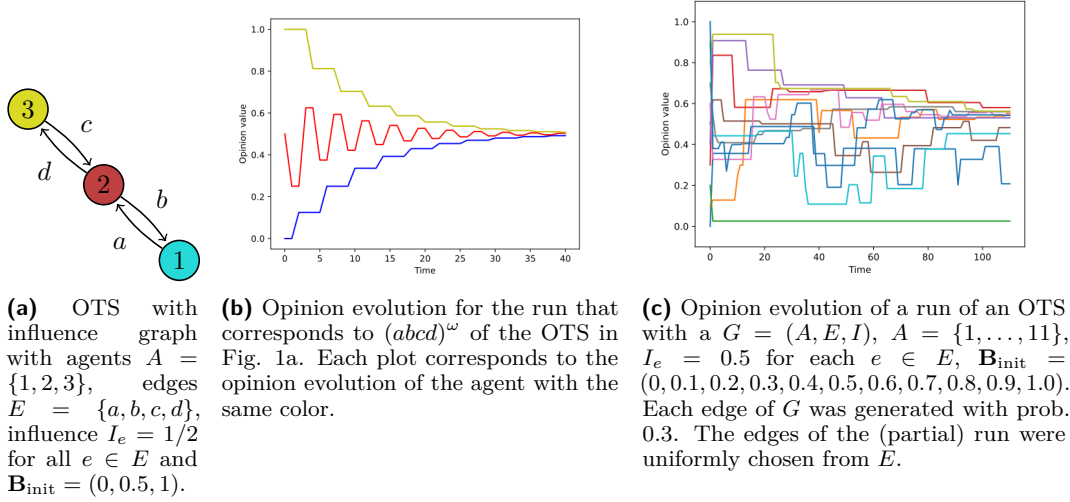
► **Definition 6** (Consensus). *Let $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ be an OTS with $G = (A, E, I)$ and $\pi = \mathbf{B}_{\text{init}} \xrightarrow{e_0} \mathbf{B}_1 \xrightarrow{e_1} \dots$ be a run. We say that an agent $i \in A$ converges to an opinion value $v \in [0, 1]$ in π if $\lim_{t \rightarrow \infty} \mathbf{B}_t[i] = v$. The run π converges to consensus if all the agents in A converge to the same opinion value in π .*

Furthermore, \mathbf{B} is said to be a consensual state if it is a constant vector; i.e., if there exists $v \in [0, 1]$ such that for every $i \in A$, $\mathbf{B}[i] = v$.

► **Example 7.** Let $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ where G is the influence graph in Fig. 1a and $\mathbf{B}_{\text{init}} = (0, 0.5, 1)$. If we perform a on \mathbf{B}_{init} we obtain $\mathbf{B}_{\text{init}} \xrightarrow{a} \mathbf{B}_1 = (0.0, 0.25, 1.0)$.

Consider the word $w = (abcd)^\omega$. Then $\pi_w = \mathbf{B}_{\text{init}} \xrightarrow{a} (0.0, 0.25, 1.0) \xrightarrow{b} (0.125, 0.25, 1.0) \xrightarrow{c} (0.125, 0.625, 1.0) \xrightarrow{d} (0.125, 0.625, 0.8125) \xrightarrow{a} \dots$. Fig. 1b suggests that π_w indeed converges to consensus (to opinion value 0.5). A more complex example of the evolution of opinions from a randomly generated graph with eleven agents is illustrated in Fig. 1c.

¹ While the actions in a run can be seen as being chosen non-deterministically by a scheduler, an OTS is a *deterministic* transition system in the sense that given a state \mathbf{B} and an action e , there exists a unique state \mathbf{B}' such that $\mathbf{B} \xrightarrow{e} \mathbf{B}'$.



■ **Figure 1** Run examples for OTS in Fig. 1a and randomly-generated OTS in Fig. 1c.

The examples above illustrate runs that may or may not converge to consensus. In the next section, we identify conditions on the influence and topology of graphs and on the runs that guarantee this central property of opinion models.

3 Strong Connectivity, Puppet-Freedom and Fairness

In this section, we discuss graph properties, as well as fairness notions and criteria from the literature on concurrent systems that give us insight into how agents converge to consensus in an OTS. For simplicity, we assume an underlying OTS $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ with an influence graph $G = (A, E, I)$. We presuppose basic knowledge of graph theory and formal languages.

3.1 Strong Connectivity

As in the DeGroot model, if there are (groups of) agents in G that do not influence each other (directly or indirectly) and their initial opinions are different, these groups may converge to different opinion values. Consider the example in Fig. 2 where the groups of agents $G_1 = \{1, 2\}$ and $G_2 = \{5, 6\}$ do not have external influence (directly or indirectly), but influence the group $G_3 = \{3, 4\}$. Each group is strongly connected within; their members influence each other. The agents in G_1 converge to an opinion, and so do the agents in G_2 , but to a different one. Hence, the agents in both groups cannot converge to consensus. The agents in G_3 do not even converge to an opinion because they are regularly influenced by the dissenting opinions of G_1 and G_2 .

The above can be prevented by requiring *strong connectivity*, i.e., there must be a path in G from any other to any other. Recall that a *graph path* from i to j of length m in G is a sequence of edges of E of the form $(i, i_1)(i_1, i_2) \dots (i_{m-1}, j)$, where the agents in the sequence are distinct. We shall refer to graph paths as *g-paths* to distinguish them from e-paths in Def. 4. We say that agent i influences agent j if there is a g-path from i to j in G . The graph G is *strongly connected* iff there is a g-path from any agent to any other in G . Hence, in strongly-connected graphs, all agents influence one another.

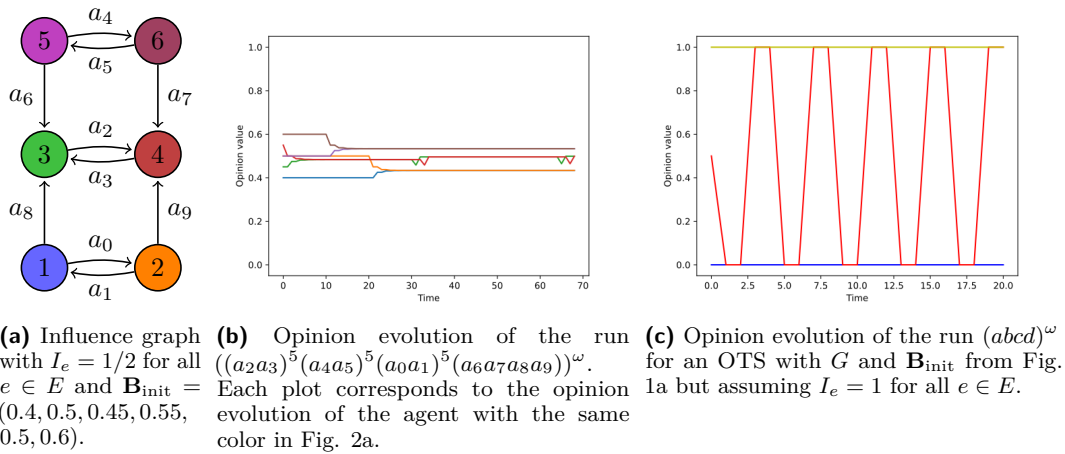


Figure 2 Run examples for OTS in Fig. 1a and Fig. 2a.

3.2 Puppet-Freedom

Nevertheless, too much influence may prevent consensus. If $\mathbf{B} \xrightarrow{(i,j)} \mathbf{B}'$ and $I_{ij} = 1$, agent j behaves as a *puppet* of i forgetting their own opinion and adopting that of j . Fig. 2c illustrates this for the strongly-connected graph in Fig. 1a but with $I_{ij} = 1$ for each $(i, j) \in E$: Agents 1 and 3 use Agent 2 as a puppet, constantly swaying his opinion between 0 and 1. We therefore say that the influence graph G is *puppet free* if for each $(i, j) \in E$, $I_{ij} < 1$.

3.3 Strong Fairness

In an OTS, if G is strongly connected but a given edge is never chosen in a run (or not chosen sufficiently often), it may amount to not having all agents influence each other in that run, hence preventing consensus. For this reason, we make some fairness assumptions about the runs.

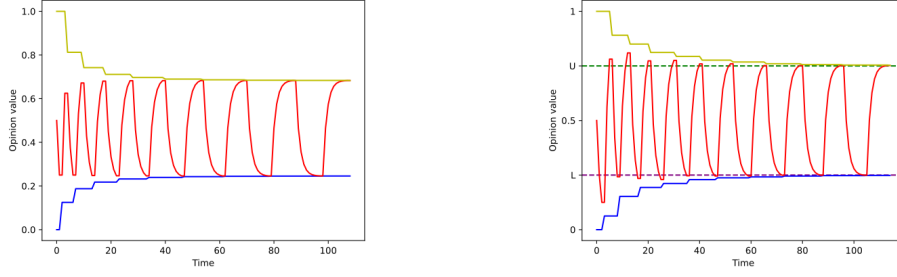
In the realm of transition systems, fairness assumptions rule out some runs, typically those where some actions are not chosen sufficiently often when they are enabled sufficiently often. There are many notions of fairness (see [5, 19, 25] for surveys), but strong fairness is perhaps one of the most representative. As noted above, every action $e \in E$ is always enabled in every run of an OTS. Thus, in our context, strong fairness of a given OTS run (ω -word) amounts to requiring that every action e occurs infinitely often in the run.

► **Definition 8** (Strong fairness). *Let w be an ω -word of an OTS. We say that w is strongly fair if every $e \in E$ occurs in every suffix of w .*

Notice that the graph from Ex. 7 is strongly connected and puppet free, and the ω -word $w = (abcd)^\omega$ is indeed strongly fair and converges to consensus. Nevertheless, puppet freedom, strong fairness, and strong connectivity are not sufficient to guarantee consensus.

► **Proposition 9.** *There exists $(G, \mathbf{B}_{\text{init}}, \rightarrow)$, where G is strongly connected and puppet free, with a strongly-fair run that does not converge to consensus.*

The proof of the existence statement in Prop. 9 is given next.



(a) Opinion evolution of the OTS from Fig. 1a for the ω -word $u = (a^n bc^n d)_{n \in \mathbb{N}^+}$. (b) Opinion evolution of the OTS from Fig. 1a for $U = 0.75$, $L = 0.25$ and the ω -word w from Cons. 10.

■ **Figure 3** Run examples for OTS in Fig. 1a.

► **Construction 10** (Counter-Example to Consensus). Let $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ be an OTS where G is the strongly-connected puppet-free influence graph in Fig. 1a and \mathbf{B}_{init} is any state of opinion such that $\mathbf{B}_{\text{init}}[1] < \mathbf{B}_{\text{init}}[2] < \mathbf{B}_{\text{init}}[3]$. We have $A = \{1, 2, 3\}$ and $E = \{a, b, c, d\}$. We construct an ω -word w such that π_w does not converge to consensus with the following infinite iterative process. Let U and L be such that $\mathbf{B}_{\text{init}}[1] < L < \mathbf{B}_{\text{init}}[2] < U < \mathbf{B}_{\text{init}}[3]$.

Process: (1) Perform a non-empty sequence of a actions with as many a 's as needed until the opinion of Agent 2 becomes smaller than L . (2) Perform the action b . (3) Perform a non-empty sequence of c 's with as many c 's as needed until the opinion of Agent 2 becomes greater than U . (4) Perform the action d . The result of this iteration is a sequence of the form a^+bc^+d . Repeat steps 1–4 indefinitely.

The above process produces the ω -sequence $w = w_1 \cdot w_2 \cdot \dots$ of the form $(a^+bc^+d)^\omega$, where each $w_i = a^{n_i}bc^{m_i}d$ is the result of the i -th iteration of the process and $n_i > 0$ and $m_i > 0$ are the number of a 's and c 's in such interaction. (The evolution of the opinion of run π_w , with $U = 0.75$, $L = 0.25$ and $\mathbf{B}_{\text{init}} = (0, 0.5, 1)$ is illustrated in Fig. 3b)

Since each action $e \in E$ appears infinitely often in w , w is strongly fair. Furthermore, right after each execution of Step 2, the opinion of Agent 1 gets closer to L , but it is still smaller than L since the opinion of Agent 2 at that point is smaller than L . For symmetric reasons, the opinion of Agent 3 gets closer to U , but it is still greater than U . Consequently, the opinion of Agent 1 is always below L , while the opinion of Agent 3 is always above U with $L < U$. Therefore, they cannot converge to the same opinion.

Another ω -word for the OTS in Fig. 1a exhibiting a behavior similar to w in Cons. 10, but whose proof of non-convergence to consensus seems more involved, is $u = (a^n bc^n d)_{n \in \mathbb{N}^+} = u_1 \cdot u_2 \cdot \dots$, where each $u_n = a^n bc^n d$. (see Fig. 3a). The delay in both w and u to execute d after b grows unboundedly due to the growing number of c 's. More precisely, let $\#e(v)$ be the number of occurrences of $e \in E$ in a finite sequence v .

► **Proposition 11.** Let $w = w_1 \cdot w_2 \cdot \dots$ be the ω -word from Cons. 10 where each w_m has the form a^+bc^+d . Then for every $m \in \mathbb{N}$, there exists $t \in \mathbb{N}$ such that $\#c(w_{m+t}) > \#c(w_m)$.

The above proposition states that the number of consecutive c 's in w grows unboundedly, and hence so does the delay for executing d right after executing b . To prevent this form of unbounded delay, we recall in the next section some notions of fairness from the literature that require, at each position of an ω -word, every action to occur within some bounded period of time.

3.4 Bounded Fairness

We start by introducing some notation to give a uniform presentation of some notions of fairness from the literature. We assume $|E| > 1$; otherwise, all the fairness notions are trivial.

A word w is a possibly infinite sequence over E . A *subword* of w is either a suffix of w or a prefix of some suffix of w . Let κ be an ordinal from the set $\omega + 1 = \mathbb{N} \cup \{\omega\}$ where ω denotes the first infinite ordinal. A κ -word is a word of length κ . Recall that each ordinal can be represented as the set of all strictly smaller ordinals. We can then view a κ -word $w = (e_i)_{i \in \kappa}$ as a function $w : \kappa \rightarrow E$ such that $w(i) = e_i$ for each $i \in \kappa$. A κ -word w is *complete* if $w(\kappa) = E$ (where $w(\kappa)$ denotes the image of the function w). A κ -window u of w is a *subword* of w of length κ . Thus, if $\kappa = \omega$ then u is a suffix of w , and if $\kappa \in \mathbb{N}$, u can be thought of as a *finite observation* of κ consecutive edges in w . We can now introduce a general notion of fairness parametric in κ .

► **Definition 12** (κ -fairness, bounded-fairness). *Let w be an ω -word over E and $\kappa \in \omega + 1$: w is κ -fair if every κ -window of w is complete. Furthermore, w is bounded fair if it is k -fair for some $k \in \mathbb{N}$.*

Notice that the notion of strong fairness in Def. 8 is obtained by taking $\kappa = \omega$; indeed, w is ω -fair iff every $e \in E$ occurs infinitely often in w . Furthermore, if $\kappa = k$ for some $k \in \mathbb{N}^+$, then we obtain the notion of k -fairness from [16]². Intuitively, if w is k -fair, then at any position of w , every $e \in E$ will occur within a window of length k from that position.

It is not difficult to see that ω -fairness is strictly weaker than bounded-fairness, which in turn is strictly weaker than any k -fairness with $k \in \mathbb{N}$. Let $F(\kappa)$ be the set of all ω -words over E that are κ -fair. We have the following sequence of strict inclusions.

► **Proposition 13.** *For every $k \in \mathbb{N}$, $F(k) \subset F(k+1) \subset (\bigcup_{\kappa \in \mathbb{N}} F(\kappa)) \subset F(\omega)$.*

► **Example 14.** Let us consider the *fair word* w from Cons. 10, the counter-example to consensus. From Prop. 11, the delay for executing action d immediately after executing action b increases without bound. Thus, for every k , there must be a non-complete k -window u of w such that d does not occur in u . Consequently, w is not bounded fair.

Not only does bounded fairness rule out the counter-example in Cons. 10, but it also guarantees consensus, as shown later, for runs of OTS with strongly-connected, puppet-free influence graphs. Nevertheless, it may be too strong of a requirement for consensus. We, therefore, introduce a weaker notion that satisfies the following criteria and guarantees consensus.

Some Fairness Criteria

Let us briefly discuss some fairness criteria and desirable properties that justify our quest for a weaker notion of fairness that guarantees consensus. An in-depth discussion about criteria for fairness notions, from which we drew some inspiration, can be found in [34, 33, 19, 5].

Machine Closure. Following [1, 26] one of the most important criteria that a notion of fairness must meet is *machine closure* (also called *feasibility* [5]). Fairness properties are properties of infinite runs; hence, a natural requirement is that any finite partial run must have the chance to be extended to a fair run. Thus, we say that a notion of fairness is *machine closed* if every finite word u can be extended to a fair ω -word $u \cdot w$.

² This notion is different from the notion of k -fairness from [9]

Clearly, k -fairness with $k \in \mathbb{N}$ is not machine closed; e.g., the word $c^k d$ with $E = \{c, d\}$ cannot be extended to a k -fair ω -word. Nevertheless, bounded fairness is machine closed: Each k -word u can be extended to a $(k + m)$ -fair word $u \cdot (e_1 \dots e_m)^\omega$ assuming $E = \{e_1, \dots, e_m\}$.

Constructive Liveness. According to [34], a notion of fairness may require that a particular action is taken sufficiently often, but it should not prevent any other actions from being taken sufficiently often. This concept is formalized in [34, 33] in a game-theoretical scenario, reminiscent of a Banach–Mazur game [28], involving an infinite interaction between a scheduler and an opponent. The opponent initiates with a word w_0 , then the scheduler appends a finite word w_1 to w_0 . This pattern continues indefinitely, resulting in an ω -word $w = w_0 \cdot w_1 \cdot w_2 \dots$. A given fairness notion is said to be a *constructive liveness* property if, regardless of what the opponent does, the scheduler can guarantee that the resulting ω -word is fair under the given notion.

The notion of *bounded fairness is not a constructive liveness property*. If an ω -word is bounded fair, it is k -fair for some $k \geq |E| > 1$. Let $c \in E$ and take as the strategy of the opponent to choose in each of their turns $w_n = c^n$. Since $|E| > 1$, then w_{2k} cannot be a complete k -window. Therefore, the resulting $w = w_0 \cdot w_1 \cdot w_2 \dots$ is not bounded fair, regardless of the strategy of the scheduler.

It is worth noticing that the above opponent’s strategy is reminiscent of our procedure to construct an ω -sequence in Cons. 10 using the unbounded growth of c ’s to prevent consensus.

Random Words. Consider a word $e_0 e_1 \dots$ where each edge or action $e_n = (i, j)$ is chosen from E *independently* with probability $p_{(i,j)} > 0$. Let us refer to such kinds of sequences as *random* words. We then say that a given notion of fairness is *random inclusive* if every random ω -word is *almost surely* (i.e., with probability one) fair under the given notion.

It follows from the Second Borel–Cantelli lemma³ that every random word is *almost surely* strongly fair. Nevertheless, the notion of bounded fairness fails to be random inclusive: If a word is bounded fair, it is k -fair for some $k \geq |E|$, and thus it needs to have the form $w_0 \cdot w_1 \dots$ where each w_m is a complete k -window. Since $1 < |E|$, the probability that a random k window is complete is strictly smaller than 1. Therefore, the probability of a random word having an infinite number of *consecutive* complete k -windows is 0.

Random words are important in simulations of our model (see Fig. 1c). Furthermore, having a notion of fairness that is random inclusive and guarantees consensus will allow us to derive and generalize consensus results for randomized opinion models, such as gossip algorithms [17]. We elaborate on this in the related work. We now introduce our new notion of fairness.

4 A New Notion of Bounded Fairness

A natural way to relax bounded fairness to satisfy constructive liveness and random inclusion is to require that the complete k -windows need only appear infinitely often: i.e., an ω word w is said to be *weakly bounded* fair if there exists $k \in \mathbb{N}$ such that every suffix of w has a k -window. Nevertheless, as it will be derived later, weak bounded fairness is not sufficient to guarantee consensus.

³ The lemma states that if the sum of the probabilities of an infinite sequence of events $E_0 E_1 \dots$ that are independent is infinite, then the probability of infinitely many of those events occurring is 1 [31]. Here, each event E_k expresses that the edge e_k occurs at time k and these events are independent because each edge (i, j) in a random word is chosen independently with probability $p_{(i,j)} > 0$.

It turns out that, to guarantee consensus, it suffices to require that a large enough number m of *consecutive* complete k -windows appear infinitely often. These consecutive windows are referred to as multi-windows.

► **Definition 15** ((m, κ) multi-window). *Let w be an ω -word over E , $m \in \mathbb{N}^+$ and $\kappa \in \omega + 1$. We say that w has an (m, κ) multi-window if there exists a subword u of w of the form $u = w_1 \cdot w_2 \cdot \dots \cdot w_m$ where each w_i is a κ -window of w . Furthermore, if each w_n in u is complete, we say that w has a complete (m, κ) multi-window. If it exists, the word u is called an (m, κ) multi-window of w .*

Notice that because of the concatenation of windows in Def. 15, by construction, no ω -word has a (m, ω) multi-window with $m > 1$: If $\kappa = \omega$ then $m = 1$. In this case, the multi-window is just a window of infinite length of w , i.e., a suffix of w .

► **Definition 16** ((m, κ) -fairness). *Let w be an ω -word over E , $m \in \mathbb{N}^+$ and $\kappa \in \omega + 1$. We say that w is (m, κ) -fair if every suffix of w has a complete (m, κ) multi-window. We say that w is m -consecutive bounded fair, or m -bounded fair, if it is (m, k) -fair for some $k \in \mathbb{N}$.*

Clearly, w is ω -fair iff it is $(1, \omega)$ -fair, and w is weakly bounded fair iff it is 1-bounded ω -fair. Let $F(m, \kappa)$ and $F(\kappa)$ be the sets of ω -words that are (m, κ) -fair and κ -fair, respectively. We have the following sequence of strict inclusions (assume $k, m \in \mathbb{N}^+$):

► **Proposition 17.** $F(k) \subset F(m+1, k) \subset F(m, k) \subset (\bigcup_{\kappa \in \mathbb{N}} F(m, \kappa)) \subset F(1, \omega) = F(\omega)$.

Compliance with Fairness Criteria. Let us consider the criteria for fairness in the previous section. The notion of m -bounded fairness is machine closed since bounded fairness is stronger than m -bounded fairness (Prop. 13 and Prop. 17) and bounded fairness is machine closed.

It is also a constructive liveness property since (m, k) fairness, for $k \geq |E|$, is stronger than m -bounded fairness (Prop. 17), and it is also a constructive liveness property: A winning strategy for the scheduler is to choose a complete (m, k) -window at each one of its turns.

Similarly, m -Bounded Fairness is random inclusive since the stronger notion (m, k) -Fairness is random inclusive for $k \geq |E|$. In a random ω -word $w = w_0 \cdot w_1 \dots$ where each w_n is a $(m \times k)$ -window, the probability that w_n is a complete (m, k) -multi-window is non-zero and independent. Thus again, by the Second Borel–Cantelli lemma, almost-surely w has infinitely many complete (m, k) multi-windows, i.e., it is almost-surely (m, k) -fair.

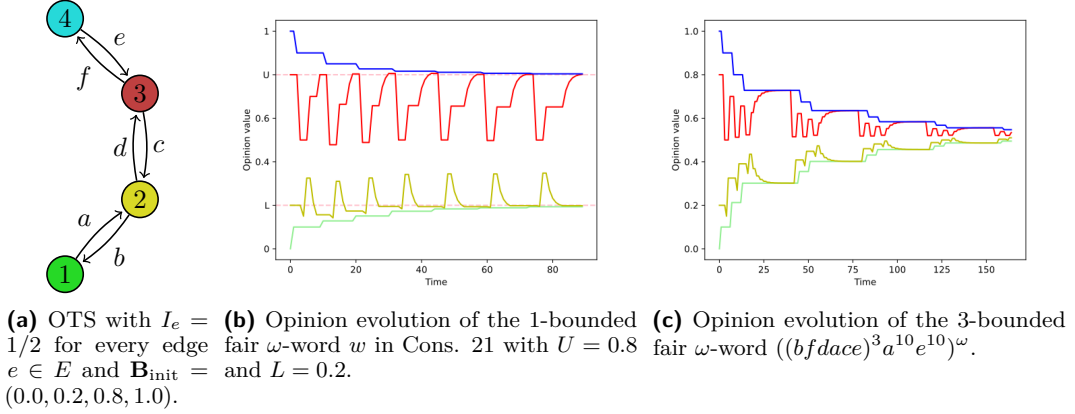
4.1 Consensus Theorem

We can now state one of our main theorems: m -bounded fairness guarantees consensus in strongly-connected, puppet-free graphs.

► **Theorem 18** (Consensus under m -bounded fairness). *Let $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ be an OTS where G is a strongly-connected, puppet-free influence graph. For every run π of M , if w_π is m -bounded fair and $m \geq |A| - 1$, then π converges to consensus.*

► **Remark 19.** A noteworthy corollary of Th. 18 is that, under the same assumptions of the theorem, if w_π is a bounded fair (a random ω -word), then π converges to consensus (π almost surely converges to consensus). This follows from the above theorem, Prop. 13, Prop. 17 and the fact that m -bounded fairness is random inclusive.

A proof of Th. 18 is given in the technical report [7]. Let us give the main intuitions here.



■ **Figure 4** Examples of an m -bounded fair runs. In Fig. 4b and 4c, each plot corresponds to the opinion of the agent with the same color in Fig. 4a.

Proof sketch. The proof focuses on the evolution of maximum and minimum opinion values. The sequences of maximum and minimum opinion values in a run, $\{\max \mathbf{B}_t\}_{t \in \mathbb{N}}$ and $\{\min \mathbf{B}_t\}_{t \in \mathbb{N}}$, can be shown to be (bounded) monotonically non-increasing and non-decreasing, respectively, so they must converge to some opinion values, say U and L with $L \leq U$.

We must then argue that $L = U$ (this implies convergence to consensus of π by the Squeeze Theorem [32]). Since w_π is m -bounded fair with $m \geq |A| - 1$, after performing all the actions of an (m, k) multi-window of w_π , for some $k \geq |E|$, all the agents of A would have influenced each other. In particular, the agents holding the maximum and minimum opinion values, say agents i and j . To see this, notice that since G is strongly connected, there is a path from i to j , $a_1 \dots a_l$ with length $l \leq |A| - 1$. Thus, after performing the first complete k -window of the (m, k) -multi-window, a_1 must be performed, after performing the second complete k -window, a_2 must be performed and so on. Hence, after performing all the actions of the multi-window, i would have influenced j . It can be shown that their mutual influence causes them to decrease their distance by a positive constant factor (here, the puppet freedom assumption is needed). Since the w_π is m -fair, there are infinitely many (m, k) -windows to be performed, and thus the sequences of maximum and minimum opinion values converge to each other, i.e., $U = L$. ◀

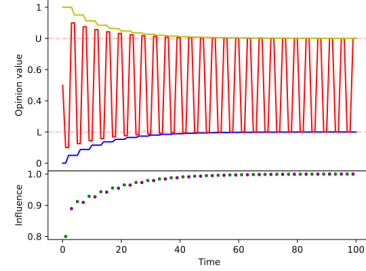
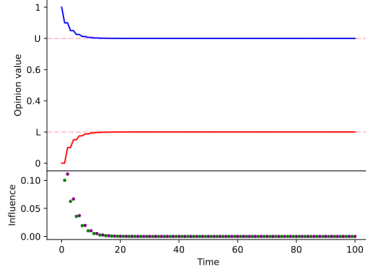
It is worth pointing out that without the condition $m \geq |A| - 1$ in Th. 18, we cannot guarantee consensus. Fig. 4c illustrates an m -bounded fair run, for $m = |A| - 1$, of an OTS with 4 agents that converges to consensus. Nevertheless, the following run construction shows that for $m = |A| - 3$, we can construct an m -bounded fair run that fails to converge to consensus (the run is illustrated in Fig. 4b). It also shows that weak bounded fairness, i.e., 1-bounded fairness, is not sufficient to guarantee convergence to consensus. We do not have a counter-example or a proof for $m = |A| - 2$.

► **Proposition 20.** *There exists $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$, where $G = (A, E, I)$ is a strongly connected, puppet-free graph, with an m -bounded fair ω -word w , $m = |A| - 3$, such that π_w does not converge to consensus.*

The proof of the above proposition is given in the following construction.



(a) $\mathbf{B}_{\text{init}} = (0.0, 1.0)$ and if $\mathbf{B}[1] = \mathbf{B}[2]$ then $I_a^{\mathbf{B}} = I_b^{\mathbf{B}} = 0.5$, otherwise $I_a^{\mathbf{B}} = \left[\frac{U - \mathbf{B}[2]}{2(\mathbf{B}[1] - \mathbf{B}[2])} \right]_0^1$, $I_b^{\mathbf{B}} = \left[\frac{L - \mathbf{B}[1]}{2(\mathbf{B}[2] - \mathbf{B}[1])} \right]_0^1$.
 (b) $\mathbf{B}_{\text{init}} = (0.0, 0.5, 1.0)$, $I_d^{\mathbf{B}} = I_b^{\mathbf{B}} = 0.5$, if $\mathbf{B}[1] = \mathbf{B}[2]$ then $I_a^{\mathbf{B}} = 0.5$, if $\mathbf{B}[2] = \mathbf{B}[3]$ then $I_c^{\mathbf{B}} = 0.5$, otherwise $I_a^{\mathbf{B}} = \left[\frac{\frac{1}{2}(\mathbf{B}[1] + L) - \mathbf{B}[2]}{\mathbf{B}[1] - \mathbf{B}[2]} \right]_0^1$, $I_c^{\mathbf{B}} = \left[\frac{\frac{1}{2}(\mathbf{B}[3] + U) - \mathbf{B}[2]}{\mathbf{B}[3] - \mathbf{B}[2]} \right]_0^1$.



(c) Opinion and influence evolution of the ω -word $(ab)^\omega$. Each plot corresponds to the opinion of the agent with the same color in Fig. 5a. The influences $I_a^{\mathbf{B}}$ and $I_b^{\mathbf{B}}$ are plotted in green and purple.
 (d) Opinion and influence evolution of the ω -word $(abcd)^\omega$. Each plot corresponds to the opinion of the agent with the same color in Fig. 5b. The influences $I_a^{\mathbf{B}}$ and $I_c^{\mathbf{B}}$ are plotted in green and purple.

■ **Figure 5** Plots for DOTS in Fig. 5a and Fig. 5b with $U = 0.8$ and $L = 0.2$.⁴

► **Construction 21** (Counter-Example to Consensus for m -bounded fairness with $m \leq |A| - 3$).
 Suppose that $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ where G is the strongly-connected, puppet-free, influence graph in Fig. 4a and \mathbf{B}_{init} is any state of opinion such that $\mathbf{B}_{\text{init}}[1] < \mathbf{B}_{\text{init}}[2] < \mathbf{B}_{\text{init}}[3] < \mathbf{B}_{\text{init}}[4]$. We have $A = \{1, 2, 3, 4\}$ and $E = \{a, b, c, d, e, f\}$. We construct an ω -word w such that π_w does not converge to consensus with the following infinite iterative process. Let U and L be such that $\mathbf{B}_{\text{init}}[2] \leq L < U \leq \mathbf{B}_{\text{init}}[3]$.

Process: (1) Perform the sequence of actions $bfdace$. (2) Perform a sequence of a 's until the opinion of Agent 2 becomes smaller than L . (3) Perform a sequence of e 's until the opinion of Agent 3 becomes greater than U . The result of this iteration is a sequence of the form $bfdace \cdot a^* e^*$. Repeat steps 1-3 indefinitely.

The above process produces the ω -sequence $w = v \cdot w_1 \cdot v \cdot w_2 \cdot \dots$ of the form $(bfdace a^* e^*)^\omega$ where $v = bfdace$ and $w_i = a^{m_i} e^{n_i}$ are results of the i -th iteration of the process, and $n_i \geq 0$ and $m_i \geq 0$ are the number of a 's and e 's in each w_i . (The opinion evolution of run π_w , with $L = 0.2$, $U = 0.8$ and $\mathbf{B}_{\text{init}} = (0.0, 0.2, 0.8, 1.0)$ is illustrated Fig. 4b)

Since the subword v is a complete $(1,6)$ -multi-window and appears infinitely often in w , w is m -bounded fair for $m = |A| - 3 = 1$. Furthermore, right after each execution of edge f in step 1, the opinion of Agent 1 gets closer to L , but it is still smaller than L since the opinion of Agent 2 at that point is smaller than L . For symmetric reasons, after action b , the opinion of Agent 4 gets closer to U , but it is still greater than U since the opinion of Agent 3 at that point is greater than U . Consequently, the opinion of Agent 1 is always below L , while the opinion of Agent 4 is always above U with $L < U$. Therefore, they cannot converge to the same opinion.

5 Dynamic Influence

The static weights of the influence graph of an OTS imply that the influence that each individual has on others remains constant throughout opinion evolution. However, in real-life scenarios, the influence of individuals can vary depending on many factors, in particular the state of opinion (or opinion climate). Indeed, individuals may gain or lose influence based on the current opinion trend or for expressing dissenting and extreme opinions, among others.

To account for the above form of dynamic influence, we extend the weight function $I : E \rightarrow (0, 1]$ of the influence graph $G = (A, E, I)$ as a function $I : E \times [0, 1]^{|A|} \rightarrow [0, 1]$ on edges and the state of opinion. The resulting graph is said to have *dynamic influence*.

► **Definition 22** (Dynamic OTS). *A Dynamic OTS (DOTS) is a tuple $(G, \mathbf{B}_{\text{init}}, \rightarrow)$ where $G = (A, E, I)$ has dynamic influence $I : E \times [0, 1]^{|A|} \rightarrow [0, 1]$. We write $I_{ij}^{\mathbf{B}}$ for $I((i, j), \mathbf{B})$. The labeled transition \rightarrow is defined as in Def. 2 but replacing I_{ij} with $I_{ij}^{\mathbf{B}}$ in Eq. 1.*

The notions of runs, words, e-paths, and related notions for DOTS remain the same as those for OTS (Def. 4). Let us consider some examples of dynamic influence.

Confirmation Bias. Under *confirmation bias* [8], an agent j is more influenced by those whose opinion is closer to theirs. The function $I_{ij}^{\mathbf{B}} = 1 - |\mathbf{B}[j] - \mathbf{B}[i]|$ captures a form of confirmation bias; the closer the opinions of i and j , the stronger the influence of i over j .

Bounded Influence. Nevertheless, if we allow dynamic influence that can converge to 0 in a given run $\mathbf{B}_{\text{init}} \xrightarrow{e_0} \mathbf{B}_1 \xrightarrow{e_1} \dots$, i.e, if $\lim_{t \rightarrow \infty} I_{i,j}^{\mathbf{B}^t} = 0$, we may reduce indefinitely influence and end up in a situation similar to non-strong connectivity of the graph, thus preventing consensus as in Section 3.1 (Fig. 2). Analogously, if $\lim_{t \rightarrow \infty} I_{i,j}^{\mathbf{B}^t} = 1$, we may end up in puppet situations preventing consensus like in Section 3.2 (Fig. 2c). Both situations are illustrated in the DOTS in Fig. 5. To prevent them, we bound the dynamic influences.

► **Definition 23** (Bounded Influence). *A DOTS $(G, \mathbf{B}_{\text{init}}, \rightarrow)$ with $G = (A, E, I)$ has bounded influence if there are constants $I_L, I_U \in (0, 1)$ such that for each $\mathbf{B} \in [0, 1]^{|A|}$, $(i, j) \in E$, we have $I_{i,j}^{\mathbf{B}} \in [I_L, I_U]$.*

The previous form of confirmation bias influence $I_{ij}^{\mathbf{B}} = 1 - |\mathbf{B}[j] - \mathbf{B}[i]|$ is not bounded. Nevertheless, the linear transformation $I_L + (I_U - I_L)I_{ij}^{\mathbf{B}}$ can be used to scale any unbounded influence $I_{ij}^{\mathbf{B}}$ into a bounded one in $[I_L, I_U]$ while preserving its shape.

We conclude with our other main theorem, whose proof is given in the technical report [7].

► **Theorem 24** (Consensus with bounded influence). *Let $M = (G, \mathbf{B}_{\text{init}}, \rightarrow)$ be a DOTS where G is a strongly-connected, influence graph. Suppose that M has bounded influence. For every run π of M , if w_π is m -bounded fair with $m \geq |A| - 1$, then π converges to consensus.*

The result generalizes Th. 18 to dynamic bounded influence. Therefore, in strongly-connected and dynamic bounded influence graphs, convergence to consensus is guaranteed for all runs that are m -bounded fair, which include each random run almost surely.

⁴ We use a clamp function for $[0, 1]$ defined as $[r]_0^1 = \min(\max(r, 0), 1)$ for every $r \in \mathbb{R}$.

6 Conclusions and Related Work

We introduced a DeGroot-based model with asynchronous opinion updates and dynamic influence using labelled transition systems. The model captures opinion dynamics in social networks more faithfully than the original DeGroot model. The fairness notions studied and the consensus results in this paper show that the model is also tractable and brings new insights into opinion formation in social networks. To our knowledge, this is the first work that uses fairness notions from concurrent systems in the context of DeGroot-based models.

There is a great deal of work on DeGroot-based models for social learning (e.g., [4, 13, 12, 38, 37, 15, 11]). We discuss work with asynchronous updates and dynamic influence, which is the focus of this paper. The work [15] introduces a version of the DeGroot model in which self-influence changes over time, while the influence on others remains the same. The works [11, 12] explore convergence and stability, respectively, in models where influences change over time. The works mentioned above do not take into account asynchronous communication, whereas this paper demonstrates how asynchronous communication, when combined with dynamic influence, can prevent consensus.

Recent works on gossip algorithms [17, 30, 2, 35] study consensus with asynchronous communications for distributed averaging and opinion dynamics. The work in [30] studies *reaching* consensus (in finite time) rather than *converging* to consensus. The works [2, 35] consider undirected cliques rather than directed graphs as influence graphs. The closest work is [17], which states consensus for random runs in directed strongly connected graphs but unlike our case all edges have the same fixed weight $q \in (0, 1)$ (i.e., they assume static influence with the same influence value for all edges). The dynamics of asymmetric gossip updates in [17] can indeed be captured as OTS, and their random runs are almost-surely m -bounded fair. Consequently, our work generalizes the consensus result in [17] by extending it to graphs with (bounded) dynamic influence and whose edges may have different weights. Furthermore, the framework in [17] does not address fairness notions which are the focus and the main novelty of our work.

The work [19] discusses probabilistic fairness as a method equally strong as strong fairness to prove liveness properties, where a liveness property is characterized by a set of states such that a run holds this property iff the run reaches a state of this set. However, the property of (*convergence to*) *consensus* (Def. 6) does not correspond to this notion of liveness since it is not about *reaching* a specific set of states but about *converging* to a consensual state. In fact, unless there are puppets or the initial state of a run is already a consensual state, consensus is never reached in finite time in our model.

Bounded fair ω -words can be characterized by Prompt Buchi Automata (PBW) [3]. Indeed, the set of bounded-fair words of an OTS can be characterized as the language of PBW. Hence, the closure properties of these automata may prove valuable for future developments of our work. It would also be interesting to see in future work whether or not the m -bounded fair words of an OTS can be characterized as the language of a PBW (or of an elegant variant of it).

In future work, we plan to study the actual value of consensus in a given system. This may provide information about the most influential agents. We also plan to study how actions can be scheduled (or manipulated), while preserving the fairness assumptions, to converge more quickly or slowly to a consensus, or to a given consensus value. For example, giving priority to edges whose agents have a greater opinion disagreement, while respecting fairness assumptions. We may build on previous work on priorities in concurrent communications [6] for this purpose.


Finally, we plan to extend our model with agents that can learn by exchanging beliefs, lies, and information, by building upon our work in concurrent constraint programming (e.g. [24, 23, 18]).

References

- 1 Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. In *Real-Time: Theory in Practice*. Springer Berlin Heidelberg, 1992.
- 2 Emerico Aguilar and Yasumasa Fujisaki. Opinion dynamics via a gossip algorithm with asynchronous group interactions. *Proceedings of the ISCTE International Symposium on Stochastic Systems Theory and its Applications*, 2019:99–102, 2019. doi:10.5687/sss.2019.99.
- 3 Shaull Almagor, Yoram Hirshfeld, and Orna Kupferman. Promptness in ω -regular automata. In *Automated Technology for Verification and Analysis*. Springer Berlin Heidelberg, 2010.
- 4 Mário S. Alvim, Bernardo Amorim, Sophia Knight, Santiago Quintero, and Frank Valencia. A Multi-agent Model for Polarization Under Confirmation Bias in Social Networks. In *41th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*, 2021. URL: <https://inria.hal.science/hal-03740263>.
- 5 Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. In *14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL 1987*, 1987. doi:10.1145/41625.41642.
- 6 Jesús Aranda, Frank D. Valencia, and Cristian Versari. On the expressive power of restriction and priorities in CCS with replication. In *Foundations of Software Science and Computational Structures, FoSSaCS 2009*, volume 5504 of *Lecture Notes in Computer Science*, 2009. doi:10.1007/978-3-642-00596-1_18.
- 7 Jesús Aranda, Sebastián Betancourt, Juan Fco. Díaz, and Frank Valencia. Fairness and consensus in opinion models (technical report), 2024. arXiv:2312.12251.
- 8 Elliot Aronson, Timothy Wilson, and Robin Akert. *Social Psychology*. Upper Saddle River, NJ : Prentice Hall, 7 edition, 2010.
- 9 Eike Best. Fairness and conspiracies. *Information Processing Letters*, 18(4):215–220, 1984. doi:10.1016/0020-0190(84)90114-5.
- 10 Arun G Chandrasekhar, Horacio Larreguy, and Juan Pablo Xandri. Testing models of social learning on networks: Evidence from a lab experiment in the field. Working Paper 21468, National Bureau of Economic Research, August 2015. doi:10.3386/w21468.
- 11 S. Chatterjee and E. Seneta. Towards consensus: Some convergence theorems on repeated averaging. *Journal of Applied Probability*, 14(1):89–97, 1977. doi:10.2307/3213262.
- 12 Zihan Chen, Jiahui Qin, Bo Li, Hongsheng Qi, Peter Buchhorn, and Guodong Shi. Dynamics of opinions with social biases. *Automatica*, 106:374–383, 2019. doi:10.1016/j.automatica.2019.04.035.
- 13 Pranav Dandekar, Ashish Goel, and David Lee. Biased assimilation, homophily and the dynamics of polarization. *Proceedings of the National Academy of Sciences of the United States of America*, 110, March 2013. doi:10.1073/pnas.1217220110.
- 14 Morris H. DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 1974. URL: <http://www.jstor.org/stable/2285509>.
- 15 Peter M. DeMarzo et al. Persuasion bias, social influence, and unidimensional opinions. *The Quarterly Journal of Economics*, 118(3):909–968, 2003. URL: <http://www.jstor.org/stable/25053927>.
- 16 Nachum Dershowitz, D. N. Jayasimha, and Seungjoon Park. *Bounded Fairness*, pages 304–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. doi:10.1007/978-3-540-39910-0_14.
- 17 F. Fagnani and S. Zampieri. Asymmetric randomized gossip algorithms for consensus. *IFAC Proceedings Volumes*, 2008. doi:10.3182/20080706-5-KR-1001.01528.

- 18 Moreno Falaschi, Carlos Olarte, Catuscia Palamidessi, and Frank D. Valencia. Declarative diagnosis of temporal concurrent constraint programs. In *Logic Programming. ICLP 2007*, 2007. doi:10.1007/978-3-540-74610-2_19.
- 19 Rob Van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Computing Surveys*, 52(4):1–38, August 2019. doi:10.1145/3329125.
- 20 Elizabeth B. Goldsmith. *Introduction to Social Influence: Why It Matters*, pages 3–22. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-20738-4_1.
- 21 Benjamin Golub and Evan Sadler. Learning in social networks. Available at SSRN 2919146, 2017.
- 22 Orna Grumberg, Nissim Francez, Johann A. Makowsky, and Willem P. de Roever. A proof rule for fair termination of guarded commands. *Information and Control*, 66(1):83–102, 1985. doi:10.1016/S0019-9958(85)80014-0.
- 23 Michell Guzmán, Stefan Haar, Salim Perchy, Camilo Rueda, and Frank Valencia. Belief, Knowledge, Lies and Other Utterances in an Algebra for Space and Extrusion. *Journal of Logical and Algebraic Methods in Programming*, September 2016. doi:10.1016/j.jlamp.2016.09.001.
- 24 Stefan Haar, Salim Perchy, Camilo Rueda, and Frank Valencia. An Algebraic View of Space/Belief and Extrusion/Utterance for Concurrency/Epistemic Logic. In *17th International Symposium on Principles and Practice of Declarative Programming (PPDP 2015)*, 2015. doi:10.1145/2790449.2790520.
- 25 M.Z. Kwiatkowska. Survey of fairness notions. *Information and Software Technology*, 31(7):371–386, 1989. doi:10.1016/0950-5849(89)90159-6.
- 26 Leslie Lamport. Fairness and hyperfairness. *Distributed Computing*, 13(4):239–245, November 2000. doi:10.1007/PL00008921.
- 27 D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming*, pages 264–277, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.
- 28 R.D. Mauldin. *The Scottish Book: Mathematics from the Scottish Café*. Birkhäuser, 1981. URL: <https://books.google.com.co/books?id=gaqEAAAIAAJ>.
- 29 Hossein Noorazar. Recent advances in opinion propagation dynamics: a 2020 survey. *The European Physical Journal Plus*, 135(6):521, June 2020. doi:10.1140/epjp/s13360-020-00541-2.
- 30 Guodong Shi, Bo Li, Mikael Johansson, and Karl Henrik Johansson. Finite-time convergent gossiping. *IEEE/ACM Transactions on Networking*, 24(5):2782–2794, 2016. doi:10.1109/TNET.2015.2484345.
- 31 Albert N. Shiryaev. *Probability-1: Volume 1*. Springer New York, 2016. doi:10.1007/978-0-387-72206-1.
- 32 Houshang H. Sohrab. *Basic Real Analysis*. Birkhauser Basel, 2nd edition, 2014. doi:10.1007/0-8176-4441-5.
- 33 Hagen Völzer and Daniele Varacca. Defining fairness in reactive and concurrent systems. *J. ACM*, 59(3), June 2012. doi:10.1145/2220357.2220360.
- 34 Hagen Völzer, Daniele Varacca, and Ekkart Kindler. Defining fairness. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 – Concurrency Theory*, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 35 Xing Wang, Bingjue Jiang, and Bo Li. Opinion dynamics on social networks. *Acta Mathematica Scientia*, 42(6):2459–2477, November 2022. doi:10.1007/s10473-022-0616-8.
- 36 Stanley Wasserman and Katherine Faust. Social network analysis in the social and behavioral sciences. In *Social Network Analysis: Methods and Applications*, pages 1–27. Cambridge University Press, 1994.
- 37 Chen X, Tsaparas P, Lijffijt J, and De Bie T. Opinion dynamics with backfire effect and biased assimilation. *PLoS ONE*, 16(9), 2021. doi:10.1371/journal.pone.0256922.
- 38 Weiguo Xia, Mengbin Ye, Ji Liu, Ming Cao, and Xi-Ming Sun. Analysis of a nonlinear opinion dynamics model with biased assimilation. *Automatica*, 120:109113, 2020. doi:10.1016/j.automatica.2020.109113.

Bidding Games with Charging

Guy Avni   

University of Haifa, Israel

Ehsan Kafshdar Goharshady   

Institute of Science and Technology Austria (ISTA), Austria

Thomas A. Henzinger   

Institute of Science and Technology Austria (ISTA), Austria

Kaushik Mallik   

Institute of Science and Technology Austria (ISTA), Austria

Abstract

Graph games lie at the algorithmic core of many automated design problems in computer science. These are games usually played between two players on a given graph, where the players keep moving a token along the edges according to pre-determined rules (turn-based, concurrent, etc.), and the winner is decided based on the infinite path (aka *play*) traversed by the token from a given initial position. In bidding games, the players initially get some monetary budgets which they need to use to bid for the privilege of moving the token at each step. Each round of bidding affects the players' available budgets, which is the only form of update that the budgets experience. We introduce bidding games *with charging* where the players can additionally improve their budgets during the game by collecting vertex-dependent monetary rewards, aka the “charges.” Unlike traditional bidding games (where all charges are zero), bidding games with charging allow non-trivial recurrent behaviors. For example, a reachability objective may require multiple detours to vertices with high charges to earn additional budget. We show that, nonetheless, the central property of traditional bidding games generalizes to bidding games with charging: For each vertex there exists a *threshold* ratio, which is the necessary and sufficient fraction of the total budget for winning the game from that vertex. While the thresholds of traditional bidding games correspond to unique fixed points of linear systems of equations, in games with charging, these fixed points are no longer unique. This significantly complicates the proof of existence and the algorithmic computation of thresholds for infinite-duration objectives. We also provide the lower complexity bounds for computing thresholds for Rabin and Streett objectives, which are the first known lower bounds in any form of bidding games (with or without charging), and we solve the following *repair problem* for safety and reachability games that have unsatisfiable objectives: Can we distribute a given amount of charge to the players in a way such that the objective can be satisfied?

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Bidding games on graphs, ω -regular objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.8

Related Version *Full Version*: <https://arxiv.org/abs/2407.06288> [13]

Funding This work was supported in part by the ERC projects ERC-2020-AdG 101020093 and CoG 863818 (ForM-SMArt) and by ISF grant no. 1679/21.

1 Introduction

Two-player *graph games* have deep connections to foundations of mathematical logic [26], and constitute a fundamental model of computations with applications in *reactive synthesis* [25] and multi-agent systems [2]. A graph game is played on a graph, called the *arena*, as follows. A token is placed on an initial vertex and the two players move the token throughout the arena to produce an infinite path, called a *play*. The winner is determined based on whether



© Guy Avni, Ehsan Kafshdar Goharshady, Thomas A. Henzinger, and Kaushik Mallik; licensed under Creative Commons License CC-BY 4.0

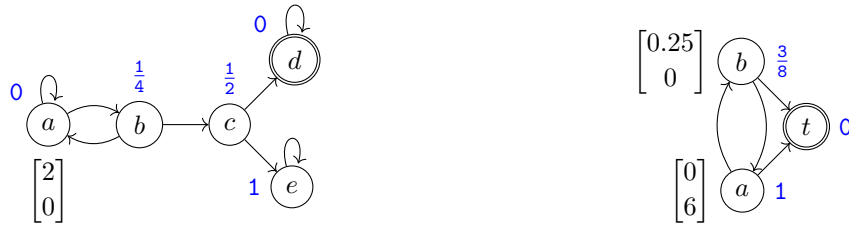
35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 8; pp. 8:1–8:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) Strategies may depend on the available budget.

(b) Nontrivial solution for safety objective of Player 2 when the unsafe vertex (which is t) is reachable from every other vertex.

■ **Figure 1** Examples to demonstrate the distinctive features of bidding games with charging, compared to traditional bidding games (without charging). The double circled vertices are the ones that Player 1 wants to reach (reachability objective), or, dually, the ones that Player 2 wants to avoid (safety objective). When a vertex v has nonzero reward for at least one of the players, the rewards are shown next to v in the vector notation $\begin{bmatrix} R_1(v) \\ R_2(v) \end{bmatrix}$. The threshold budget of Player 1 for each vertex is shown in blue next to the vertex.

the play fulfills a given temporal objective (or specification). Traditionally, graph games are *turn-based*, where the players move the token in alternate turns. *Bidding games* are graph games where who moves the token at each step is determined by an auction (a *bidding*). Concretely, both players are allocated initial budgets, and in each turn, they concurrently place bids from their available budgets, the highest bidder moves the token, and pays his bid according to one of the following pre-determined mechanisms. In *Richman* bidding, the bid is paid to the lower bidder, in *poorman* bidding, the bid is paid to an imaginary “bank” and the money is lost, and in *taxman* bidding, a fixed fraction of the bid is paid to the bank (the “tax”) and the rest goes to the lower bidder. The outcome of the game is an infinite play and, as usual, the winner is determined based on whether the play fulfills a given objective.

Bidding games model strategic decision-making problems where resources need to be invested dynamically towards the fulfillment of an objective. For example, a taxi driver needs to decide how to “invest” his gas supply in order to collect as many passengers as possible, internet advertisers need to invest their advertising budgets in ongoing auctions for advertising slots with the goal of maximizing visibility [5], or a coach in an NBA tournament needs to decide his roster for each game while “investing” his players’ limited energy with the goal of winning the tournament [8]. While in all these scenarios the investment resources can be “charged,” e.g., by visiting a gas station, by adding funds, or by allowing the players to rest, respectively, charging budgets cannot be modeled in traditional bidding games.

We study, for the first time, *bidding games with charging*, where the players can increase their available budgets by collecting vertex-dependent charges. Every vertex v in the arena is labeled with a pair of non-negative rational numbers denoted $R_1(v)$ and $R_2(v)$. Suppose the game enters a vertex v , where for $i \in \{1, 2\}$, Player i ’s budget is B_i with $B_1 + B_2 = 1$. First, the budgets are charged to $B'_1 = B_1 + R_1(v)$ and $B'_2 = B_2 + R_2(v)$. Second, we normalize the sum of budgets to 1 by defining $B''_1 = B'_1 / (B'_1 + B'_2)$ and $B''_2 = B'_2 / (B'_1 + B'_2)$. Finally, the players bid from their new available budgets B''_1 and B''_2 , and the bids are resolved using any of the traditional mechanisms. Note that traditional bidding games are a special case of bidding games with charging in which all charges are 0. The normalization step plays an important role and will be discussed in Ex. 4.

► **Example 1.** We illustrate the model and show a distinctive feature that is not present in traditional bidding games. Consider the bidding game in Fig. 1a, where the objective of Player 1, the reachability player, is to reach d and the objective of Player 2, the safety player, is to prevent this. Consider Richman bidding. We show that from vertex b , Player 1 can win with a budget of $B_1 = \frac{1}{4} + \epsilon \leq 1$ for every $\epsilon > 0$. Player 1 bids $\frac{1}{4}$ at b . We consider two cases. *First*, Player 2 wins the bidding and proceeds to c . She pays Player 1 at least $\frac{1}{4}$, and Player 1's budget at c becomes at least $\frac{1}{2} + \epsilon$. Player 1 can now win the bidding by bidding all of his budget (recall that the sum of budgets is 1), and can proceed to d to win the game. *Second*, suppose that Player 2 loses the bidding at b . Player 1 proceeds to a with a budget of ϵ . We charge his budget to $B'_1 = 2 + \epsilon$ and after re-normalizing his new budget becomes $B''_1 = \frac{B'_1}{3} > \frac{2}{3}$. Player 1 increases his budget by forcing the game to stay in a for three consecutive turns: He first bids $\frac{1}{3}$, and his budget exceeds $(\frac{1}{3} + 2)/3 = \frac{7}{9}$, then he bids $\frac{2}{9}$ and $\frac{4}{27}$ in the following two turns, after which his budget exceeds $\frac{73}{81} > \frac{7}{8}$. Since every budget greater than $\frac{7}{8}$ suffices to guarantee winning three consecutive biddings, he can now force the game to reach d , resulting in a win.

We point out a distinction from traditional bidding games (without charging). In games without charging, it is known that if a player wins, he can win using a *budget agnostic* winning strategy: For every vertex v , there is a successor u such that upon winning the bidding at v , the strategy proceeds to u regardless of the current available budget.¹ However, it is not hard to see that there is no winning budget-agnostic strategy in the game above; indeed, in order to win, Player 1 must eventually go right from b , but when his budget is $0.25 < B_1 \leq 0.75$, he needs to go left and going right will make him lose. ─

Another distinctive feature of bidding games with charging is that safety games have non-trivial solutions, while in traditional bidding games, the only way to ensure safety is by reaching a vertex with no path to the unsafe vertices [21, 4, 7]. Therefore, charging opens doors to new applications of bidding games for when safety objectives are involved. For example, in *auction-based scheduling* [14], bidding games are used to compose two policies at runtime such that the objectives of both policies are fulfilled. With traditional bidding games, auction-based scheduling cannot support long-run safety due to the aforementioned reasons. Bidding games with charging creates the possibility to extend auction-based scheduling for richer classes of objectives than what can be supported currently.

► **Example 2.** We show that Player 2, the safety player, wins the game depicted in Fig. 1b starting from b when Player 1's budget is $B_1 < \frac{3}{8}$. Fulfilling safety requires the game to forever loop over a and b ; such an outcome is not possible in traditional bidding games since t is reachable from both a and b . After charging at b , we have $B''_1 < \frac{1}{2}$. Player 2 bids $\frac{1}{2}$, trivially wins the bid and moves the token to a . Her budget is charged to at least $\frac{6}{7}$, meaning that Player 1's budget is at most $\frac{1}{7}$. She bids $\frac{3}{16}$, trivially wins the bidding and move the token to b . When entering b her budget is at least $\frac{6}{7} - \frac{3}{16} > \frac{5}{8}$, meaning that Player 1's budget is less than $\frac{3}{8}$, and she can keep repeating the same strategy to win the game. ─

The central quantity in bidding games is the pair of *thresholds* on the players' budgets which enable them to win. Formally, for $i \in \{1, 2\}$, Player i 's threshold at vertex v , denoted $Th_i(v)$, is the smallest value in $[0, 1]$ such that for every $\epsilon > 0$, Player i can guarantee winning from v with an initial budget of $Th_i(v) + \epsilon$. The thresholds in the vertices in Figures 1a and 1b are depicted beside them in blue. When $Th_1(v) + Th_2(v) = 1$, we say that a *threshold*

¹ We refrain from calling the strategy *memoryless* since it might bid differently in successive visits to v .

8:4 Bidding Games with Charging

■ **Table 1** Upper complexity bounds for bidding games with charging (“w/ chg.”) in comparison with traditional bidding games (“w/o chg.”).

	Reachability		Safety		Büchi		Co-Büchi	
	w/ chg.	w/o chg.	w/ chg.	w/o chg.	w/ chg.	w/o chg.	w/ chg.	w/o chg.
Richman	coNP	NP \cap coNP	NP	NP \cap coNP	Π_2^P	NP \cap coNP	Σ_2^P	NP \cap coNP
Taxman and poorman	PSPACE	PSPACE	PSPACE	PSPACE	2-EXP	PSPACE	2-EXP	PSPACE

exists and define the threshold to be $Th(v) = Th_1(v)$. Existence of thresholds is a form of *determinacy*: for every Player 1 budget $B_1 \neq Th_1(v)$, one of the players has a winning strategy. We establish that bidding games with charging are also determined for reachability and Büchi objectives, and, dually, for safety and co-Büchi objectives. The proofs of these claims are however significantly more involved than the case of traditional bidding games. For instance, for traditional bidding games, the existence of thresholds for Büchi objectives follows from the existence of thresholds for reachability objectives, with the observation that for every bottom strongly connected component (BSCC), every vertex has a threshold 0 or 1, so that winning the Büchi game boils down to *reaching* one of the BSCCs with thresholds 0 (the “winning” BSCCs). This approach fails for games with charging. First, players may be able to trap the game within an SCC that is not part of any BSCC, and second, the thresholds in a BSCC might not be all 0 or 1 as seen in Ex. 2. In order to show the existence of thresholds in Büchi games, we develop a novel fixed point algorithm that is based on repeated solutions to reachability bidding games.

We study the complexity of finding thresholds. Here too, the techniques differ and are more involved than traditional bidding games. In Richman-bidding games without charging, thresholds correspond to the unique solution of a system of linear equations. In games with charging, however, thresholds correspond to the least and the greatest fixed points, and we present a novel encoding of the problem using mixed-integer linear programming. We summarize our complexity results in Tab. 1 along with a comparison with known results in traditional bidding games. Finally, we show that Richman games with Rabin and Streett objectives are NP-hard and coNP-hard, respectively. This result establishes the first lower complexity bound in any form of bidding games (with or without charging). Upper bounds for Rabin and Streett objectives are left open.

Finally, we introduce and study a *repair* problem in bidding games: Given a bidding game, a target threshold t in a vertex v , and a repair budget C , decide if it is possible to add charges to the vertices of \mathcal{G} in a total sum that does not exceed C such that $Th(v) \leq t$. Repairing is relevant when the bidding game is not merely given to us as a fixed input, but rather the design of the game is part of the solution itself. For instance, we have already mentioned auction-based scheduling [14], where the strongest guarantees can be provided when in two bidding games that are played on the same arena, the sum of thresholds in the initial vertex is less than 1. When this requirement fails, repairing can be applied to lower the thresholds. We show that the repair problem for safety objectives is in PSPACE and for reachability objectives is in 2EXPTIME.

Related work

Bidding games (without charging) were introduced by Lazarus et al. [22, 21], and were extended to infinite-duration objectives by Avni et al. [4, 5, 6, 9]. Many variants of bidding games have been studied, including *discrete*-bidding [19, 1, 12], which restricts the granularity

of bids, *all-pay* bidding [8, 9], which model allocation of non-refundable resources, partial-information games [10], which restricts the observation power of one of the players, and non-zero-sum bidding games [23], which allow the players' objectives to be non-complimentary. The inspiration behind charging comes from other forms of resource-constrained games that allow to refill depleted resources or accumulate new resources to perform certain tasks [17, 16, 15]. The unique challenge in our case is the additional layer of bidding, which separates resource (budget) accumulation and spending.

2 Bidding Games with Charging

A *bidding game with charging* is a two-player game played on an arena $\langle V, E, R_1, R_2 \rangle$ between Player 1 and Player 2,² where V is a finite set of vertices, $E \subseteq V \times V$ is a set of directed edges, and $R_1, R_2 : V \rightarrow \mathbb{R}_{\geq 0}$ are the charging functions of Player 1 and Player 2, respectively. We denote the set of *successors* of the vertex v by $\mathcal{S}(v) = \{u : \langle v, u \rangle \in E\}$. Bidding games with no charging will be referred to as *traditional* bidding games, which is a special case with $R_1 \equiv R_2 \equiv 0$. The default ones in this paper are bidding games with charging and, to avoid clutter, we typically refer to them simply as *bidding games*.

A bidding game proceeds as follows. A *configuration* of a bidding game is a pair $c = \langle v, B_1 \rangle \in V \times [0, 1]$, which indicates that the token is placed on the vertex v and Player 1's current budget is B_1 . We always normalize the sum of budgets to 1, thus, implicitly, Player 2's budget is $B_2 = 1 - B_1$. At configuration c , we charge and normalize the budgets. Formally, the game proceeds to an *intermediate configuration* $c' = \langle v, B'_1 \rangle$ defined by $B'_1 = \frac{B_1 + R_1(v)}{1 + R_1(v) + R_2(v)}$. Player 2's budget becomes $B'_2 = 1 - B'_1 = \frac{B_2 + R_2(v)}{1 + R_1(v) + R_2(v)}$. Then, the players simultaneously bid for the privilege of moving the token. Formally, for $i \in \{1, 2\}$, Player i chooses an *action* $\langle b_i, u_i \rangle$, where $b_i \in [0, B'_i]$ and $u_i \in \mathcal{S}(v)$. Given both players' actions, the next configuration is $\langle u, B''_1 \rangle$, where $u = u_1$ when $b_1 \geq b_2$ and $u = u_2$ when $b_2 > b_1$, and B''_1 is determined based on the *bidding mechanism* defined below. Note that we arbitrarily break ties in favor of Player 1, but it can be shown that all our results remain valid no matter how ties are resolved. In the definitions below we assume that Player 1 is the higher bidder, i.e., $b_1 \geq b_2$, and the case where $b_2 > b_1$ is dual:

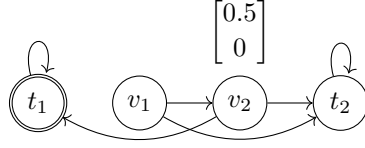
Richman bidding. The higher bidder pays his bid to the lower bidder. Formally, $B''_1 = B'_1 - b_1$, and $B''_2 = B'_2 + b_1$.

Poorman bidding. The higher bidder pays his bid to the bank and we re-normalize the budget to sum up to 1. Formally, $B''_1 = \frac{B'_1 - b_1}{1 - b_1}$, and $B''_2 = \frac{B'_2}{1 - b_1}$.

Taxman bidding. For a predetermined and fixed fraction $\tau \in [0, 1]$, called the *tax rate*, the higher bidder pays fraction τ of his bid to the bank, and the rest to the lower bidder. Formally, $B''_1 = \frac{B'_1 - b_1}{1 - \tau \cdot b_1}$, and $B''_2 = \frac{B'_2 + (1 - \tau) \cdot b_1}{1 - \tau \cdot b_1}$. Note that taxman bidding with $\tau = 0$ coincides with Richman bidding and with $\tau = 1$ coincides with poorman bidding.

In a bidding game, a *history* is a finite sequence $\langle v_0, B_0 \rangle, \langle v_0, B'_0 \rangle, \dots, \langle v_n, B_n \rangle, \langle v_n, B'_n \rangle$ which alternates between configurations and intermediate configurations. For $i \in \{1, 2\}$, a *strategy* for Player i is a function π_i that maps a history to an action $\langle b_i, u_i \rangle$. We typically consider *memoryless* strategies, which are functions from intermediate configurations to actions. An initial configuration $c_0 = \langle v_0, B_0 \rangle$ and two strategies π_1 and π_2 give rise to an infinite *play*, denoted $\text{play}(c_0, \pi_1, \pi_2)$, and is defined inductively, where the inductive step is based on the definitions above. Let $\text{play}(c_0, \pi_1, \pi_2) = \langle v_0, B_0 \rangle, \langle v_0, B'_0 \rangle, \dots$. The *path* that corresponds to $\text{play}(c_0, \pi_1, \pi_2)$ is $v_0, v_1, \dots \in V^\omega$.

² We will use the pronouns "he" and "she" for Player 1 and Player 2, respectively.



■ **Figure 2** Example of a poorman bidding game where without normalization thresholds are not uniquely determined.

Each game is equipped with an *objective* $\varphi \subseteq V^\omega$. Each play has a *winner*. Player 1 wins a play if its corresponding path is in φ , and Player 2 wins otherwise. For an objective φ , a Player 1 strategy π_1 is *winning* from a configuration c if for every Player 2 strategy π_2 , the play $\text{play}(c, \pi_1, \pi_2)$ is winning for Player 1, and the definition for Player 2 is dual. For $i \in \{1, 2\}$, we say Player i *wins* from configuration c for φ if he has a winning strategy from c . We will use \bar{x} to denote the complement of x , where x can be either an objective or a set of vertices. We consider the following objectives:

Reachability. For a set of vertices $T \subseteq V$, the reachability objective is defined as $\text{Reach}(T) := \{v_0 v_1 \dots \in V^\omega \mid \exists i \in \mathbb{N}. v_i \in T\}$. Intuitively, T represents the set of target vertices, and $\text{Reach}(T)$ is satisfied if T is eventually visited by the given path.

Safety. For a set of vertices $S \subseteq V$, the safety objective is defined as $\text{Safe}(S) := \{v_0 v_1 \dots \in V^\omega \mid \forall i \in \mathbb{N}. v_i \in S\}$. Intuitively, S represents the set of safe vertices, and $\text{Safe}(S)$ is satisfied if S is not left ever during the given path. Safety and reachability are dual to each other, i.e., $\text{Safe}(S) = \text{Reach}(\bar{S})$.

Büchi. For a set of vertices $B \subseteq V$, the Büchi objective is defined as $\text{Büchi}(B) := \{v_0 v_1 \dots \in V^\omega \mid \forall i \in \mathbb{N}. \exists j > i. v_j \in B\}$. Intuitively, Büchi(B) is satisfied if B is visited infinitely often during the given path.

Co-Büchi. For a set of vertices $C \subseteq V$, the co-Büchi objective is defined as $\text{Co-Büchi}(C) := \{v_0 v_1 \dots \in V^\omega \mid \exists i \in \mathbb{N}. \forall j > i. v_j \in C\}$. Intuitively, Co-Büchi(C) is satisfied if only C is visited from some point onward during the play. Büchi and co-Büchi objectives are dual to each other, i.e., $\text{Co-Büchi}(C) = \text{Büchi}(\bar{C})$.

A central concept in bidding games is the pair of *thresholds* for the two players. Roughly, they are the smallest budgets needed by the respective player for winning the game from a given vertex. We formalize this below.

► **Definition 3 (Thresholds).** Let \mathcal{G} be a given arena and $M \in \{\text{Richman}, \text{poorman}, \text{taxman}\}$ be a given bidding mechanism. For an objective φ , the thresholds $\text{Th}_1^{\mathcal{G}, M, \varphi}, \text{Th}_2^{\mathcal{G}, M, \varphi}: V \rightarrow [0, 1]$ are functions such that for every $v \in V$ and every $\epsilon > 0$:

- $\text{Th}_1^{\mathcal{G}, M, \varphi}(v) := \inf_{B_1 \in [0, 1]} \{B_1 : \text{Player 1 wins from } \langle v, B_1 + \epsilon \rangle \text{ for } \varphi, \text{ for every } \epsilon > 0\}$.
- $\text{Th}_2^{\mathcal{G}, M, \varphi}(v) := \inf_{B_2 \in [0, 1]} \{B_2 : \text{Player 2 wins from } \langle v, 1 - B_2 - \epsilon \rangle \text{ for } \bar{\varphi}, \text{ for every } \epsilon > 0\}$.

When $\text{Th}_1^{\mathcal{G}, M, \varphi}(v) + \text{Th}_2^{\mathcal{G}, M, \varphi}(v) = 1$ for every vertex v , we say that the threshold exists in \mathcal{G} , denote it $\text{Th}^{\mathcal{G}, M, \varphi}(v)$, and define $\text{Th}^{\mathcal{G}, M, \varphi}(v) = \text{Th}_1^{\mathcal{G}, M, \varphi}(v)$.

Whenever the game graph and the bidding mechanism are clear from the context, we simply write Th_1^φ , Th_2^φ , and Th^φ .

► **Example 4 (The importance of normalization).** Consider the poorman bidding game that is depicted in Fig. 2. Intuitively, Player 1 wins from v_1 iff he wins the first two consecutive biddings. Formally, the game starts at v_1 and t_i is Player i 's target, for $i \in \{1, 2\}$. We first analyze the game with a normalization step. We argue that $\text{Th}(v_2) = \frac{1}{4}$; indeed, since

$\frac{1/4+1/2}{3/2} = \frac{1}{2}$, entering v_2 with a budget greater than $\frac{1}{4}$ allows Player 1 to secure winning. We argue that $Th(v_1) = \frac{4}{7}$; indeed, Player 1 must bid above Player 2's budget and win the bidding, and note that $\frac{4/7-3/7}{3/7} = \frac{1}{4}$. Note that thresholds are in fact a *ratio*. Stated differently, consider a configuration $\langle v_1, B_1, B_2 \rangle$ with $B_1 + B_2$ not necessarily equals 1, then Player 1 wins iff $\frac{B_1}{B_1+B_2} > \frac{4}{7}$. Crucially, the ratio between B_1 and B_2 is fixed. We will prove that this is a general phenomenon on which our algorithms depend.

When a normalization step is *not* performed, Player 1's threshold for winning is a non-linear function of Player 2's initial budget. Intuitively, when no normalization is performed, the charge is more meaningful when the budgets are smaller. Consider a configuration $\langle v_1, B_1, B_2 \rangle$ with $B_1 + B_2$ being not necessarily equal to 1. Note that Player 1 must win the first bidding, thus the second configuration must be $\langle v_2, B_1 - B_2, B_2 \rangle$. When no normalization is performed after charging, the intermediate configuration is $\langle v_2, B_1 - B_2 + 0.5, B_2 \rangle$. Clearly, Player 1 wins iff $B_1 - B_2 + 0.5 > B_2$. For example, when $B_2 = 1$, then Player 1's threshold is $\frac{3}{2}$ and when $B_2 = 2$, then Player 1's threshold is $\frac{7}{2}$. These amount to ratios of $\frac{3}{5}$ and $\frac{7}{11}$, respectively, meaning that Player 1's threshold is a non-linear function of Player 2's budget. We point out that this is also the case in poorman discrete-bidding games [11], where thresholds can only be approximated, even in extremely simple games. \square

We formulate the decision problem related to the computation of thresholds. We will write that a given objective φ is of type Reach, Safe, Büchi, or Co-Büchi if φ can be expressed as a reachability, safety, Büchi, or co-Büchi objective (on a given arena), respectively.

► **Definition 5** (Finding threshold budgets). *Let $M \in \{\text{Richman, poorman, taxman}\}$, and $S \in \{\text{Reach, Safe, Büchi, Co-Büchi}\}$. The problem THRESH_S^M takes as input an arena \mathcal{G} , an initial vertex v , and an objective $\varphi \in S$, and accepts the input iff $Th_1^{\mathcal{G}, M, \varphi}(v) \leq 0.5$.*

3 Reachability Bidding Games with Charging

In this section, we show the existence of thresholds in taxman-bidding games with charging with reachability and, dually, with safety objectives. Throughout this section, we fix an arena $\mathcal{G} = \langle V, E, R_1, R_2 \rangle$. For a given set of vertices $T \subseteq V$, the objective of Player 1, the reachability player, is $\text{Reach}(T)$, and, the objective of Player 2, the safety player, is $\text{Safe}(\overline{T})$.

3.1 Bounded-Horizon Reachability and Safety

We start with the simpler case of *bounded-horizon* reachability objectives, and in the next section, we will extend the technique to general games. Let $t \in \mathbb{N}$. The bounded-horizon reachability, denoted $\text{Reach}(T, t)$, intuitively requires Player 1 to reach T within t steps. Formally, $\text{Reach}(T, t) := \{v_0 v_1 \dots \mid \exists i \leq t. v_i \in T\}$. *Bounded-horizon safety* is the dual objective $\text{Safe}(\overline{T}, t) := \{v_0 v_1 \dots \mid \forall i \leq t. v_i \notin T\} = V^\omega \setminus \text{Reach}(T, t)$.

In the following, we characterize the thresholds for $\text{Reach}(T, t)$ and $\text{Safe}(\overline{T}, t)$ by induction on t . The induction step relies on the following operator on functions.

► **Definition 6.** *Define the function $\text{clamp}_{[0,1]}(x) := \min(1, \max(0, x))$; that is, given x , $\text{clamp}_{[0,1]}(x) = x$, when $0 < x < 1$, and otherwise it “saturates” x at the boundaries 0 or 1. Let $\tau \in [0, 1]$ be the tax rate. We define two operators on functions $Av_1, Av_2: [0, 1]^V \rightarrow [0, 1]^V$ as follows. For $i \in \{1, 2\}$ and $f \in [0, 1]^V$:*

$$Av_i(f)(v) := \text{clamp}_{[0,1]} \left(\frac{(1-\tau)f(v^-) + f(v^+)}{[f(v^+) - f(v^-) - 1]\tau + 2} \cdot (1 + R_1(v) + R_2(v)) - R_i(v) \right)$$

where v^+ and v^- are the successors of v with the largest and the smallest value of $f(\cdot)$, respectively, i.e., $v^+ = \arg \max_{u \in \mathcal{S}(v)} f(u)$ and $v^- = \arg \min_{u \in \mathcal{S}(v)} f(u)$.

Note that for Richman bidding, i.e., when $\tau = 0$, for $i \in \{1, 2\}$, we have

$$Av_i(f)(v) := \text{clamp}_{[0,1]} \left(\frac{f(v^+) + f(v^-)}{2} \cdot (1 + R_1(v) + R_2(v)) - R_i(v) \right)$$

In this case, the function Av_i computes the average (the name “ Av ” stands for “average”) of its argument f on v^- and v^+ , and then performs an affine transformation followed by the saturation $\text{clamp}_{[0,1]}(\cdot)$ on the result. For poorman bidding, i.e., when $\tau = 1$, we have

$$Av_i(f)(v) := \text{clamp}_{[0,1]} \left(\frac{f(v^+)}{f(v^+) - f(v^-) + 1} \cdot (1 + R_1(v) + R_2(v)) - R_i(v) \right)$$

We define two functions f_1 and f_2 which will be shown to coincide with the thresholds.

► **Definition 7.** *Define the functions $f_1, f_2: V \times \mathbb{N} \rightarrow [0, 1]$ inductively on t . For every $v \in T$ and $t \in \mathbb{N}$, define $f_1(v, t) := 0$ and $f_2(v, t) := 1$. For every $v \notin T$, define $f_1(v, 0) := 1$ and $f_2(v, 0) := 0$, and for every $t > 0$, define $f_1(v, t) := Av_1(f_1(\cdot, t-1))(v)$ and $f_2(v, t) := Av_2(f_2(\cdot, t-1))(v)$.*

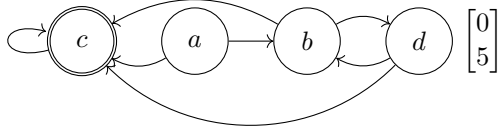
Lem. 8 shows that f_1 and f_2 coincide with the thresholds of the (bounded-horizon) reachability and safety players, respectively. Intuitively, for $\text{Reach}(T, 0)$, Player 1 wins with even zero budget from vertices that are already in T , and loses with even the maximum budget from vertices that are not in T . We capture this as $f_1(v, 0) = 0$ if $v \in T$, and $f_1(v, 0) = 1$ otherwise. Furthermore, if Player 1 has a budget more than $f_1(v, t)$ at v , then we show that he has a memoryless policy such that no matter which vertex v' the token reaches in the next step, his budget will remain more than $f_1(v', t-1)$. It follows inductively that he will reach T in t steps from v . The argument for the safety player is dual. Lem. 8 also establishes the existence of thresholds.

► **Lemma 8.** *For every vertex $v \in V$ and $t \geq 0$, we have $Th_1^{\text{Reach}(T,t)}(v) = f_1(v, t)$ and $Th_2^{\text{Reach}(T,t)}(v) = f_2(v, t)$. Moreover, thresholds exist: $Th_1^{\text{Reach}(T,t)}(v) + Th_2^{\text{Reach}(T,t)}(v) = 1$.*

Proof. We sketch the proof for Richman bidding and the full proof can be found in the extended version of the paper [13]. We show $f_1(v, t) \geq Th_1^{\text{Reach}(T,t)}(v)$. It is dual to show $f_2(v, t) \geq Th_2^{\text{Reach}(T,t)}(v)$, and the other directions of the inequalities follow from the relationship $f_1(v, t) = 1 - f_2(v, t)$, which is not hard to verify. The proof of $f_1(v, t) \geq Th_1^{\text{Reach}(T,t)}(v)$ proceeds by induction over t . The base case, $t = 0$, is not hard to verify. For $t \geq 1$, assume that $f_1(v, t-1) \geq Th_1^{\text{Reach}(T,t-1)}(v)$, and we prove the claim for t . Let $v \in V$ and $B_1 > f_1(v, t)$. We describe a Player 1 winning strategy from $\langle v, B_1 \rangle$. Player 1 bids $b_1 = \frac{f_1(v^+, t-1) - f_1(v^-, t-1)}{2}$, and proceeds to v^- upon winning the bidding (recall that v^- is the successor of v that attains the minimal value of $f_1(\cdot, t-1)$). If Player 1 wins the bidding, he pays b_1 to Player 2, and it can be verified that his new available budget in the next vertex v^- remains above $f_1(v^-, t-1)$. On the other hand, if Player 1 loses the bidding, he receives at least b_1 from Player 2 (because Player 2 must have bid higher than b_1), and the token is moved by Player 2 to some successor v' of v . It can be verified that even in this case, Player 1's new available budget remains above $f_1(v^+, t-1) > f_1(v', t-1)$. By the induction hypothesis, from the new vertex Player 1 can reach T in at most $t-1$ steps. Therefore, from v Player 1 can reach T in at most t steps. ◀

The following lemma establishes monotonicity of f_1 and f_2 with respect to t , which will play a key role in the proof of existence of thresholds for the unbounded counterparts of the objectives. Intuitively, reaching T within t steps is harder than reaching T within $t' > t$ turns, thus less budget is needed for the latter case. Dually, guaranteeing safety for t turns is easier than guaranteeing safety for $t' > t$ turns.

► **Lemma 9.** *For $v \in V$ and $t' > t$, it holds that $f_1(v, t') \leq f_1(v, t)$ and $f_2(v, t') \geq f_2(v, t)$.*



■ **Figure 3** Non-unique fixed points of the threshold update functions.

$t \backslash v$	a	b	c	d	e
0	1	1	1	0	1
1	1	1	0.5	0	1
2	1	0.75	0.5	0	1
3	0.625	0.75	0.5	0	1
4	0.0625	0.5625	0.5	0	1
5	0	0.28125	0.5	0	1
≥ 6	0	0.25	0.5	0	1

■ **Figure 4** Finite-Horizon reachability thresholds for the game in Fig. 1a. The number at row i and column v is $Th_1^{\text{Reach}(d,i)}(v)$.

3.2 Existence of Thresholds (for Reachability and Safety Objectives)

We define two functions f_1^* and f_2^* which will be shown to coincide with the thresholds for the unbounded horizon reachability and safety objectives, respectively.

► **Definition 10.** Define the functions $f_1^*, f_2^*: V \rightarrow [0, 1]$, such that for every $v \in V$:

$$f_1^*(v) := \lim_{t \rightarrow \infty} f_1(v, t) \quad \text{and} \quad f_2^*(v) := \lim_{t \rightarrow \infty} f_2(v, t).$$

Since f_1 and f_2 are bounded in $[0, 1]$ and monotonic by Lem. 9, the limits in Def. 10 are well defined. Since $f_1(v, 0)$ and $f_2(v, 0)$ assign, respectively, the maximum (i.e., 1) and the minimum (i.e., 0) value to every vertex $v \notin T$, hence from the Kleene fixed point theorem, it follows that f_1^* and f_2^* will be, respectively, the greatest and the least fixed points of the operators Av_1 and Av_2 on the directed-complete partial order $\langle [0, 1]^V, \leq \rangle$.

► **Proposition 11.** Consider the directed-complete partial order $L = \langle [0, 1]^V, \leq \rangle$, where for every $x, y \in [0, 1]^V$, $x \leq y$ iff $x_i \leq y_i$ for every $i \in V$. The functions f_1^* and f_2^* are, respectively, the greatest and the least fixed points of the functions Av_1 and Av_2 on L , subjected to the constraints $f_1^*(v) = 0$ and $f_2^*(v) = 1$ for every $v \in T$.

The following example demonstrates that, unlike traditional bidding games, the fixed points of the functions Av_1 and Av_2 on L may not be unique.

► **Example 12** (Multiple fixed-points). Consider the bidding game in Fig. 3, where the objective of Player 1 is to reach c . It can be easily verified that both $f_1^* = \{a \mapsto 0.25, b \mapsto 0.5, c \mapsto 0, d \mapsto 1\}$ and $f_1^{*'} \equiv 0$ are fixed points of the operator Av_1 over L in this case. ◻

The following theorem establishes the existence of thresholds.

► **Theorem 13.** For every vertex $v \in V$, it holds that $Th_1^{\text{Reach}(T)}(v) = f_1^*(v)$ and $Th_2^{\text{Reach}(T)}(v) = f_2^*(v)$. Moreover, thresholds exist: $Th_1^{\text{Reach}(T)}(v) + Th_2^{\text{Reach}(T)}(v) = 1$.

Proof. We prove that $f_1^*(v) \geq Th_1^{\text{Reach}(T)}(v)$, for every $v \in V$. Let $B_1 > f_1^*(v)$. There exists a $t \in \mathbb{N}$ such that $B_1 > f_1(v, t)$. Player 1 uses the strategy from Lem. 8, guaranteeing that T is reached within t steps. Next, we prove that $f_2^*(v) \geq Th_2^{\text{Reach}(T)}(v)$, for every $v \in V$. Let $B_2 > f_2^*(v)$. We know that $f_2^* = Av_2(f_2^*)$ (from Prop. 11), and let v^+, v^- be the successors of v with the greatest and the least value of f_2 . Player 2 bids $b_2 = \frac{f_2(v^+) - f_2(v^-)}{[f_2(v^+) - f_2(v^-) - 1]\tau + 2}$, which evaluates under Richman bidding to $b_2 = \frac{f_2^*(v^+) - f_2^*(v^-)}{2}$ and under poorman bidding

8:10 Bidding Games with Charging

to $b_2 = \frac{f_2(v^+) - f_2(v^-)}{f_2(v^+) - f_2(v^-) + 1}$. Player 2 proceeds to v^- upon winning. In the extended version [13], we prove that no matter how Player 1 bids, Player 2's strategy guarantees that in the next vertex v' her new budget $B'_2 > f_2^*(v')$. Recall that by construction, $f_2^*(v'') = 1$, for every $v'' \in T$. Thus, T is never reached since Player 2's budget can never exceed 1. Finally, the other directions, i.e., the inequalities $f_1^*(v) \leq Th_1^{\text{Reach}(T)}(v)$ and $f_2^*(v) \leq Th_2^{\text{Reach}(T)}(v)$, and the existence claim follows from the observation that $f_1^*(v) = 1 - f_2^*(v)$, for every $v \in V$. ◀

It follows that the threshold can be computed using fixed point iterations sketched in Prop. 11; this iterative approach is illustrated in the following example.

► **Example 14.** Consider the bidding game in Fig. 1a with Richman bidding. The finite horizon reachability thresholds are depicted in table 4. Suppose the game starts from $\langle a, 0.1 \rangle$ which is winning for the reachability player. In this case, $t = 4$ is the smallest integer for which Player 1's budget 0.1 at a is larger than $Th_1^{\text{Reach}(d,t)}(a)$, which is 0.0625. Therefore, according to the strategy of Player 1 as described in the proof of Lem. 8, Player 1 has a strategy for reaching d in 4 steps. First, his budget is charged to $\frac{0.1+2}{3} = 0.7$. His winning strategy (described in the proof of Lem. 8) dictates that he should bid $\frac{Th_1^{\text{Reach}(d,3)}(b) - Th_1^{\text{Reach}(d,3)}(a)}{2} = 0.0625$. In case of winning, he pays the bid to the safety player and keeps the token at a , with budget 0.6375 which is then charged to 0.87916. This is enough for winning 3 consecutive biddings and moving the token to d . In case of losing the first bid, his budget will increase to at least 0.76. This amount is more than both $Th_1^{\text{Reach}(d,3)}(a)$ and $Th_1^{\text{Reach}(d,3)}(b)$, therefore he can guarantee a win in at most 3 steps.

Now suppose the game starts from $\langle b, 0.2 \rangle$ meaning that the safety player has 0.8 budget and can win the game. She has a budget-agnostic strategy which dictates her to bid 0.25 in b (see the proof of Lem. 8 for a sketch of Player 2's strategy). She definitely wins this bidding as her opponent has only 0.2 budget. She then moves the token to c and pays 0.25 to the reachability player, leaving her with 0.55 of the total budget. She can then bid 0.5, win the bidding and move the token to e , where the token stays indefinitely. ▽

3.3 Complexity Bounds (for Reachability and Safety Objectives)

Since f_1^* and f_2^* are fixed points of the operators Av_1 and Av_2 , respectively, hence $f_1^* = Av_1(f_1^*)$ and $f_2^* = Av_2(f_2^*)$. Moreover, f_1^* is the greatest fixed point, which means that $Th_1^{\text{Reach}(T)} = f_1^*$ can be computed by finding the element-wise maximum function h in $[0, 1]^V$ that satisfies $h(v) = 0$ for $v \in T$ and $h(v) = Av_1(h)(v)$ for $v \notin T$. This is formalized below:

$$\begin{aligned} & \max_h \sum_{v \in V} h(v) \\ & \text{subjected to constraints:} \\ & \forall v \in T . h(v) = 0, \\ & \forall v \notin T . h(v) = Av_1(h(\cdot))(v) \\ & \quad = \text{clamp}_{[0,1]} \left(\frac{(1-\tau)h(v^-) + h(v^+)}{[h(v^+) - h(v^-) - 1]\tau + 2} \cdot (1 + R_1(v) + R_2(v)) - R_1(v) \right), \\ & h(v^+) = \max_{u \in S(v)} h(u), \quad h(v^-) = \max_{u \in S(v)} h(u). \end{aligned} \tag{1}$$

► **Proposition 15.** *The solution of the optimization problem in (1) is equivalent to the threshold function $Th_1^{\text{Reach}(T)}$ of the reachability player.*

Due to Thm. 13, for every vertex $v \in V$, we have $Th_2^{\text{Reach}(T)}(v) = 1 - Th_1^{\text{Reach}(T)}(v)$. Consequently, we obtain the following upper complexity bounds.

► **Theorem 16.** *The following hold:*

- (i) $\text{THRESH}_{\text{Reach}}^{\text{taxman}} \in \text{PSPACE}$ and $\text{THRESH}_{\text{Safe}}^{\text{taxman}} \in \text{PSPACE}$,
- (ii) $\text{THRESH}_{\text{Reach}}^{\text{Richman}} \in \text{coNP}$ and $\text{THRESH}_{\text{Safe}}^{\text{Richman}} \in \text{NP}$.

Proof.

Proof of (i). It can be shown that we can construct a polynomially sized (w.r.t. the game) function $\varphi: \mathbb{R}^V \rightarrow \mathbb{B}$ such that for every $h \in \mathbb{R}^V$, $\varphi(h)$ is true iff h is a fixed point of Av_1 (details can be found in the extended version [13]). Hence, if the system has a solution where $h(v) > 0.5$, the greatest fixed point $Th_1^{\text{Reach}(T)}$ satisfies $Th_1^{\text{Reach}(T)}(v) > 0.5$. This is an instance of existential theory of reals which is known to be in PSPACE. Therefore, it is possible to decide $Th_1^{\text{Reach}(T)}(v) > 0.5$ (equivalently $Th_2^{\text{Reach}(T)}(v) < 0.5$) in PSPACE.

Proof of (ii). We provide the following reduction from the optimization problem to an instance of MILP; a different proof with a polynomial certificate is provided in the extended version [13]. Let O be the optimization problem as stated in the Section 3 with $\tau = 0$.

Let M be any constant strictly greater than $\max_{u \in V} \{1 + R_1(u) + R_2(u)\}$. For each node u define two new variables $h^-(u), h^+(u)$ and add the following constraints to O :

$$\begin{aligned} h^+(w) &\geq h(w) \quad \forall w \in \mathcal{S}(u) & h^-(w) &\leq h(w) \quad \forall w \in \mathcal{S}(u) \\ h^+(w) &\leq h(w) + (1 - b_u^w) \cdot M \quad \forall w \in \mathcal{S}(u) & h^-(w) &\geq h(w) - (1 - c_u^w) \cdot M \quad \forall w \in \mathcal{S}(u) \\ \sum_{w \in \mathcal{S}(u)} b_u^w &= 1 & \sum_{w \in \mathcal{S}(u)} c_u^w &= 1 \\ b_u^w &\in \{0, 1\} \quad \forall w \in \mathcal{S}(u) & c_u^w &\in \{0, 1\} \quad \forall w \in \mathcal{S}(u) \end{aligned}$$

This guarantees that $h^+(u) = h(u^+)$ and $h^-(u) = h(u^-)$, so they can be replaced. Next, replace each $\min(1, x)$ by $\frac{(x-1)-|x-1|}{2} + 1$ and $\max(0, x)$ by $\frac{x+|x|}{2}$. Then replace each $|y|$ with a fresh variable a_y and add the following constraints to O :

1. $-a_y \leq y \leq a_y$
2. $y + M \cdot z_y \geq a_y \wedge -y + M \cdot (1 - z_y) \geq a_y \wedge z_y \in \{0, 1\}$

The first constraint ensures that $|y| \leq a_y$ and the second one that $|y| \geq a_y$. Therefore, it is guaranteed that $|y| = a_y$. The MILP instance O is equivalent to the optimization problem in section 3. In order to decide whether $Th_1(v) \geq 0.5$ it suffices to decide satisfiability of O with the additional constraint that $h(v) \geq 0.5$ and this decision problem is known to be in NP. ◀

4 Büchi Bidding Games with Charging

We proceed to Büchi objectives, for which the proof of existence of thresholds is shown to be significantly more involved than for Büchi games without charging. The key distinction is that thresholds in traditional strongly-connected Büchi games are trivial: If even one of the vertices is a Büchi target vertex, the Büchi player's threshold in *each* vertex is 0 and otherwise is 1 [3]. This property gives us a simple reduction from traditional Büchi bidding games to reachability bidding games. With charging, this property no longer holds. For example, alter the game in Fig. 1b to make it strongly-connected by adding an edge from t to b . The thresholds remain above 0, i.e., there are initial budgets with which Player 2 wins.

Our existence proof, which is inspired by an existence proof for discrete-bidding games [12], follows a fixed-point characterization that is based on solutions to *frugal-reachability* games, which are defined below. We note that the proof has a conceptual similarity with Zielonka's algorithm [27] in turn-based Büchi games, which characterizes the set of winning vertices based on repeated calls to an algorithm for turn-based reachability games.

4.1 Frugal-Reachability Objectives

We introduce frugal reachability objectives. Consider a taxman-bidding game with charging $\mathcal{G} = \langle V, E, R_1, R_2 \rangle$. Let $T \subseteq V$ be a set of target vertices and $\text{fr} : T \rightarrow [0, 1]$ be a function that assigns each target with a *frugal budget*. The *frugal reachability* objective $\text{FrugalReach}(T, \text{fr})$ requires Player 1 to reach T such that the first time a vertex $v \in T$ is reached, Player 1's budget must exceed $\text{fr}(v)$, thus:

$$\text{FrugalReach}(T, \text{fr}) := \{ \langle v_0, B_1^0 \rangle \langle v_1, B_1^1 \rangle \dots \mid \exists i . v_i \in T \wedge B_1^i > \text{fr}(v_i) \wedge \forall j < i . v_j \notin T \}$$

We stress that $\text{FrugalReach}(T, \text{fr})$ is a set of *plays*, whereas the other objectives we have considered so far (reachability, Büchi, etc.) were sets of *paths*.

Existence of thresholds $Th_1^{\text{FrugalReach}(T, \text{fr})}$ and $Th_2^{\text{FrugalReach}(T, \text{fr})}$ for the frugal-reachability objective and its dual are shown in the following theorem. The proof can be found in the extended version [13] and follows similar arguments as reachability bidding games with the following change in the base case. For $v \in T$ and $t \in \mathbb{N}$, recall that we define $f_1(v, t) = 0$ (Def. 7), which intuitively means that Player 1 wins if he reaches v with *any* budget. Instead, we now define $f_1(v, t) = \text{fr}(v)$, requiring Player 1 to reach v with a budget of $\text{fr}(v)$. Dually, we define $f_2(v, t) = 1 - \text{fr}(v)$.

► **Theorem 17.** *The thresholds $Th_1^{\text{FrugalReach}(T, \text{fr})}$ and $Th_2^{\text{FrugalReach}(T, \text{fr})}$ exist.*

4.2 Bounded-Visit Büchi and Co-Büchi

We first prove the existence of thresholds for the simpler case of bounded-visit Büchi and co-Büchi objectives, where we impose, respectively, lower and upper bounds on the number of visits to the Büchi target vertices $B \subseteq V$. Let $k \in \mathbb{N}$ be a given bound. The bounded-visit Büchi, denoted as $\text{Büchi}(B, k)$, intuitively requires Player 1 to visit B at least k times. Formally, $\text{Büchi}(B, k) := \{v_0 v_1 \dots \mid |\{i \in \mathbb{N} \mid v_i \in B\}| \geq k\}$. Bounded-visit co-Büchi is the dual objective $\text{Co-Büchi}(\overline{B}, k) := \{v_0 v_1 \dots \mid |\{i \in \mathbb{N} \mid v_i \in B\}| < k\} = V^\omega \setminus \text{Büchi}(B, k)$.

Like before, we introduce two functions g_1 and g_2 , which will be shown to characterize the thresholds for $\text{Büchi}(B, k)$ and $\text{Co-Büchi}(B, k)$, respectively.

► **Definition 18.** *Define the functions $g_1, g_2 : V \times \mathbb{N} \rightarrow [0, 1]$ inductively as follows. For every $v \in V$, define $g_1(v, 0) = 0$. For every $v \in B$, define $g_1(v, 1) := 0$ and $g_1(v, k) := Av_1(g_1(\cdot, k-1))(v)$ for $k > 1$, and for every $v \notin B$ and every $k > 0$, define $g_1(v, k) := Th_1^{\text{FrugalReach}(B, g_1(\cdot, k))}(v)$. We proceed to define g_2 . For every $v \in V$, define $g_2(v, 0) := 1$. For every $v \in B$, define $g_2(v, 1) := 1$ and $g_2(v, k) := Av_2(g_2(\cdot, k-1))(v)$, for $k > 1$, and for every $v \notin B$ and every $k > 0$, define $g_2(v, k) := Th_2^{\text{FrugalReach}(B, 1-g_2(\cdot, k))}(v)$.*

We prove the existence of thresholds and their correspondence to g_1 and g_2 .

► **Lemma 19.** *For every $v \in V$ and $k \geq 0$, we have $g_1(v, k) = Th_1^{\text{Büchi}(B, k)}(v)$ and $g_2(v, k) = Th_2^{\text{Co-Büchi}(B, k)}(v)$. Moreover, the thresholds exist: $Th_1^{\text{Büchi}(B, k)}(v) + Th_2^{\text{Co-Büchi}(B, k)}(v) = 1$.*

Proof. The proof proceeds by induction over k (see details in the extended version [13]). For the base case, $k = 0$, clearly, Player 1 wins since no further visit to B is required and Player 2 loses, which coincides with the definitions $g_1(v, 0) = 0$ and $g_2(v, 0) = 1$, for all $v \in V$. We describe the inductive step for Player 1. For $v \in B$, the proof follows from the induction hypothesis: similar to Lem. 8, when $B_1 > g_1(v, k)$, Player 1 can bid so that no matter how Player 2 bids and moves the token (upon winning), in the next configuration $\langle v', B'_1 \rangle$, we have $B'_1 > g_1(v', k - 1)$. Finally, for $v \notin B$, recall that $g_1(v, k) := Th_1^{\text{FrugalReach}(B, g_1(\cdot, k))}(v)$. That is, a budget of $B_1 > g_1(v, k)$ means that he can follow a winning strategy in the frugal-reachability game, which forces the game to B and upon reaching $v' \in B$, Player 1's budget exceeds $g_1(v', k)$. The proof then follows from the induction hypothesis. ◀

We establish monotonicity of the thresholds, which confirms that *Player 1* needs higher budget for forcing larger numbers of visits to B .

► **Lemma 20.** *For $v \in V$ and $k \in \mathbb{N}$, we have $Th_1^{\text{Büchi}(B, k)}(v) \leq Th_1^{\text{Büchi}(B, k+1)}(v)$ and $Th_2^{\text{Büchi}(B, k)}(v) \geq Th_2^{\text{Büchi}(B, k+1)}(v)$. Moreover, the thresholds are bounded by 0 and 1.*

4.3 Existence of Thresholds (for Büchi and Co-Büchi Objectives)

We define two functions g_1^* and g_2^* , which will be shown to coincide with the thresholds for the general (unbounded) Büchi and co-Büchi objectives, respectively.

► **Definition 21.** *Define the functions $g_1^*, g_2^*: V \rightarrow \mathbb{R}$ as follows. For every $v \in B$, define $g_1^*(v) := \lim_{k \rightarrow \infty} g_1(v, k)$ and $g_2^*(v) := \lim_{k \rightarrow \infty} g_2(v, k)$. For every $v \notin B$, define $g_1^*(v) := Th_1^{\text{FrugalReach}(B, \text{fr})}(v)$ where $\text{fr}: b \mapsto g_1^*(b)$ for every $b \in B$ and $\text{fr}: v \mapsto 0$ (can be arbitrary) for every $v \notin B$. Likewise, for every $v \notin B$, define $g_2^*(v) := Th_2^{\text{FrugalReach}(B, \text{fr})}(v)$ where $\text{fr}: b \mapsto 1 - g_2^*(b)$ for every $b \in B$ and $\text{fr}: v \mapsto 0$ (can be arbitrary) for every $v \notin B$.*

Monotonicity (Lem. 20) and boundedness of g_1 and g_2 imply the well-definedness of g_1^* and g_2^* . We now establish the existence and the characterization of thresholds.

► **Theorem 22.** *For every $v \in V$, we have $Th_1^{\text{Büchi}(B)}(v) = g_1^*(v)$ and $Th_2^{\text{Büchi}(B)}(v) = g_2^*(v)$. Moreover, thresholds exist: $Th_1^{\text{Büchi}(B)}(v) + Th_2^{\text{Büchi}(B)}(v) = 1$.*

Proof. First, we show that $g_1^*(v) \geq Th_1^{\text{Büchi}(B)}(v)$. Consider a configuration $\langle v, B_1 \rangle$. When $B_1 > g_1^*(v)$, Player 1 wins as follows. If $v \notin B$, he plays according to a winning strategy in a frugal-reachability game to guarantee reaching some $v' \in B$ with a budget that exceeds $g_1^*(v')$. For $v \in B$, he bids so that in the next configuration $\langle v', B'_1 \rangle$, we have $B'_1 > g_1^*(v')$. Second, we show that $g_2^*(v) \geq Th_2^{\text{Büchi}(B)}(v)$. When $B_2 = 1 - B_1 > g_2^*(v)$, Player 2 wins as follows. If $v \in B$, then there exists k such that $B_2 > g_2(v, k)$. Lem. 19 shows that she can win the co-Büchi objective by preventing B to be reached more than k times. If $v \notin B$, she has a strategy to make the token either (i) not reach B , or (ii) reach $v' \in B$ with a budget at least $g_2^*(v')$. In both cases, she wins by repeating the strategy. Finally, by Lem. 19, we have $g_1(v, k) + g_2(v, k) = 1$, for all $k \in \mathbb{N}$. Thus, in the limit, we have $g_1^*(v) + g_2^*(v) = 1$, for $v \in B$. From this, the other sides of the above inequalities, i.e., $g_1^*(v) \leq Th_1^{\text{Büchi}(B)}(v)$ and $g_2^*(v) \leq Th_2^{\text{Büchi}(B)}(v)$, and the existence claim follow in a straightforward manner. ◀

4.4 Complexity Bounds (for Büchi and Co-Büchi Objectives)

The computation of the thresholds $Th_1^{\text{Büchi}(B)} \equiv g_1^*$ and $Th_2^{\text{Büchi}(B)} \equiv g_2^*$ involves a nested fixed point computation. For example, for g_1^* , the outer fixed point is the smallest fixed point of the sequence $g_1(\cdot, 0), g_1(\cdot, 1), \dots$ for vertices in B , and for every $k = 0, 1, \dots$, the

8:14 Bidding Games with Charging

inner fixed point is the usual greatest fixed point for frugal reachability thresholds required to reach B with the leftover frugal budget $g_1(\cdot, k)$ from outside B . The nested fixed point can be characterized as the solution of the following bilevel optimization problem.

$$\min_{h \in \mathbb{R}^V} \sum_{b \in B} h(b)$$

subjected to constraints:

$$\begin{aligned} h &\in \arg \max_{h' \in \mathbb{R}^V} \left\{ \sum_{v \in V \setminus B} h'(v) \mid \forall b \in B . h'(b) = h(b) \right\}, \\ \forall v \in V . h(v) &= Av_1(h(\cdot))(v) \\ &= \text{clamp}_{[0,1]} \left(\frac{(1-\tau)h(v^-) + h(v^+)}{[h(v^+) - h(v^-) - 1]\tau + 2} \cdot (1 + R_1(v) + R_2(v)) - R_1(v) \right), \\ h(v^+) &= \max_{u \in S(v)} h(u), \quad h(v^-) = \min_{u \in S(v)} h(u). \end{aligned} \tag{2}$$

► **Proposition 23.** *The solution of the optimization problem in (2) is equivalent to the threshold function $Th_1^{\text{Büchi}(B)}$ of the Büchi player.*

► **Theorem 24.** *The following hold:*

- (i) $\text{THRESH}_{\text{Büchi}}^{\text{taxman}} \in 2\text{EXPTIME}$ and $\text{THRESH}_{\text{Co-Büchi}}^{\text{taxman}} \in 2\text{EXPTIME}$, and
- (ii) $\text{THRESH}_{\text{Büchi}}^{\text{Richman}} \in \Pi_2^P$ and $\text{THRESH}_{\text{Co-Büchi}}^{\text{Richman}} \in \Sigma_2^P$

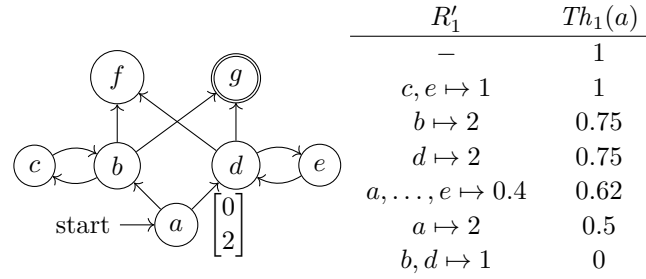
The bounds in (i) follow from a reduction to an equivalent query in the theory of reals. For (ii), we can check if the solution of the optimization problem in (2) is larger than 0.5, and if this is true then we conclude that $0.5 < Th_1^{\text{Büchi}(B)}(v)$ and can output a *negative* answer. Since (2) is a bilevel MILP, hence the check can be done in Σ_2^P [20], and the overall complexity is Π_2^P . The other case is dual. Details of the proof can be found in the extended version [13].

5 Lower Complexity Bounds

In this section we show how to simulate a turn-based game using a Richman-bidding game with charging. Thus, solving Richman-bidding games with charging is at least as hard as their turn-based counterparts. Specifically, we obtain that solving Rabin bidding games with charging is NP-hard. This is a distinction from traditional Richman-bidding games, where solving Rabin games is in NP and coNP. Since taxman-bidding games generalize Richman-bidding games, hence it follows that Rabin taxman-bidding games are also NP-hard.

► **Lemma 25.** *Given a turn-based game \mathcal{G} , an initial vertex v , and an objective φ , there is a bidding game with charging \mathcal{G}' of size linear in \mathcal{G} , with the same objective and initial vertex such that Player 1 can win \mathcal{G} from v if and only if $\langle \mathcal{G}, v, \varphi \rangle \in \text{THRESH}_S^{\text{Richman}}$.*

Proof. The definitions of turn-based games and the detailed proof can be found in the extended version [13]. Intuitively, \mathcal{G}' contains the same set of vertices as \mathcal{G} with two additional sink vertices s_1 and s_2 , where s_i is losing for Player i , for $i \in \{1, 2\}$. For every vertex v , if v is controlled by Player 1 in \mathcal{G} , then in \mathcal{G}' , we define Player 1's charge to be $R_1(v) = 2$. Moreover, we add an edge from v to s_1 , requiring Player 1 to win the bidding in v . Note that even if Player 1 starts with a budget of $\epsilon > 0$, at v , after charging and



■ **Figure 5** Illustrating the repair problem. LEFT: A reachability game with the objective $\text{Reach}(\{g\})$. RIGHT: With no repair (first row), $Th_1(a) = 1$. We depict repairs (first col.) and the changes they imply to the thresholds (second col.), for a repair budget of $C = 2$.

normalization, his budget exceeds $2/3$. Player 2's vertices are dual. It is not hard to verify that Player 1 can win in \mathcal{G} from v if and only if $Th^\varphi(v) = 0$, and Player 2 can win in \mathcal{G} from v if and only if $Th^\varphi(v) = 1$. ◀

Since turn-based Rabin games are NP-hard, we obtain the following.

▶ **Theorem 26.** We have $\text{THRESH}_{\text{Rabin}}^{\text{Richman}} \in \text{NP-hard}$ and $\text{THRESH}_{\text{Streett}}^{\text{Richman}} \in \text{coNP-hard}$.

6 Repairing Bidding Games

In this section, we introduce the *repair* problem for bidding games. Intuitively, the goal is to add minimal charges to the vertices of an arena so as to decrease the threshold in the initial vertex to a target threshold. Formally, we define the following problem.

▶ **Definition 27** (Repairing bidding games). Consider an arena \mathcal{G} with a vertex v , a bidding mechanism $M \in \{\text{Richman}, \text{poorman}, \text{taxman}\}$, a class of objectives $S \in \{\text{Reach}, \text{Safe}, \text{Büchi}, \text{Co-Büchi}\}$, and a repair budget $C \in \mathbb{R}_{\geq 0}$. The set of repaired arenas, denoted $\text{Repaired}(\mathcal{G}, C)$, are arenas obtained from \mathcal{G} by adding Player 1 charges whose sum does not exceed C . Formally, $\text{Repaired}(\mathcal{G}, C) := \{\langle V, E, R'_1, R_2 \rangle \text{ is an arena} \mid \forall v \in V. R'_1(v) \geq R_1(v) \wedge \sum_{v \in V} (R'_1(v) - R_1(v)) \leq C\}$. The problem R_THRESH_S^M takes as input $\langle \mathcal{G}, v, \varphi, C \rangle$, where $\varphi \in S$, and accepts iff there exists $\mathcal{G}' \in \text{Repaired}(\mathcal{G}, C)$ with $\langle \mathcal{G}', v, \varphi \rangle \in \text{THRESH}_S^M$.

▶ **Example 28.** We illustrate the non-triviality of the repair problem in Fig. 5. Observe that neither assigning charges uniformly nor assigning charges to a single vertex, decrease the threshold sufficiently, whereas adding a charge of 1 to both b and d is a successful repair. ◻

▶ **Theorem 29.** The following hold:

- (i) $\text{R_THRESH}_{\text{Reach}}^{\text{Richman}} \in 2\text{EXPTIME}$,
- (ii) $\text{R_THRESH}_{\text{Safe}}^{\text{Richman}} \in \text{PSPACE}$.

Proof. We introduce notation for the proof. Let $G = \langle V, E, R_1, R_2 \rangle$ be a bidding game and U be a set of vertices. Define $A_U^G: [0, 1]^V \rightarrow \{0, 1\}$ such that for every $h \in [0, 1]^V$, $A_U^G(h) = 1$ iff $h(v) = Av_1(h)(v)$ for every $v \notin U$. Observe that $Th_1^{\text{Reach}(T)}$ is the largest h for which $A_T^G(h) = 1$ and moreover $h(v) = 0$ for every $v \in T$.

Proof of (i). Consider a bidding game $G = \langle V, E, R_1, R_2 \rangle$ where the objective of Player 1 is $\text{Reach}(T)$ for some $T \subseteq V$. The goal is to check if it is possible to increase R_1 by a total of C such that the reachability threshold at $a \in V$ falls below 0.5. This is equivalent to:

$$\exists R'_1 \in \mathbb{R}^V . (R'_1 \geq R_1) \wedge (|R'_1 - R_1|_1 \leq C) \wedge \\ \left(\forall Th_1 \in \mathbb{R}^V . \left[(\forall v \in T . Th_1(v) = 0) \wedge A_T^{\mathcal{G}'}(Th_1) \right] \Rightarrow Th_1(a) \leq 0.5 \right)$$

where $\mathcal{G}' = \langle V, E, R'_1, R_2 \rangle$. The validity of the above formula can be checked by applying a quantifier elimination method. Therefore, the decision problem is in 2EXPTIME.

Proof of (ii). Consider a bidding game $G = \langle V, E, R_1, R_2 \rangle$ where the objective of Player 1 is $\text{Safe}(T)$ for some $T \subseteq V$. The goal is to check if it is possible to increase R_1 by a total of C such that the safety threshold at $a \in V$ falls below 0.5. This is equivalent to:

$$\exists R'_1 \in \mathbb{R}^V . (R'_1 \geq R_1) \wedge (|R'_1 - R_1|_1 \leq C) \wedge \\ \left(\exists Th_1 \in \mathbb{R}^V \text{ s.t. } [Th_1(a) \leq 0.5 \wedge A_T^{\mathcal{G}'}(Th_1) \wedge \forall u \in T, Th_1(u) = 1] \right)$$

where $\mathcal{G}' = \langle V, E, R'_1, R_2 \rangle$. The above formula can be seen as an input instance of existential theory of reals which is known to be in PSPACE. ◀

7 Conclusion and Future Work

We introduce and study a generalization of bidding games in which players' budgets are charged throughout the game. One advantage of the model over traditional bidding games is that long-run safety is not trivial. We show that the model maintains the key favorable property of traditional bidding games, namely the existence of thresholds, the proof of which is, however, significantly more challenging due to the non-uniqueness of thresholds. We characterize thresholds in terms of greatest and least fixed points of certain monotonic operators. Finally, we establish the first complexity lower bounds in continuous-bidding games and study, for the first time, a repair problem in this model.

There are plenty of open questions and directions for future research. First, it is important to extend the results to richer classes of ω -regular objectives, like parity, Rabin, and Streett, as well as to quantitative objectives, like mean-payoff. Second, tightening the complexity bounds is an important open question. For example, it might be the case that finding thresholds in Richman-bidding games with charging is in NP and coNP. Third, traditional reachability Richman-bidding games are equivalent to a class of stochastic games [18] called *random-turn games* [24], and the equivalence is general and intricate in infinite-duration games [4, 5, 7, 9]. It is unknown if such a connection exists for games with charging, and if it does, then many of the open questions may be solved via available tools for stochastic games. Finally, there are various possible extensions, like charges disappearing after a vertex is visited, charges that are collectible in multiple installments, etc.

References



- 1 M. Aghajohari, G. Avni, and T. A. Henzinger. Determinacy in discrete-bidding infinite-duration games. *Log. Methods Comput. Sci.*, 17(1), 2021.
- 2 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
- 3 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-duration bidding games. In *Proc. 28th CONCUR*, volume 85 of *LIPICs*, pages 21:1–21:18, 2017.
- 4 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-duration bidding games. *J. ACM*, 66(4):31:1–31:29, 2019.
- 5 G. Avni, T. A. Henzinger, and R. Ibsen-Jensen. Infinite-duration poorman-bidding games. In *Proc. 14th WINE*, volume 11316 of *LNCS*, pages 21–36. Springer, 2018.

- 6 G. Avni, T. A. Henzinger, and Đ. Žikelić. Bidding mechanisms in graph games. In *Proc. 44th MFCS*, volume 138 of *LIPICs*, pages 11:1–11:13, 2019.
- 7 G. Avni, T. A. Henzinger, and D. Zikelic. Bidding mechanisms in graph games. *J. Comput. Syst. Sci.*, 119:133–144, 2021.
- 8 G. Avni, R. Ibsen-Jensen, and J. Tkadlec. All-pay bidding games on graphs. In *Proc. 34th AAAI*, pages 1798–1805. AAAI Press, 2020.
- 9 G. Avni, I. Jecker, and Đ. Žikelić. Infinite-duration all-pay bidding games. In *Proc. 32nd SODA*, pages 617–636, 2021.
- 10 G. Avni, I. Jecker, and D. Zikelic. Bidding graph games with partially-observable budgets. In *Proc. 37th AAAI*, 2023.
- 11 G. Avni, T. Meggendorfer, S. Sadhukhan, J. Tkadlec, and Đ. Zikelic. Reachability poorman discrete-bidding games. In *Proc. 26th ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 141–148. IOS Press, 2023.
- 12 G. Avni and S. Sadhukhan. Computing threshold budgets in discrete-bidding games. In *Proc. 42nd FSTTCS*, volume 250 of *LIPICs*, pages 30:1–30:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 13 Guy Avni, Ehsan Kafshdar Goharshady, Thomas A. Henzinger, and Kaushik Mallik. Bidding games with charging, 2024. [arXiv:2407.06288](https://arxiv.org/abs/2407.06288).
- 14 Guy Avni, Kaushik Mallik, and Suman Sadhukhan. Auction-based scheduling. In *TACAS (3)*, volume 14572 of *Lecture Notes in Computer Science*, pages 153–172. Springer, 2024.
- 15 František Blahoudek, Petr Novotný, Melkior Ornik, Pranay Thangeda, and Ufuk Topcu. Efficient strategy synthesis for mdps with resource constraints. *IEEE Transactions on Automatic Control*, 2022.
- 16 Patricia Bouyer, Uli Fahrenberg, Kim G Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In *Formal Modeling and Analysis of Timed Systems: 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings 6*, pages 33–47. Springer, 2008.
- 17 Arindam Chakrabarti, Luca De Alfaro, Thomas A Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *International Workshop on Embedded Software*, pages 117–133. Springer, 2003.
- 18 A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.
- 19 M. Develin and S. Payne. Discrete bidding games. *The Electronic Journal of Combinatorics*, 17(1):R85, 2010.
- 20 Robert G Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical programming*, 32(2):146–164, 1985.
- 21 A. J. Lazarus, D. E. Loeb, J. G. Propp, W. R. Stromquist, and D. H. Ullman. Combinatorial games under auction play. *Games and Economic Behavior*, 27(2):229–264, 1999.
- 22 A. J. Lazarus, D. E. Loeb, J. G. Propp, and D. Ullman. Richman games. *Games of No Chance*, 29:439–449, 1996.
- 23 R. Meir, G. Kalai, and M. Tennenholtz. Bidding games and efficient allocations. *Games and Economic Behavior*, 112:166–193, 2018. [doi:10.1016/j.geb.2018.08.005](https://doi.org/10.1016/j.geb.2018.08.005).
- 24 Y. Peres, O. Schramm, S. Sheffield, and D. B. Wilson. Tug-of-war and the infinity laplacian. *J. Amer. Math. Soc.*, 22:167–210, 2009.
- 25 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
- 26 M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.
- 27 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

Risk-Averse Optimization of Total Rewards in Markovian Models Using Deviation Measures

Christel Baier  

Technische Universität Dresden, Germany

Jakob Piribauer  

Technische Universität Dresden, Germany; Universität Leipzig, Germany

Maximilian Starke

Technische Universität Dresden, Germany

Abstract

This paper addresses objectives tailored to the *risk-averse* optimization of accumulated rewards in Markov decision processes (MDPs). The studied objectives require maximizing the expected value of the accumulated rewards minus a penalty factor times a deviation measure of the resulting distribution of rewards. Using the variance in this penalty mechanism leads to the variance-penalized expectation (VPE) for which it is known that optimal schedulers have to minimize future expected rewards when a high amount of rewards has been accumulated. This behavior is undesirable as risk-averse behavior should keep the probability of particularly low outcomes low, but not discourage the accumulation of additional rewards on already good executions.

The paper investigates the semi-variance, which only takes outcomes below the expected value into account, the mean absolute deviation (MAD), and the semi-MAD as alternative deviation measures. Furthermore, a penalty mechanism that penalizes outcomes below a fixed threshold is studied. For all of these objectives, the properties of optimal schedulers are specified and in particular the question whether these objectives overcome the problem observed for the VPE is answered. Further, the resulting algorithmic problems on MDPs and Markov chains are investigated.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Markov decision processes, risk-aversion, deviation measures, total reward

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.9

Related Version *Full Version*: <https://arxiv.org/abs/2407.06887> [6]

Supplementary Material *Software (Source Code)*:

<https://github.com/experiments-collection/risk-averse-stochastic-shortest-paths>
archived at `swh:1:dir:eddcf497fe105ca58ea2b4f67171e814a6d35f29`

Funding This work was partly funded by the DFG Grant 389792660 as part of TRR 248 (Foundations of Pervasive Software Systems), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), and the DFG project BA 1679/11-1.

1 Introduction

Markov decision processes (MDPs) are a prominent model for systems whose behavior is subject to *non-determinism* and *probabilism*. Non-deterministic behavior might arise, e.g., if a system is employed in an unknown environment, can be controlled by a user, or works concurrently. On the other hand, if, e.g., sufficiently much data on the failure of components is available or randomized algorithms make use of randomization explicitly, it is reasonable to model these aspects of the system as probabilistic.

In order to model quantitative aspects of a system, such as energy consumption, execution time, or utility, MDPs are often equipped with a *reward function* that specifies how much reward is received in each step of an execution. A typical task is then to resolve the non-deterministic choices by specifying a *scheduler*, a.k.a. *policy*, such that the expected value of



© Christel Baier, Jakob Piribauer, and Maximilian Starke;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 9; pp. 9:1–9:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the total accumulated reward is maximal (or minimal). In verification, such optimization problems naturally occur when investigating the worst- or best-case expected value of the accumulated reward where worst- and best-case range over all resolutions of the non-deterministic choices. If additionally a target state has to be reached almost surely, this problem is known as the *stochastic shortest path problem* [7, 12].

Risk-averse optimization

If the objective is the maximization of the expected value of the accumulated rewards, all other aspects of the probability distribution of accumulated rewards are disregarded. This might lead to undesirable behavior as the optimal scheduler might receive low rewards with high probability as long as the expected value is optimal. In many situations, however, a slightly lower expected reward is preferable if it is obtained by a more “stable” behavior in which the risk of encountering low rewards is reduced. E.g., in a traffic control scenarios, it might be important to reduce the risk of congestions while ensuring a reasonable average throughput instead of solely optimizing the average throughput.

In order to define objectives incentivizing such risk-averse behavior, it is worth taking a look at finance and in particular portfolio optimization. Here, Markowitz proclaimed that a portfolio of financial positions should be chosen such that it is Pareto optimal with respect to the expected return and the variance of the return [23]. One way extensively studied in finance to obtain Pareto optimal portfolios is to maximize the *variance-penalized expectation (VPE)*, which is the expected value minus a penalty factor λ times the variance. The parameter λ can be used to obtain different levels of risk-aversion.

Besides the variance, further deviation measures have been investigated to reduce risk in portfolio optimization: The use of the *semi-variance*, which – in contrast to the variance – only takes the deviation of outcomes below the expected value into account, as a penalty mechanism has been introduced in this context by Markowitz [24]. Furthermore, instead of considering quadratic deviations from the expected value as in the case of variance and semi-variance, the *mean absolute deviation (MAD)* can be used to obtain the MAD-penalized expectation (MADPE) studied for portfolio optimization in [18]. The MAD measures the expected absolute deviation from the expected value.

In this paper, we investigate these different deviation measure based penalty mechanisms in the context of the maximization of rewards in MDPs.

Variance-penalized expectation in MDPs (VPE)

Recently, the maximization of the VPE of accumulated rewards in MDPs was studied in [25]: On the positive side, it is shown that optimal schedulers for the VPE can be chosen to be deterministic finite-memory schedulers. Nevertheless, the optimization of the VPE is shown to be computationally hard: The threshold problem whether the optimal VPE exceeds a given threshold ϑ is EXPTIME-hard. An optimal scheduler can be computed in exponential space.

A main drawback of the VPE, however, is of conceptual nature: In [25], it is shown that VPE-optimal schedulers have to *minimize* the future expected rewards as soon as a high amount of rewards (above a computable bound B) has been accumulated. We call such schedulers *eventually reward-minimizing schedulers (ERMin-schedulers)*. Intuitively, the reason is that a further accumulation of additional rewards after a high amount of rewards has already been accumulated has a stronger effect on the variance than on the expected value due to the quadratic nature of the variance. Conceptually, this can be considered to be a flaw in the use of the VPE as an objective to yield risk-averse behavior.

■ **Table 1** Overview of the complexity results and the types of schedulers needed for the optimization of the studied objectives and the VPE. The entries “-” indicate that the problem was not studied further as the scheduler needed for the optimization are the undersirable ERMin-schedulers.

	hardness of threshold problem	computation of optimum	optimal schedulers
VPE [25]	EXPTIME-hard; in P for Markov chains	in exponential space	deterministic, finite-memory ERMin-schedulers
SVPE	–	–	randomized, ERMin-schedulers can be necessary
MADPE ($\lambda \leq 1/2$), SMADPE ($\lambda \leq 1$)	PP-hard for acyclic Markov chains	quadratic program of exponential size	randomized, finite-memory ERMax-schedulers
MADPE ($\lambda > 1/2$), SMADPE ($\lambda > 1$)	–	–	randomized, ERMin-schedulers can be necessary
TBPE	PP-hard for acyclic Markov chains	in pseudo-polynomial time	deterministic, finite-memory ERMax-schedulers

The desired behaviour a suitable objective should induce is that a scheduler achieves a high expected accumulated reward, while keeping the probability of particularly bad outcomes low. Improving on already good outcomes should not have a negative effect. So, we want optimal schedulers to be *eventually reward-maximizing (ERMax-schedulers)*, i.e., that they maximize the expected reward once the accumulated reward exceeds some bound B .

Deviation-measure-penalized expectation

Towards this goal, we investigate objectives in the spirit of the VPE, which are of the form $\mathbb{E}^{\mathfrak{S}}(\text{rew}) - \lambda \mathbb{DEV}^{\mathfrak{S}}(\text{rew})$ where a penalty factor λ times a deviation measure $\mathbb{DEV}^{\mathfrak{S}}(\text{rew})$ of the probability distribution of accumulated rewards under a scheduler \mathfrak{S} is subtracted from the expected accumulated reward $\mathbb{E}^{\mathfrak{S}}(\text{rew})$.

The first deviation measure we investigate is the MAD. In contrast to the variance, the contribution of an outcome to the MAD only grows linearly with its distance to the expected value. For the MAD and the variance, we also study one-sided variants in which only outcomes below the expected value are considered: The semi-MAD (SMAD) and semi-variance quantify the average absolute or squared deviation below the expected value by assigning deviation 0 to all outcomes above the expected value. Finally, we investigate a simpler alternative to the MADPE: Instead of measuring the deviation from the expected value of accumulated rewards, which itself depends on the chosen scheduler, we consider a threshold-based penalized expectation (TBPE), where outcomes below a threshold t that can be chosen externally are penalized either linearly or according to more complicated functions.

Contributions

The main contributions, also summarized in Table 1, are as follows.

- We show that optimal schedulers for the MADPE can be chosen to be ERMax-schedulers, as desired, if the risk-aversion parameter λ is sufficiently small, i.e. if $\lambda \leq 1/2$. This bound on the parameter is shown to be tight. Furthermore, we show that randomized schedulers are necessary for the optimization.

We formulate the optimization problem as a quadratic program and obtain a EXPSPACE-upper complexity bound for the threshold problem for the MADPE. On the other hand, we show that already in acyclic Markov chains the threshold problems for the MADPE and the MAD are PP-hard under polynomial-time Turing reductions.

As the semi-MAD is always half of the MAD, the results transfer to the semi-MADPE.

- We investigate the semivariance-penalized expectation (SVPE) and show – somewhat surprisingly – that, for any risk-aversion parameter λ , there are MDPs in which optimal schedulers are ERMin-schedulers. Hence, the SVPE as objective does not overcome the undesirable effects observed for the VPE. Furthermore, we show that, in contrast to the VPE, randomization is necessary for the optimization of the SVPE.
- We show that the TBPE can be optimized in pseudo-polynomial time and that deciding if the TBPE exceeds a bound for linear penalty functions even in acyclic Markov chains is PP-hard under polynomial-time Turing reductions.

As a proof-of-concept, we analyze our algorithms for the optimization of the MADPE and for the TBPE in a small series of experiments.

Related work

The above mentioned work on the VPE for accumulated rewards in MDPs [25] is the closest related work to our paper. Earlier work on the VPE in MDPs addressed the finite-horizon setting with terminal rewards [11] or applied the notion to mean payoff and discounted rewards [13]. Further, [31] presents a policy iteration algorithm converging against *local* optima for a similar measure. The computation of the variance of accumulated rewards has been studied in Markov chains [30] and in MDPs [21, 22]. In [8], the satisfiability of constraints on the expected mean payoff in conjunction with constraints on the variance or related notions such as a local variability are studied for MDPs.

For MDPs, the SVPE of random variables defined in terms of the stationary distribution has been studied via the use of reinforcement learning algorithms [20]. Conceptually and methodologically this work is nevertheless not closely related to our work. We are not aware of investigations of the MADPE on MDPs.

Furthermore, several approaches to formalize various other risk-averse optimization problems for accumulated rewards in MDPs have been proposed and studied in the literature. This includes the computation of worst- or best-case quantiles [29, 4, 16, 27], also called *values-at-risk*: Given a probability p , quantiles on the accumulated rewards are the best bound C such that the accumulated rewards stays below C with probability at most p under all or under some scheduler. While quantiles still disregard the distribution below, the *conditional value-at-risk* and the *entropic value-at-risk* are more involved measures that quantify how far the probability mass of the tail of the probability distribution lies below a given quantile. In the context of risk-averse optimization in MDPs, these measures have been studied in [19] and [1]. A further approach, the *entropic risk* measure, reweighs outcomes by an exponential utility function. Optimizing this entropic risk measure leads to schedulers that tend to still achieve a high expected value while keeping the probability of low outcomes small. The entropic risk measure applied to accumulated rewards have been studied in [3] for stochastic games that extend MDPs with an adversarial player.

2 Preliminaries

Notations for Markov decision processes

A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, Act, P, s_{init}, rew)$ where S is a finite set of states, Act a finite set of actions, $P: S \times Act \times S \rightarrow [0, 1] \cap \mathbb{Q}$ the transition probability function, $s_{init} \in S$ the initial state, and $rew: S \times Act \rightarrow \mathbb{N}$ the reward function. Note that we only allow non-negative rewards and that rational rewards can be transformed to integral rewards by multiplying all rewards with the least common multiple of all denominators of the rational rewards. We require that $\sum_{t \in S} P(s, \alpha, t) \in \{0, 1\}$ for all $(s, \alpha) \in S \times Act$. We say that action α is enabled in state s iff $\sum_{t \in S} P(s, \alpha, t) = 1$ and denote the set of all actions that are enabled in state s by $Act(s)$. If $Act(s) = \emptyset$, we say that s is a *trap* state. The paths of \mathcal{M} are finite or infinite sequences $s_0 \alpha_0 s_1 \alpha_1 \dots$ where states and actions alternate such that $P(s_i, \alpha_i, s_{i+1}) > 0$ for all $i \geq 0$. For $\pi = s_0 \alpha_0 s_1 \alpha_1 \dots \alpha_{k-1} s_k$, $rew(\pi) = rew(s_0, \alpha_0) + \dots + rew(s_{k-1}, \alpha_{k-1})$ – and analogously for infinite paths – denotes the accumulated reward of π , $P(\pi) = P(s_0, \alpha_0, s_1) \cdot \dots \cdot P(s_{k-1}, \alpha_{k-1}, s_k)$ its probability, and $last(\pi) = s_k$ its last state. A path is called *maximal* if it is infinite or ends in the trap state *goal*. The *size* of \mathcal{M} is the sum of the number of states plus the total sum of the logarithmic lengths of the non-zero probability values $P(s, \alpha, s')$ as fractions of co-prime integers and the weight values $rew(s, \alpha)$.

A *Markov chain* is an MDP in which the set of actions is a singleton. In this case, we can drop the set of actions and consider a Markov chain as a tuple $\mathcal{M} = (S, P, s_{init}, rew)$ where P now is a function from $S \times S$ to $[0, 1]$ and rew a function from S to \mathbb{N} .

An *end component* of \mathcal{M} is a strongly connected sub-MDP formalized by a subset $S' \subseteq S$ of states and a non-empty subset $\mathfrak{A}(s) \subseteq Act(s)$ for each state $s \in S'$ such that for each $s \in S'$, $t \in S$ and $\alpha \in \mathfrak{A}(s)$ with $P(s, \alpha, t) > 0$, we have $t \in S'$ and such that in the resulting sub-MDP all states are reachable from each other. An end-component is a 0-end-component if it only contains state-action-pairs with reward 0.

Scheduler

A *scheduler* for \mathcal{M} is a function \mathfrak{S} that assigns to each non-maximal path π a probability distribution over $Act(last(\pi))$. If the choice of a scheduler \mathfrak{S} depends only on the current state, i.e., if $\mathfrak{S}(\pi) = \mathfrak{S}(\pi')$ for all non-maximal paths π and π' with $last(\pi) = last(\pi')$, we say that \mathfrak{S} is *memoryless* and also view it as functions mapping states $s \in S$ to probability distributions over $Act(s)$. A scheduler \mathfrak{S} that satisfies $\mathfrak{S}(\pi) = \mathfrak{S}(\pi')$ for all pairs of finite paths π and π' with $last(\pi) = last(\pi')$ and $rew(\pi) = rew(\pi')$ is called *reward-based* and can be viewed as a function from state-reward pairs $S \times \mathbb{N}$ to probability distributions over actions. If there is a finite set X of memory modes and a memory update function $U: S \times Act \times S \times X \rightarrow X$ such that the choice of \mathfrak{S} only depends on the current state after a finite path and the memory mode obtained from updating the memory mode according to U in each step, we say that \mathfrak{S} is a *finite-memory scheduler*. A scheduler \mathfrak{S} is called *deterministic* if $\mathfrak{S}(\pi)$ is a Dirac distribution for each path π in which case we also view the scheduler as a mapping to actions in $Act(last(\pi))$.

Probability measure

We write $\Pr_{\mathcal{M}, s}^{\mathfrak{S}}$ to denote the probability measure induced by a scheduler \mathfrak{S} and a state s of an MDP \mathcal{M} . It is defined on the σ -algebra generated by the cylinder sets $Cyl(\pi)$ of all maximal extensions of a finite path $\pi = s_0 \alpha_0 s_1 \alpha_1 \dots \alpha_{k-1} s_k$ with $s_0 = s$ by assigning

to $Cyl(\pi)$ the probability that π is realized under \mathfrak{S} , which is $\mathfrak{S}(s_0)(\alpha_0) \cdot P(s_0, \alpha_0, s_1) \cdot \dots \cdot \mathfrak{S}(s_0 \alpha_0 \dots s_{k-1})(\alpha_{k-1}) \cdot P(s_{k-1}, \alpha_{k-1}, s_k)$. For a set of states T , we use $\diamond T$ to denote the event that a state in T is reached. For details, see [26].

For a random variable X that is defined on (some of the) maximal paths in \mathcal{M} , we denote the expected value of X under the probability measure induced by a scheduler \mathfrak{S} and state s by $\mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}(X)$. We define $\mathbb{E}_{\mathcal{M},s}^{\min}(X) = \inf_{\mathfrak{S}} \mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}(X)$ and $\mathbb{E}_{\mathcal{M},s}^{\max}(X) = \sup_{\mathfrak{S}} \mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}(X)$ where \mathfrak{S} ranges over all schedulers for \mathcal{M} under which X is defined almost surely. The variance of X under the probability measure determined by \mathfrak{S} and s in \mathcal{M} is denoted by $\mathbb{V}_{\mathcal{M},s}^{\mathfrak{S}}(X)$ and defined by $\mathbb{V}_{\mathcal{M},s}^{\mathfrak{S}}(X) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}((X - \mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}(X))^2) = \mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}(X^2) - \mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}(X)^2$. Furthermore, for a measurable set of paths ψ with positive probability, $\mathbb{E}_{\mathcal{M},s}^{\mathfrak{S}}(X|\psi)$ denotes the conditional expectation of X under ψ . If $s = s_{init}$, we sometimes drop the subscript s .

Accumulated rewards

Given an MDP $\mathcal{M} = (S, Act, P, s_{init}, rew)$, the total accumulated reward is given by the extension of the function rew to maximal paths. We can check whether $\mathbb{E}_{\mathcal{M}}^{\max}(rew) = \infty$ by checking whether all (maximal) end components are 0-end components in polynomial time [12]. For our purposes, only MDPs \mathcal{M} with $\mathbb{E}_{\mathcal{M}}^{\max}(rew) < \infty$ are interesting. In these MDPs, we can collapse all end components \mathcal{E} , which are all 0-end components, to single states $s_{\mathcal{E}}$ while adding a transition with reward 0 to a new trap state. This does not affect the possible distributions of the random variable rew that can be realized by a scheduler [12]. Furthermore, the behavior of the MDP starting from a state s with $\mathbb{E}_{\mathcal{M},s}^{\max}(rew) = 0$, i.e., from a state s from which no positive reward is reachable, is irrelevant. So, we can collapse all these states s with $\mathbb{E}_{\mathcal{M},s}^{\max}(rew) = 0$ (together with the new trap state) to a single trap state that we call $goal$. By these constructions, we obtain a new MDP \mathcal{M}' in which exactly the same distributions of the total reward can be realized by schedulers as in \mathcal{M} . As \mathcal{M}' does not contain any end components anymore and $goal$ is the only trap state in \mathcal{M}' , the state $goal$ is now reached with probability 1 under any scheduler. In the light of the described constructions, we work under the following assumption:

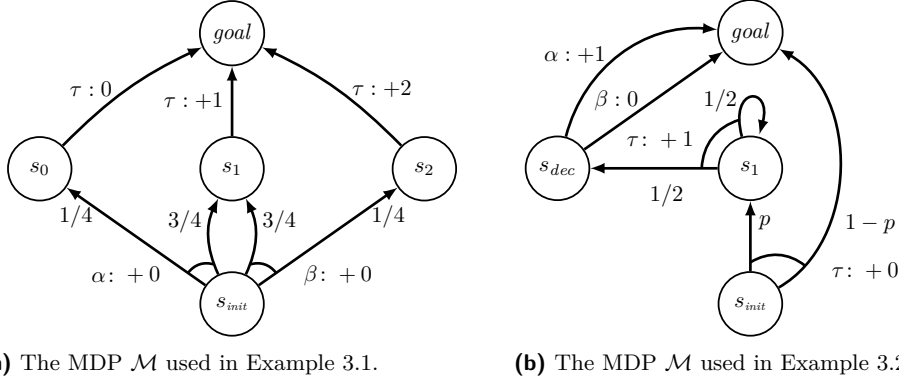
► **Assumption 1.** *W.l.o.g., we assume that all MDPs have a trap state $goal$, which is reached with probability 1 under all schedulers. We add this trap state to the signature and hence denote MDPs \mathcal{M} as tuples $\mathcal{M} = (S, Act, P, s_{init}, rew, goal)$.*

All objectives studied in this paper depend only on the distribution of the random variable rew . By the following lemma, which is folklore and follows from the formulation in [25, Lemma 2] (see also the full version [6]), we can restrict ourselves to reward-based schedulers.

► **Lemma 2.1.** *Let $\mathcal{M} = (S, Act, P, s_{init}, rew, goal)$ be an MDP satisfying Assumption 1. Then, for any scheduler \mathfrak{S} there is a reward-based scheduler \mathfrak{T} such that the distribution of the random variable rew is the same under the probability measures $\Pr_{\mathcal{M}}^{\mathfrak{S}}$ and $\Pr_{\mathcal{M}}^{\mathfrak{T}}$.*

3 Mean absolute deviation-penalized expectation

As described in the introduction, the VPE suffers from the drawback that optimal schedulers are ERMin-schedulers, which is an undesirable behavior. Intuitively, the reason for this behavior in the case of VPE lies in the fact that the variance grows quadratically with the distance to the expected value. A natural alternative is choosing the absolute distance rather than the quadratic distance from the expected value as the measure for the penalty. So, we define the *mean absolute deviation* (MAD) of a random variable X as the probability-weighted sum of the distance to the expected value: $\text{MAD}(X) \stackrel{\text{def}}{=} \mathbb{E}(|X - \mathbb{E}(X)|)$.

(a) The MDP \mathcal{M} used in Example 3.1.(b) The MDP \mathcal{M} used in Example 3.2.

■ **Figure 1** Two example MDPs.

We consider the MAD-penalized expectation (MADPE) of the accumulated weight in an MDP $\mathcal{M} = (S, Act, P, s_{init}, rew, goal)$ analogously to the VPE: We define the MAD of the accumulated reward rew under scheduler \mathfrak{S} as $\text{MAD}_{\mathcal{M}}^{\mathfrak{S}}(rew) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(|rew - \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(rew)|)$. The MAD-penalized expectation with parameter $\lambda \in \mathbb{R}$ is now $\text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}}(rew) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(rew) - \lambda \text{MAD}_{\mathcal{M}}^{\mathfrak{S}}(rew)$ analogously to the VPE. Our goal is to find

$$\text{MADPE}[\lambda]_{\mathcal{M}}^{\max}(rew) \stackrel{\text{def}}{=} \sup_{\mathfrak{S}} \text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}}(rew)$$

as well as an optimal scheduler. In the sequel, we will prove the following results. Omitted proofs can be found in [6].

1. In general, randomization is necessary to optimize the MADPE.
2. If $\lambda > \frac{1}{2}$, then there is an MDP \mathcal{M} such that any optimal scheduler for the MADPE is an ERMin-scheduler.
3. If $\lambda \leq \frac{1}{2}$, for any MDP \mathcal{M} , optimal schedulers can be chosen to be reward-based ERMax-schedulers.
4. If $\lambda \leq \frac{1}{2}$, the optimal MADPE can be computed in exponential time.
5. Even for acyclic Markov chains, deciding whether the MADPE exceeds a given threshold ϑ is PP-hard under polynomial-time Turing reductions.

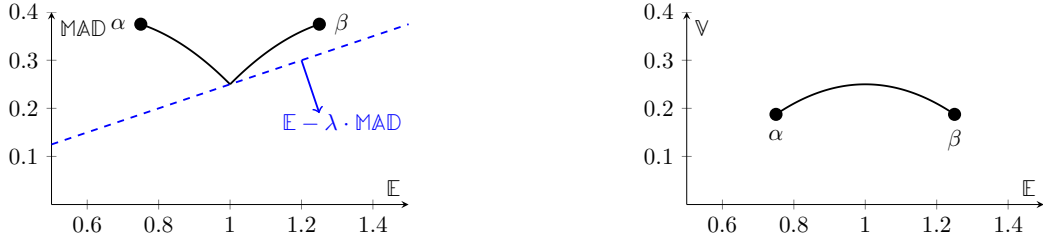
3.1 Randomization and optimality of ERMin-schedulers

We work with MDPs $\mathcal{M} = (S, Act, P, s_{init}, rew, goal)$ satisfying Assumption 1. First, we show that randomization is necessary for the optimization of the MADPE in the following example.

► **Example 3.1.** Consider the MDP \mathcal{M} in Figure 1a. We consider the schedulers \mathfrak{S}_{α} choosing α in s_{init} , \mathfrak{S}_{β} choosing β , and $\mathfrak{S}_{1/2}$ choosing α and β with probability $1/2$ each and obtain: $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_{\alpha}}(rew) = 3/4$, $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_{1/2}}(rew) = 1$, and $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_{\beta}}(rew) = 5/4$. The MADs are $\text{MAD}_{\mathcal{M}}^{\mathfrak{S}_{\alpha}}(rew) = 3/8$, $\text{MAD}_{\mathcal{M}}^{\mathfrak{S}_{1/2}}(rew) = 1/4 \cdot 1 = 1/4$, and $\text{MAD}_{\mathcal{M}}^{\mathfrak{S}_{\beta}}(rew) = 3/8$. Clearly, the MADPE under \mathfrak{S}_{β} is better than under \mathfrak{S}_{α} for any $\lambda > 0$. For the MADPE of $\mathfrak{S}_{1/2}$ and \mathfrak{S}_{β} with $\lambda = 4$, we obtain

$$\text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}_{1/2}}(rew) = 1 - \frac{1}{4}\lambda = 0, \quad \text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}_{\beta}}(rew) = \frac{5}{4} - \frac{3}{8}\lambda = -\frac{1}{4}.$$

So, the randomized scheduler $\mathfrak{S}_{1/2}$ is better than the deterministic schedulers \mathfrak{S}_{α} and \mathfrak{S}_{β} . In Figure 2, we depict the MAD in comparison to the expected value of any randomized scheduler for \mathcal{M} . The kink in the graph at expected value 1 can be explained by the fact



■ **Figure 2** Plot of MAD and variance over the expected value for schedulers obtained by choosing α with probability $p \in [0, 1]$ in the MDP \mathcal{M} depicted in Figure 1a.

that the MAD contains a summand for $|1 - \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew})|$. The dotted blue line consists of all points in the MAD-E-plane with the same MADPE as the scheduler $\mathfrak{S}_{1/2}$ illustrating that this scheduler is in fact optimal as the MADPE increases in the direction of the arrow. For comparison, we also depict the variances of randomized schedulers over the expectation. Clearly, for any λ the deterministic scheduler choosing β will always be VPE-optimal.

In the next example, we will illustrate that the MADPE fails to guarantee in general that optimal schedulers are eventually reward-maximizing.

► **Example 3.2.** Consider the MDP \mathcal{M} depicted in Figure 1b for $p \in (0, 1/3]$. Always choosing α in state s_{dec} maximizes the expected value. Under this scheduler, the expected value is $3p \leq 1$ as moving from state s_1 to state s_{dec} takes two steps in expectation. So, under any scheduler, the expected value lies between 0 and 1. So, all paths leading via s_1 yield a reward above the expected value, while only the path going directly to *goal* from s_{init} yields a reward below the expected value. For the MAD under a scheduler \mathfrak{S} , we obtain $\text{MAD}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) = 2 \cdot (1 - p) \cdot \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew})$ (see the full version [6] for the calculations).

For a given $\lambda > \frac{1}{2}$, we can choose $p \in (0, 1/3]$ such that $\lambda > \frac{1}{2(1-p)}$ and hence $\lambda \cdot 2 \cdot (1-p) > 1$. Now, under any scheduler \mathfrak{S} , the MADPE for parameter λ is

$$\text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) = \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) - \lambda \cdot 2 \cdot (1-p) \cdot \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) = (1 - \lambda \cdot 2 \cdot (1-p)) \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}).$$

As $1 - \lambda \cdot 2 \cdot (1-p) < 0$, a scheduler maximizing the MADPE has to minimize the expected value of *rew*. In \mathcal{M}_p , this means always choosing β . So, for any $\lambda > \frac{1}{2}$, there is an MDP in which optimal schedulers have to minimize the future expected rewards no matter how large the accumulated reward already is.

3.2 Sufficiently small parameters λ

As we have seen, the MADPE as an objective does not in general guarantee that optimal schedulers are ERMax-schedulers. In this section, we now show that this desirable property is guaranteed if the risk-aversion parameter λ is at most $\frac{1}{2}$.

By Lemma 2.1, we already know that we can restrict ourselves to reward-based schedulers when optimizing the MADPE. For two reward-based schedulers \mathfrak{S} and \mathfrak{T} and a natural number k , we define the reward-based scheduler $\mathfrak{S} \uparrow_k \mathfrak{T}$ on state-reward-pairs $(s, w) \in S \times \mathbb{N}$ by $(\mathfrak{S} \uparrow_k \mathfrak{T})(s, w) = \begin{cases} \mathfrak{S}(s, w) & \text{if } w < k, \\ \mathfrak{T}(s, w) & \text{if } w \geq k \end{cases}$, where we view \mathfrak{S} and \mathfrak{T} as functions from $S \times \mathbb{N}$ to distributions over actions.

For risk-aversion parameters λ of at most $1/2$, the following theorem implies that optimal schedulers for the MADPE can be chosen to be ERMax-schedulers.

► **Theorem 3.3.** *Let $\mathcal{M} = (S, Act, P, s_{init}, rew, goal)$ be an MDP satisfying Assumption 1 and let $\lambda \in (0, \frac{1}{2}]$ be a parameter for the MADPE. Further, let \mathfrak{T} be a memoryless deterministic scheduler with $\mathbb{E}_{\mathcal{M},s}^{\mathfrak{T}}(rew) = \mathbb{E}_{\mathcal{M},s}^{\max}(rew)$. Let $k = \lceil \mathbb{E}_{\mathcal{M}}^{\max}(rew) \rceil$. Then, for any reward-based scheduler \mathfrak{S} , we have $\text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}}(rew) \leq \text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S} \uparrow k \mathfrak{T}}(rew)$.*

The theorem is shown by expressing the MADPE using conditional expectations under the condition that the reward exceeds the bound k . Note that the theorem implies that it does not matter which expectation optimal scheduler \mathfrak{T} is chosen after a reward of at least $\mathbb{E}_{\mathcal{M}}^{\max}(rew)$ has been accumulated.

3.3 Computing the maximal MADPE

Theorem 3.3 tells us that the value $\text{MADPE}[\lambda]_{\mathcal{M}}^{\max}$ in an MDP \mathcal{M} for $\lambda \in (0, 1/2]$ is the supremum of $\text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}}$ over all reward-based schedulers \mathfrak{S} that behave according to a fixed memoryless deterministic scheduler \mathfrak{T} maximizing the expected reward as soon as a reward of more than $\mathbb{E}_{\mathcal{M}}^{\max}(rew)$ has been accumulated. Let us denote the set of such schedulers by $\text{Sched}_{\mathcal{M}}^{\mathfrak{T}}$.

The result shares some similarity with the results in [5] on the computation of maximal conditional expected rewards under the condition that a set of target states is reached. In both cases, a reward-based scheduler that has to keep track of the accumulated reward up to some bound B has to be computed. The bound B , however, is obtained quite differently. Here, the maximal expected accumulated reward can be used as this bound. The bound in [5] is in general much larger (although also exponential). Similar reward-based schedulers are also necessary for the model-checking of temporal formulas with certain reward operators [10] and for the optimization of the variance-penalized expectation [25].

We are now in the position to provide a model transformation such that afterwards we can restrict ourselves to memoryless schedulers. Given the MDP $\mathcal{M} = (S, Act, P, s_{init}, rew, goal)$, let $k = \lceil \mathbb{E}_{\mathcal{M}}^{\max}(rew) \rceil$ and let ℓ be the largest reward of a state-weight pair in \mathcal{M} . We now define the MDP $\mathcal{N} = (S', Act', P', s'_{init}, rew', goal')$.

The state space $S' = S \times \{0, \dots, k + \ell - 1\} \cup \{goal'\}$ and represents states together with the reward that has been accumulated so far, as well as a new trap state $goal'$. The initial state is $s'_{init} = (s_{init}, 0)$. The set of actions is extended by one new action τ . The transition probability function P' for $(s, w) \in S \times \{0, \dots, k + \ell - 1\}$ and $\alpha \in Act$ is given by $P'((s, w), \alpha, (t, v)) = P(s, \alpha, t)$ if $w \leq k - 1$ and $v = w + rew(s, \alpha)$, and is set to 0 otherwise. So, in all states in $S \times \{k, \dots, k + \ell - 1\}$ and in $\{goal'\} \times \{0, \dots, k - 1\}$ none of the actions in Act are enabled. Instead in these states the new action τ is enabled and leads to the trap state $goal'$ with probability 1. The reward function is 0 on all state-action pairs containing an action from Act . Only the new action τ gets assigned a reward by

$$\begin{aligned} rew'((goal, w)) &= w && \text{for all } w \in \{0, \dots, k + \ell - 1\} \quad \text{and} \\ rew'((s, w)) &= w + \mathbb{E}_{\mathcal{M},s}^{\max}(rew) && \text{for } s \in S \setminus \{goal\} \text{ and } w \in \{k, \dots, k + \ell - 1\}. \end{aligned}$$

So, in \mathcal{N} , rewards are only received in the very last step when entering the trap state $goal'$.

Now, a scheduler $\mathfrak{S} \in \text{Sched}_{\mathcal{M}}^{\mathfrak{T}}$ for \mathcal{M} can be seen as a memoryless scheduler for \mathcal{N} and vice versa: The scheduler \mathfrak{S} makes decision for all state-reward pairs (s, w) with $s \neq goal$ and $w < \mathbb{E}_{\mathcal{M}}^{\max}(rew)$. For higher values of accumulated reward, it switches to the behavior of the memoryless scheduler \mathfrak{T} . A memoryless scheduler for \mathcal{N} has to choose a probability distribution over Act on the same pairs (s, w) . For higher values of w or for pairs $(goal, w)$, only action τ is enabled in \mathcal{N} . So, with a slight abuse of notation, we interpret schedulers in $\text{Sched}_{\mathcal{M}}^{\mathfrak{T}}$ for \mathcal{M} also as memoryless schedulers for \mathcal{N} and vice versa.

► **Remark 3.4.** As reward-based schedulers are sufficient to maximize the MADPE and in \mathcal{N} rewards are only received in the last step, we can conclude that memoryless schedulers are sufficient to maximize the MADPE in \mathcal{N} .

► **Lemma 3.5.** *Given \mathcal{M} and \mathcal{N} as above, a scheduler $\mathfrak{S} \in \text{Sched}_{\mathcal{M}}^{\mathfrak{X}}$ and $\lambda \in (0, 1/2]$, we have $\text{MADPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) = \text{MADPE}[\lambda]_{\mathcal{N}}^{\mathfrak{S}}(\text{rew}')$.*

We utilize the MDP \mathcal{N} to compute the maximal MADPE via a quadratic program:

► **Theorem 3.6.** *Let \mathcal{M} be an MDP with non-negative rewards and $\lambda \in (0, 1/2]$. Then, $\text{MADPE}[\lambda]_{\mathcal{M}}^{\max}$ is the optimal solution to a linearly-constrained quadratic program that can be constructed from \mathcal{M} and λ in exponential time.*

Note that the MDP \mathcal{N} can be constructed in exponential time from \mathcal{M} as the numerical value of the maximal expected value $\mathbb{E}_{\mathcal{M}}^{\max}(\text{rew})$ is at most exponentially large in the size of \mathcal{M} . So, it is sufficient to construct a quadratic program from \mathcal{N} in polynomial time. In the sequel, we provide the construction of the quadratic program and prove its correctness.

We start by providing linear constraints that specify the possible combinations of expected frequencies of state-action-pairs under some scheduler. We use variables $x_{s,w,\alpha}$ for all $s \in S$, $w \in \{0, 1, \dots, k + \ell - 1\}$, and $\alpha \in \text{Act}'((s, w))$. For these variables, we require

$$x_{s,w,\alpha} \geq 0, \quad \text{and} \quad (1)$$

$$\sum_{\alpha \in \text{Act}(s)} x_{s,w,\alpha} = \sum_{t \in S, \beta \in \text{Act}(t)} x_{t,w-\text{rew}(t,\beta),\beta} \cdot P(t, \beta, s) + \mathbb{1}_{(s,w)=(s_{\text{init}},0)} \quad (2)$$

where $\mathbb{1}_{(s,w)=(s_{\text{init}},0)} = 1$ iff $s = s_{\text{init}}$ and $w = 0$, and $\mathbb{1}_{(s,w)=(s_{\text{init}},0)} = 0$ otherwise. In any solution to these two constraints, the variables $x_{s,w,\alpha}$ represent the expected frequency with which action α is chosen in state (s, w) under some scheduler. This is made precise below.

Rewards are only accumulated on the final transitions from a state (s, w) to goal' via action τ for $s = \text{goal}$ or $w \geq k$. As these transitions lead to the absorbing state with probability 1, the expected frequency with which the action τ is chosen is the probability with which the respective transition is taken. So, we can encode the expected value in an auxiliary variable e defined via the constraint

$$e = \sum_{w=0}^{k-1} x_{\text{goal},w,\tau} \cdot w + \sum_{w=k}^{k+\ell-1} \sum_{s \in S} x_{s,w,\tau} \cdot (w + \mathbb{E}_{\mathcal{M},s}^{\max}(\text{rew})). \quad (3)$$

► **Lemma 3.7.** *For any solution vector to constraints (1) – (3), there is a scheduler \mathfrak{S} for \mathcal{N} such that $\Pr_{\mathcal{N}}^{\mathfrak{S}}(\diamond(s, w)) = x_{s,w,\tau}$ for all (s, w) with $s = \text{goal}$ or $w \geq k$ and such that $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) = e$; and vice versa.*

Now, we can use these auxiliary variables to encode the MADPE as an objective function:

$$\text{maximize } e - \lambda \left(\sum_{w=0}^{k-1} x_{\text{goal},w,\tau} \cdot |w - e| + \sum_{w=k}^{k+\ell-1} \sum_{s \in S} x_{s,w,\tau} \cdot |w + \mathbb{E}_{\mathcal{M},s}^{\max}(\text{rew}) - e| \right) \quad (4)$$

This function still contains the absolute value operator. However, all absolute value terms occur with a negative sign. Therefore, we can use further variables g_i for $i \in \{0, \dots, k - 1\}$ and $h_{s,w}$ for $(s, w) \in S \times \{k, \dots, k + \ell - 1\}$ to capture the absolute value. The following constraints state that these variables are at least as big as the respective absolute value terms. For $w \in \{0, \dots, k - 1\}$, we require

$$g_w \geq w - e \quad \text{and} \quad -g_w \leq w - e. \quad (5)$$

For $(s, w) \in S \times \{k, \dots, k + \ell - 1\}$, we require

$$h_{s,w} \geq w + \mathbb{E}_{\mathcal{M},s}^{\max}(rew) - e \quad \text{and} \quad -h_{s,w} \leq w + \mathbb{E}_{\mathcal{M},s}^{\max}(rew) - e. \quad (6)$$

The new objective function can now be written as

$$\text{maximize} \quad e - \lambda \left(\sum_{w=0}^{k-1} x_{goal,w,\tau} \cdot g_w + \sum_{w=k}^{k+\ell-1} \sum_{s \in S} x_{s,w,\tau} \cdot h_{s,w} \right). \quad (7)$$

► **Theorem 3.8.** *The optimal solution to (7) under constraints (1) - (3), (5), and (6) is the maximal MADPE $\text{MADPE}[\lambda]_{\mathcal{N}}^{\max}$.*

Proof. As all variables are non-negative, the variables g_w with $0 \leq w \leq k - 1$ and $h_{s,w}$ with $w \geq k$ in the objective function (7) occur under a negative sign. To maximize the objective function, these variables hence have to be set to the minimal possible values given the value of the variable e . By constraints (5) and (6), these minimal possible values are the values $|w - e|$ and $|w + \mathbb{E}_{\mathcal{M},s}^{\max}(rew) - e|$, respectively. So, the optimal value of this quadratic objective function is the same as of the objective function (4), which directly encodes the MADPE. ◀

3.4 Computational hardness of the MADPE

The complexity class PP [14] is characterized as the class of languages \mathcal{L} that have a probabilistic polynomial-time bounded Turing machine $M_{\mathcal{L}}$ such that $\tau \in \mathcal{L}$ if and only if $M_{\mathcal{L}}$ accepts τ with probability at least $1/2$ for all words τ . We will show PP-hardness under polynomial-time Turing reductions. So, for the reduction, we allow querying an oracle for the problem we reduce to. A polynomial time algorithm for a problem that is PP-hard under polynomial Turing reductions would imply that the polynomial hierarchy collapses [28].

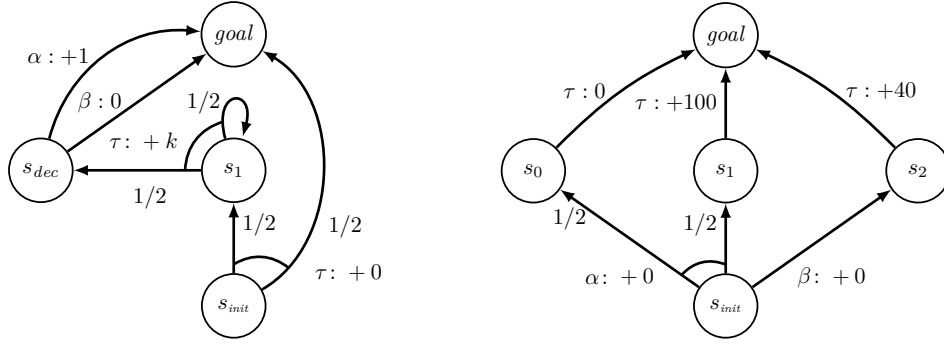
► **Theorem 3.9.** *Deciding for an acyclic Markov chain \mathcal{M} and a threshold $\vartheta \in \mathbb{Q}$ whether $\text{MAD}_{\mathcal{M}}(rew) \geq \vartheta$ is PP-hard under polynomial-time Turing reductions.*

Proof sketch. We reduce from the following problem that is shown to be PP-hard in [16]: Given an acyclic Markov chain $\mathcal{M} = (S, P, s_{init}, rew)$, and a natural number t , decide whether $\Pr_{\mathcal{M}}(rew > t) \geq 1/2$. We first show that the exact value $\text{MAD}_{\mathcal{M}}(rew)$ can be computed in acyclic Markov chains via a binary search using polynomially many calls to an oracle for the threshold problem. Then, we prove that $\Pr_{\mathcal{M}}(rew > t)$ can be computed by comparing the MAD in two variations of \mathcal{M} that ensure that the expected value of rew in these variations is t and $t + 1/2$, respectively. ◀

► **Corollary 3.10.** *Deciding for an acyclic Markov chain \mathcal{M} , $\lambda \in \mathbb{Q}_+$ and $\vartheta \in \mathbb{Q}$ if $\text{MADPE}[\lambda]_{\mathcal{M}}(rew) \geq \vartheta$ is PP-hard under polynomial-time Turing reductions.*

4 Semi-deviation measure-penalized expectation

To overcome the restrictions on the parameter λ for the MADPE or to overcome the undesirable behavior observed for the VPE, one might be tempted to consider the semi-MAD (SMAD) or the semi-variance as a deviation measure that only considers outcomes below the expected value as a measure for the penalty.


 (a) The MDP \mathcal{M} used in Example 4.1.

 (b) The MDP \mathcal{M} used in Example 4.2.

 ■ **Figure 3** Two example MDPs for phenomena of the SVPE.

Semi-MAD-penalized expectation

We define $\text{SMAD}(X) = \mathbb{E}(\max(0, \mathbb{E}(X) - X))$ for a random variable X . So, all outcomes above the expected value do not contribute to the SMAD. However, the SMAD is always half the MAD, i.e., $\text{SMAD}(X) = \text{MAD}(X)/2$, as one can easily compute (see [6]). So, using the SMAD as a penalty term is the same as using the MAD besides a rescaling of the penalty factor λ by a factor of 2.

Semi-variance-penalized expectation (SVPE)

We now define the semi-variance, to only treat outliers below the expected value with a quadratic penalty. However we will see that SVPE-optimal schedulers might still have to be ERMin-schedulers. We define the semi-variance by ignoring outliers above the expected value as follows $\text{SV}(X) := \mathbb{E}((\min(X - \mathbb{E}(X), 0))^2)$. Applied to the accumulated reward in an MDP $\mathcal{M} = (S, \text{Act}, P, s_{\text{init}}, \text{rew}, \text{goal})$, we define $\text{SV}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) := \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}((\min(\text{rew} - \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}), 0))^2)$ for schedulers \mathfrak{S} . Using this as a penalty, we obtain the SVPE for a parameter λ

$$\text{SVPE}[\lambda]_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) = \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) - \lambda \cdot \text{SV}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew})$$

and define the optimal value $\text{SVPE}[\lambda]_{\mathcal{M}}^{\max}(\text{rew})$ as usual. Besides the possible necessity of ERMin-schedulers, we will see that randomization is necessary to optimize the SVPE in contrast to the VPE, for which optimal deterministic (finite-memory) schedulers exist [25].

► **Example 4.1** (ERMin-schedulers). Let $\lambda > 0$ be a parameter for the SVPE. Consider the MDP \mathcal{M} depicted in Figure 3a where the weight k is some natural number $k > 1/\lambda$. First, observe that under any scheduler \mathfrak{S} , we have $k \leq \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\text{rew}) \leq k + 1/2$. Now, let $\ell \geq 2$ be a natural number and let \mathfrak{S}_p be a family of schedulers for $p \in [0, 1]$ that behaves exactly the same on all paths except for the path that reaches s_{dec} with accumulated reward exactly $\ell \cdot k$. In this state, \mathfrak{S}_p chooses α with probability p and β with probability $1 - p$.

We now want to compare the SVPE of \mathfrak{S}_p for $p > 0$ to the SVPE of \mathfrak{S}_0 . So, let $\lambda > 0$ be given. First, we define $E := \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_0}(\text{rew})$ and observe $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_p}(\text{rew}) = E + \frac{p}{2^{\ell+1}}$ as the path on which \mathfrak{S}_p and \mathfrak{S}_0 differ has probability $\frac{1}{2^{\ell+1}}$. Furthermore, both schedulers differ only on a path with a reward higher than the maximal possible expected accumulated reward, which is $k + 1/2$. This means that the semivariance under \mathfrak{S}_p will be larger as under \mathfrak{S}_0 . Note that exactly the outcomes with reward at most k contribute to the semivariance and these outcomes have exactly the same probability under \mathfrak{S}_p and \mathfrak{S}_0 . However, the expected value under \mathfrak{S}_p is

higher. We estimate $\mathbb{S}\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) - \mathbb{S}\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_0}(rew) \geq \frac{1}{2}(E + \frac{p}{2^{\ell+1}})^2 - \frac{1}{2}E^2$ by only considering the increase in the squared distance from the mean for the outcome 0 that occurs with probability $1/2$ under both schedulers. So, we can conclude $\mathbb{S}\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) - \mathbb{S}\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_0}(rew) \geq \frac{1}{2}(\frac{2Ep}{2^{\ell+1}} + (\frac{p}{2^{\ell+1}})^2) \geq \frac{Ep}{2^{\ell+1}}$. For the SVPE, this implies $\mathbb{S}\mathbb{V}\mathbb{P}\mathbb{E}[\lambda]_{\mathcal{M}}^{\mathfrak{S}_p}(rew) - \mathbb{S}\mathbb{V}\mathbb{P}\mathbb{E}[\lambda]_{\mathcal{M}}^{\mathfrak{S}_0}(rew) \leq \frac{p}{2^{\ell+1}} - \lambda \frac{Ep}{2^{\ell+1}}$. As $E \geq k$ and $\lambda > 1/k$, the SVPE under scheduler \mathfrak{S}_0 is higher than under \mathfrak{S}_p . Note that $\ell \geq 2$ was chosen arbitrarily. So, this argument shows that any scheduler can be improved by always scheduling β in s_{dec} as soon as the accumulated reward is at least $2k$.

For each $\lambda > 0$, we have provided an MDP in which optimal schedulers are necessarily ERMin-schedulers. This is exactly the undesirable behavior as for the VPE we aim to overcome. So, the SVPE is not a suitable alternative.

► **Example 4.2.** To conclude, we show that randomization is necessary to maximize the SVPE. Consider the MDP \mathcal{M} depicted in Figure 3b. Let \mathfrak{S}_p be the scheduler that chooses action α with probability p . Further, let $\lambda = \frac{1}{100}$. We compute $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) = 40 + 10p$. Under \mathfrak{S}_p , reward 40 is accumulated with probability $1 - p$ and reward 0 with probability $p/2$. So, we obtain $\mathbb{S}\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) = (1 - p) \cdot (10p)^2 + \frac{p}{2} \cdot (40 + 10p)^2 = 800p + 500p^2 - 50p^3$. Finally, we compute $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) - \lambda \mathbb{S}\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) = 40 + 2p - 5p^2 + \frac{1}{2}p^3$. We determine the unique maximum of this expression on the interval $[0, 1]$ at the zero of its derivative, which lies at $p \approx 0.206$. So, randomization is necessary in order to maximize the SVPE in this MDP.

To conclude, let us compute the variance to illustrate that randomization is not increasing the VPE. We obtain $\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) = \mathbb{S}\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) + \frac{p}{2}(60 - 10p)^2 = 2600p - 100p^2$. For the VPE for an arbitrary parameter $\lambda > 0$, this results in $\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) - \lambda \mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_p}(rew) = 40 + 10p - 2600\lambda p + 100\lambda p^2$. Due to the positive coefficient in front of p^2 this is a parabola opened upwards. So, for any λ , one of the deterministic schedulers with $p = 0$ or $p = 1$ is optimal.

5 Threshold-based penalty

The MADPE penalizes outcomes below the expected value of the accumulated reward. The computation of the optimal MADPE via a quadratic program of exponential size, however, might not be feasible on large models. A conceptually simpler alternative, for which we will be able to provide a pseudo-polynomial optimization algorithm, is to externally fix a threshold t and to penalize outcomes below this threshold t . To this end, we define a threshold-based penalty function $TBP_t^\lambda: \mathbb{R} \rightarrow \mathbb{R}$ for parameters $\lambda, t > 0$ by $TBP_t^\lambda(x) = x - \lambda \cdot \max(t - x, 0)$.

This function returns x if x is at least t and otherwise penalizes the deviation below the value t linearly with the penalty factor λ . In an MDP \mathcal{M} , our goal is now to maximize – by choosing a scheduler \mathfrak{S} – the threshold-based-penalized expectation (TBPE)

$$\mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(TBP_t^\lambda(rew)) = \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(rew) - \lambda \mathbb{E}_{\mathcal{M}}^{\mathfrak{S}}(\max(t - rew, 0))$$

Note that in a Markov chain \mathcal{N} , the TBPE agrees with the SMADPE if we set $t = \mathbb{E}_{\mathcal{N}}(rew)$.

The main theorem is the following. Omitted proofs can be found in the full version [6].

► **Theorem 5.1.** *Let $\mathcal{M} = (S, Act, P, s_{init}, rew, goal)$ be an MDP satisfying Assumption 1 and let $t, \lambda > 0$ be rationals. Then, $\mathbb{E}_{\mathcal{M}}^{\max}(TBP_t^\lambda(rew))$ and an optimal scheduler can be computed in time polynomial in the size of \mathcal{M} and in the numerical value of t .*

The theorem follows from the following lemma:

► **Lemma 5.2.** *Given \mathcal{M} , t , and λ as in Theorem 5.1, we can construct an MDP \mathcal{M}' with reward function rew' (that takes rational rewards that may be negative) and with $|S| \cdot \lceil t \rceil$ many states in time polynomial in $|S| \cdot \lceil t \rceil$ such that $\mathbb{E}_{\mathcal{M}}^{\max}(TBP_t^\lambda(rew)) = \mathbb{E}_{\mathcal{M}'}^{\max}(rew')$.*

Proof sketch. The MDP \mathcal{M}' is an unfolding of the MDP \mathcal{M} that keeps track of the accumulated reward until it exceeds t . So, states are extended with a second component specifying the reward accumulated so far. This second component does not change anymore once it reaches t . For a state action pair $((s, w), \alpha)$, the new reward function is defined as $rew'((s, w), \alpha) = TBP_t^\lambda(w + rew(s, \alpha)) - TBP_t^\lambda(w)$. The initial state $(s_{init}, 0)$ is reached via one additional new transition with reward $TBP_t^\lambda(0)$ (which is negative). ◀

While \mathcal{M}' constructed in this proof has a rational reward function that may be negative, the MDP \mathcal{M}' does not contain end components. Hence, the maximization of the expected accumulated reward in \mathcal{M}' can be carried out in polynomial time [7] leading to Theorem 5.1. Furthermore, memoryless deterministic schedulers for \mathcal{M}' are sufficient for the maximization. These schedulers correspond to deterministic, finite-memory ERMax-schedulers for \mathcal{M} .

► **Remark 5.3.** The proof of Lemma 5.2 (and Thm. 5.1) works analogously for any penalty function that penalizes outcomes below t : for any function m such that $m(x) = x$ for $x \geq t$ that is computable in polynomial time on natural numbers, we can construct \mathcal{M}' with a reward function rew' with $|S| \cdot \lceil t \rceil$ many states in time polynomial in $|S| \cdot \lceil t \rceil$ such that $\mathbb{E}_{\mathcal{M}}^{\max}(m(rew)) = \mathbb{E}_{\mathcal{M}'}^{\max}(rew')$ (for more details, see [6]). Again, \mathcal{M}' has no end components and the maximal expected reward in \mathcal{M}' can be computed in time polynomial in the size of \mathcal{M}' [7].

Finally, we show a hardness result similar as for the MADPE.

► **Theorem 5.4.** *Given an acyclic Markov chain $\mathcal{M} = (S, P, s_{init}, rew)$ and $\vartheta, t \in \mathbb{Q}$, deciding whether $\mathbb{E}_{\mathcal{M}}(TBP_t^1(rew)) \geq \vartheta$ is PP-hard under polynomial-time Turing reductions.*

Note that this hardness result holds for a fixed parameter. The choice of this parameter $\lambda = 1$ is arbitrary. The proof works analogously for any positive parameter $\lambda > 0$.

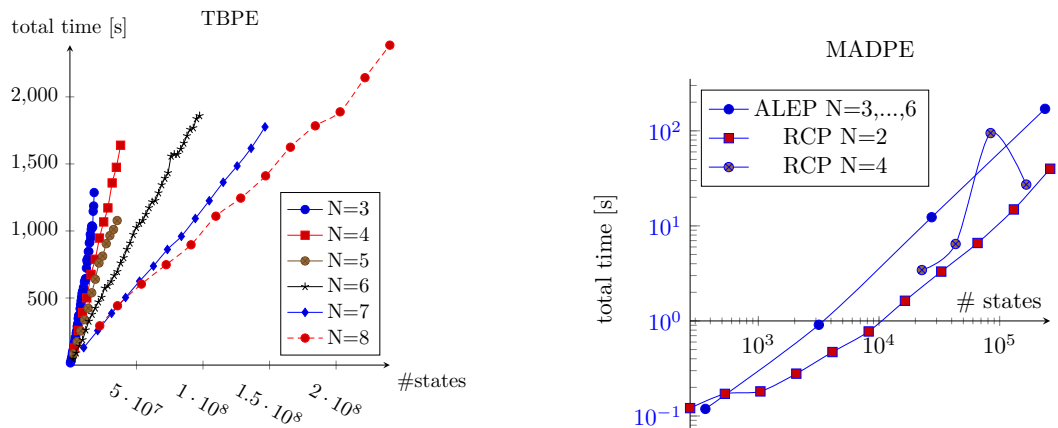
6 Prototypical implementation and first experiments

To give a prototypical proof-of-concept for the application of the MADPE and TBPE in practice, we run experiments using the model-checker PRISM [9] and the optimization problem solver Gurobi [15]. The source code for the experiments is available on github¹. All measurements were done on a machine running Windows 10 Pro 22H2 with an Intel Core i9-9900K CPU and 32GB RAM. We use MDP models written in the PRISM input language (available on the PRISM website²) for the asynchronous leader election protocol (ALEP) [17] and, in the case of the MADPE, also for the randomized consensus protocol (RCP) [2]. For both protocols, parameters can be chosen leading to models of different sizes and in the models for both protocols non-negative rewards are specified.

To test our algorithm for the TBPE, for each PRISM model for the ALEP with number of processes $N = 3, \dots, 8$, we added a single module which implements the reward counter until reaching the threshold and a new reward definition as in the construction used in the proof of Lemma 5.2. We used the penalty factor $\lambda = \frac{3}{2}$ in our examples and varied the threshold t . In Figure 4a, the sizes of the unfolded MDPs for varying values of t , which are proportional to t , and the time needed to compute the maximal TBPE are shown. We observe that for this example the required time grows approximately linearly with the size of the unfolded MDP and consequently with the numerical value of t . For the model with $N = 8$, which

¹ <https://github.com/experiments-collection/risk-averse-stochastic-shortest-paths>

² <https://www.prismmodelchecker.org/>



(a) The number of states of the unfolded MDPs and the time to compute the optimal TBPE for different parameter choices for the ALEP.

(b) Time to build and solve the quadratic program for the maximization of the MADPE.

■ **Figure 4** Experimental evaluation of the algorithms for TBPE and MADPE.

has approximately $1.8 \cdot 10^7$ many states, and $t = 13$, the unfolded MDP has approximately $2.4 \cdot 10^8$ many states and the computation of the optimal TBPE takes approximately 2385 seconds. More detailed plots for different values for N can be found in Appendix A.

To test our algorithm for the MADPE using quadratic programs, we use the ALEP and the RCP models with various parameter choices. The parameter λ is set to 0.4. First we run PRISM to obtain a model representation with all states, transitions, rewards and the maximal expected total reward from each state. Second, we run a python script which constructs all the constraints as described in Section 3 to obtain a linearly constrained program with a quadratic objective. The script uses Gurobi [15] to then solve the optimization problem. The diagram in Figure 4b shows the total time for running the toolchain over the number of reachable states of each model according to PRISM's output. For the largest tested models with approximately $2 \cdot 10^5$ many states, the maximal MADPE could be computed in less than 200 seconds.

7 Conclusion

For various deviation measures, we investigated the deviation-measure-penalized expectation as risk-averse objective applied to the maximization of accumulated rewards in MDPs. As known from the literature, the VPE suffers from the fact that optimal schedulers have to be ERMin-schedulers. Surprisingly, this can still be the case for the SVPE. For the MADPE, a different picture arises: If the penalty factor λ is at most $1/2$, optimal schedulers can be chosen to be ERMax-schedulers. If $\lambda > 1/2$, ERMin-schedulers can be necessary. Finally, the threshold-based penalty mechanism in the TBPE ensures that optimal schedulers are ERMax-schedulers. For an overview of the further results regarding computational complexity and the structure of optimal schedulers see Table 1.

Despite the PP-hardness results for acyclic Markov chains, the first experimental evaluation of the two cases that ensure the existence of optimal ERMax-schedulers, namely the TBPE in general and the MADPE for small penalty factors $\lambda \leq 1/2$, indicates that the optimization seems to be possible in reasonable time on models of considerable size. Further experiments on the scalability of the algorithms, however, are left as future work. In addition, future experiments should examine whether the optimal schedulers for the different measures show a reasonable risk-averse behavior in case studies.

We addressed the maximization of accumulated rewards here. As we work with non-negative rewards, the case of minimization is not symmetric and is subject to future investigations. Finally, the studied objectives can be transferred to other random variables such as the mean payoff, which is a further interesting direction for future work.

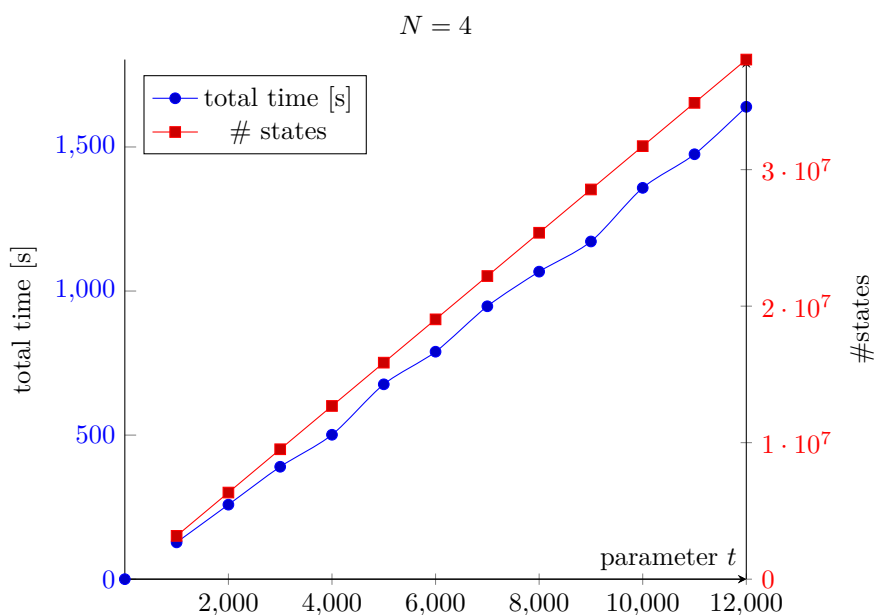
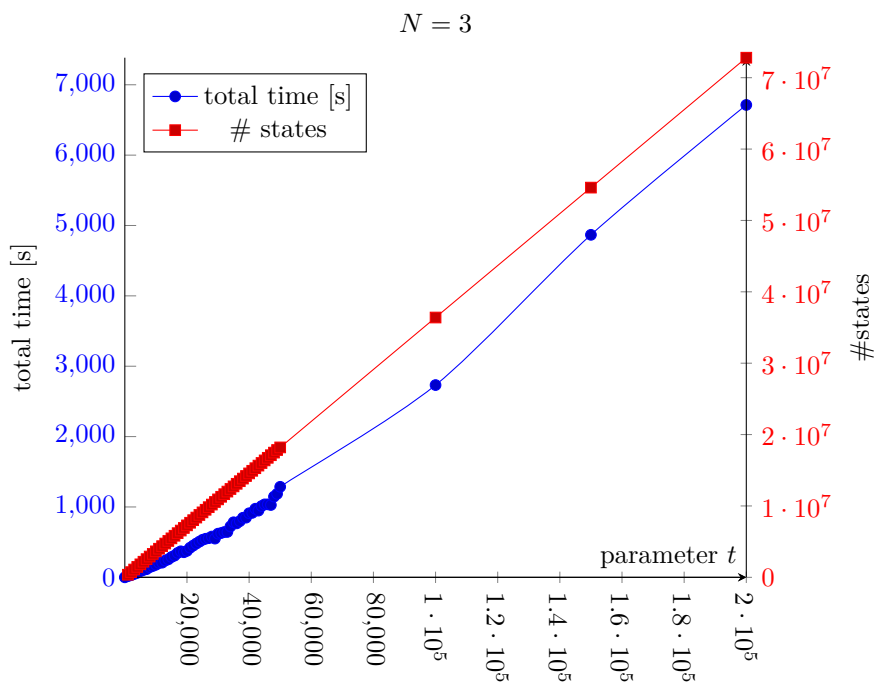
References

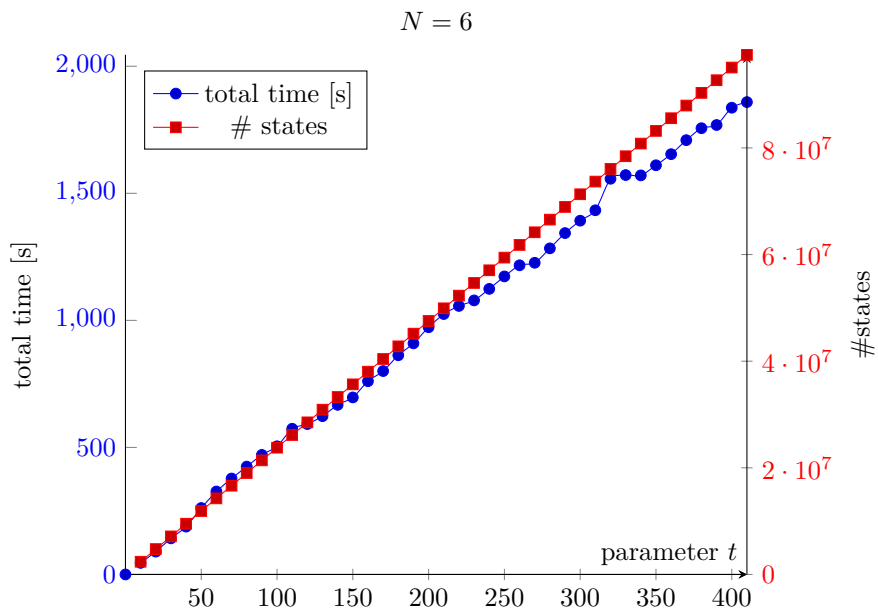
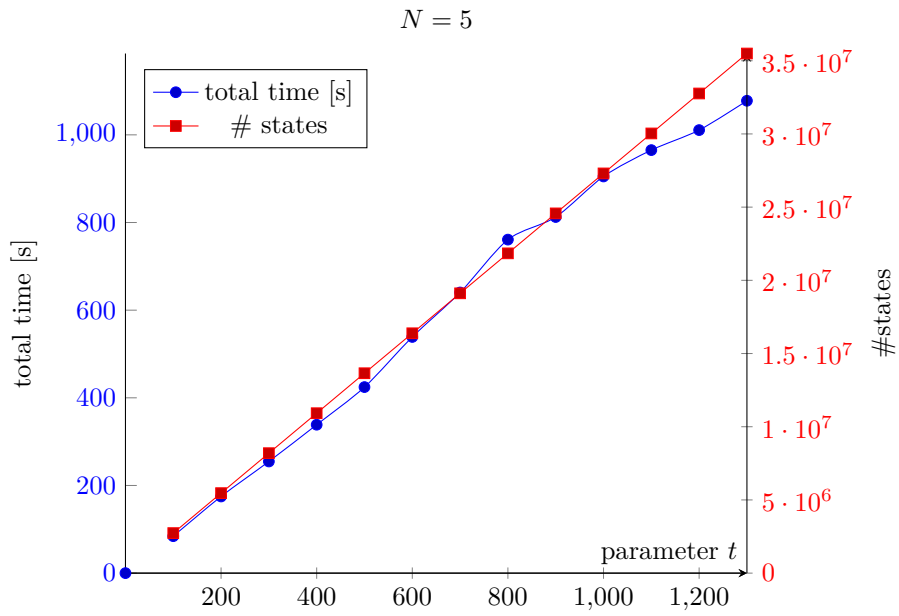
- 1 Mohamadreza Ahmadi, Anushri Dixit, Joel W. Burdick, and Aaron D. Ames. Risk-averse stochastic shortest path planning. In *2021 60th IEEE Conference on Decision and Control (CDC), Austin, TX, USA, December 14-17, 2021*, pages 5199–5204. IEEE, 2021. doi:10.1109/CDC45484.2021.9683527.
- 2 James Aspnes and Maurice Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, 1990. doi:10.1016/0196-6774(90)90021-6.
- 3 Christel Baier, Krishnendu Chatterjee, Tobias Meggendorfer, and Jakob Piribauer. Entropic risk for turn-based stochastic games. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPICs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.MFCS.2023.15.
- 4 Christel Baier, Marcus Daum, Clemens Dubsclaff, Joachim Klein, and Sascha Klüppelholz. Energy-utility quantiles. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*, volume 8430 of *Lecture Notes in Computer Science*, pages 285–299. Springer, 2014. doi:10.1007/978-3-319-06200-6_24.
- 5 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Maximizing the conditional expected reward for reaching the goal. In Axel Legay and Tiziana Margaria, editors, *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10206 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2017. doi:10.1007/978-3-662-54580-5_16.
- 6 Christel Baier, Jakob Piribauer, and Maximilian Starke. Risk-averse optimization of total rewards in markovian models using deviation measures, 2024. arXiv:2407.06887.
- 7 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16:580–595, 1991. doi:10.1287/moor.16.3.580.
- 8 Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Trading performance for stability in markov decision processes. *J. Comput. Syst. Sci.*, 84:144–170, 2017. doi:10.1016/j.jcss.2016.09.009.
- 9 T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In N. Piterman and S. Smolka, editors, *Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *LNCS*, pages 185–191. Springer, 2013. doi:10.1007/978-3-642-36742-7_13.
- 10 Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods Syst. Des.*, 43(1):61–92, 2013. doi:10.1007/S10703-013-0183-7.
- 11 EJ Collins. Finite-horizon variance penalised Markov decision processes. *Operations-Research-Spektrum*, 19(1):35–39, 1997.
- 12 Luca de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *10th International Conference on Concurrency Theory (CONCUR)*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81, 1999. doi:10.1007/3-540-48320-9_7.
- 13 Jerzy A Filar, Lodewijk CM Kallenberg, and Huey-Miin Lee. Variance-penalized Markov decision processes. *Mathematics of Operations Research*, 14(1):147–161, 1989. doi:10.1287/moor.14.1.147.
- 14 John Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977. doi:10.1137/0206049.

- 15 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 16 Christoph Haase and Stefan Kiefer. The odds of staying on budget. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2015. doi:10.1007/978-3-662-47666-6_19.
- 17 A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1), 1990. doi:10.1016/0890-5401(90)90004-2.
- 18 Hiroshi Konno and Hiroaki Yamazaki. Mean-absolute deviation portfolio optimization model and its applications to tokyo stock market. *Management Science*, 37(5):519–531, 1991. URL: <http://www.jstor.org/stable/2632458>.
- 19 Jan Kretínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 609–618. ACM, 2018. doi:10.1145/3209108.3209176.
- 20 Xiaoteng Ma, Shuai Ma, Li Xia, and Qianchuan Zhao. Mean-semivariance policy optimization via risk-averse reinforcement learning. *J. Artif. Intell. Res.*, 75:569–595, 2022. doi:10.1613/jair.1.13833.
- 21 Petr Mandl. On the variance in controlled Markov chains. *Kybernetika*, 7(1):1–12, 1971. URL: <http://www.kybernetika.cz/content/1971/1/1>.
- 22 Shie Mannor and John N. Tsitsiklis. Mean-variance optimization in Markov decision processes. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML'11*, pages 177–184, Madison, WI, USA, 2011. Omnipress. URL: https://icml.cc/2011/papers/156_icmlpaper.pdf.
- 23 Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. URL: <http://www.jstor.org/stable/2975974>.
- 24 Harry M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. Yale University Press, 1959. URL: <http://www.jstor.org/stable/j.ctt1bh4c8h>.
- 25 Jakob Piribauer, Ocan Sankur, and Christel Baier. The variance-penalized stochastic shortest path problem. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 129:1–129:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.129.
- 26 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994. doi:10.1002/9780470316887.
- 27 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional markov decision processes. *Formal Methods Syst. Des.*, 50(2-3):207–248, 2017. doi:10.1007/s10703-016-0262-7.
- 28 Seinosuke Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. doi:10.1137/0220053.
- 29 Michael Ummels and Christel Baier. Computing quantiles in Markov reward models. In Frank Pfenning, editor, *16th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 7794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2013. doi:10.1007/978-3-642-37075-5_23.
- 30 Tom Verhoeff. Reward variance in Markov chains: A calculational approach. In *Proceedings of Eindhoven FASTAR Days*. Technische Universiteit Eindhoven, 2004.
- 31 Li Xia. Risk-sensitive Markov decision processes with combined metrics of mean and variance. *Production and Operations Management*, 29(12):2808–2827, 2020. doi:10.1111/poms.13252.

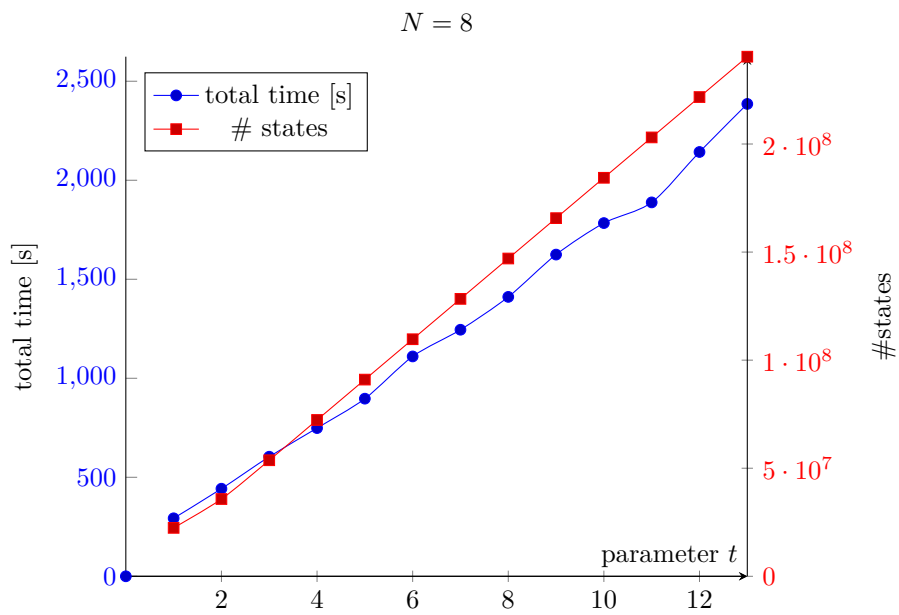
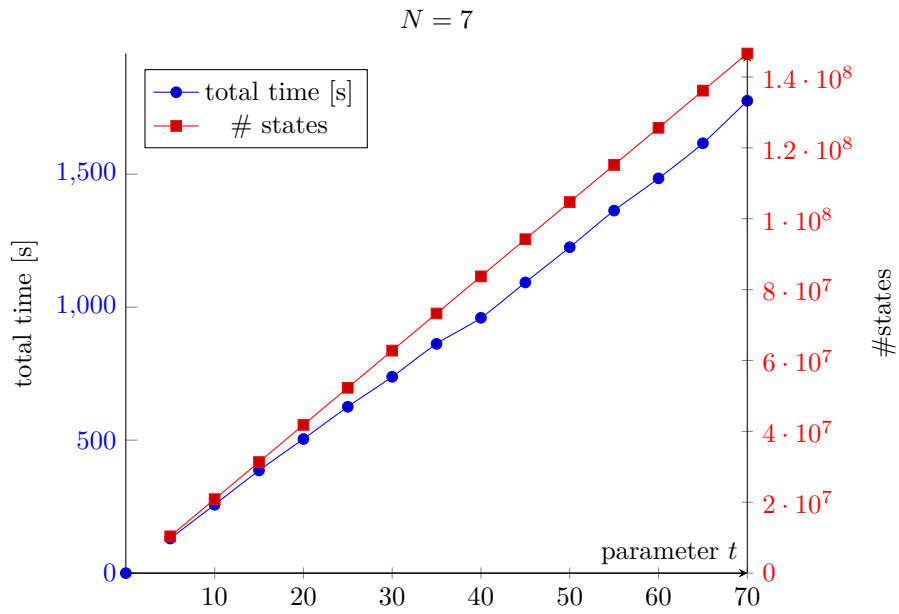
A Experimental evaluation

In the sequel, the number of states of the unfolded MDPs as well as the time to compute the maximal TBPE as described in Section 6 are depicted for the Asynchronous Leader Election Protocol with parameter $N = 3, \dots, 8$ and varying values of the parameter t . The number of states of the unfolded MDPs grows linearly in t as expected. Interestingly, also the required times seem to grow linearly in t .





9:20 Risk-Averse Optimization of Total Rewards in Markovian Models



Passive Learning of Regular Data Languages in Polynomial Time and Data

Mrudula Balachander 

Université libre de Bruxelles, Belgium

Emmanuel Filiot 

Université libre de Bruxelles, Belgium

Raffaella Gentilini 

Università degli Studi di Perugia, Italy

Abstract

A regular data language is a language over an infinite alphabet recognized by a deterministic register automaton (DRA), as defined by Benedikt, Ley and Puppis. The later model, which is expressively equivalent to the deterministic finite-memory automata introduced earlier by Francez and Kaminsky, enjoys unique minimal automata (up to isomorphism), based on a Myhill-Nerode theorem.

In this paper, we introduce a polynomial time passive learning algorithm for regular data languages from positive and negative samples. Following Gold’s model for learning languages, we prove that our algorithm can identify in the limit any regular data language L , i.e. it returns a minimal DRA recognizing L if a characteristic sample set for L is provided as input. We prove that there exist characteristic sample sets of polynomial size with respect to the size of the minimal DRA recognizing L . To the best of our knowledge, it is the first passive learning algorithm for data languages, and the first learning algorithm which is fully polynomial, both with respect to time complexity and size of the characteristic sample set.

2012 ACM Subject Classification Theory of computation \rightarrow Formal languages and automata theory; Theory of computation \rightarrow Automata over infinite objects

Keywords and phrases Register automata, passive learning, automata over infinite alphabets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.10

Funding *Mrudula Balachander*: PhD funded by F.R.S.-FNRS.

Emmanuel Filiot: Senior research associate of the F.R.S.-FNRS. His contribution to this work was partly funded by the FNRS MIS project F451019F and the FNRS PDR project T.0117.24.

Acknowledgements We thank Marie Tcheng for spotting some issue in the characterization of completeness in a preliminary version of this paper.

1 Introduction

Finite-memory automata (FMA) have been introduced by Francez and Kaminsky in the 90s [27], as an extension of finite automata to infinite alphabets of data values, which can be stored and compared using a finite set of registers. Since the introduction of FMA, a rich literature on automata for languages over infinite alphabets has emerged, e.g. see [12, 6, 35, 13, 11, 31, 24]. Automata for data languages have many applications in computer science, for instance in verification of concurrent systems [13, 14, 2, 1], of programs with dynamic allocation [24] as well as in reactive system synthesis [28, 20, 19]. Unlike in the finite alphabet setting, non-determinism and determinism often yield expressively inequivalent models, such as for FMA. In this paper, we consider languages defined by deterministic FMA (DFMA), which we call regular data languages. Regular data languages form a robust class of languages. They are for instance captured by many variations on the



© Mrudula Balachander, Emmanuel Filiot, and Raffaella Gentilini; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 10; pp. 10:1–10:21



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

data storage mechanism of DFMA: allowing registers to be emptied during computation, or requiring that they must hold distinct values, or allowing their contents to be reassigned [33], or finally requiring that they must follow a *last appearance record (LAR)* policy [6].

Register automata with last appearance record. The latter model of [6], which we simply call *deterministic register automata (DRA)*, is the chosen model in this paper for regular data languages. As shown in [6], for any regular data language L , there exists a unique DRA for L which is both minimal with respect to the number of states and number of registers. This is particularly relevant for learning regular data languages. In this model, data values are stored in a list whose length can vary during execution but is of bounded length k . Moreover, data occurring in the list are distinct. When a new data d is read, it is compared to the list α of stored values. The data d , if ever stored, has to be stored at the end of α , resulting in a new list αd . Values in α can be erased, and must be if d already occurs in α (to keep distinct values in memory), or if the list exceeds length k . E.g., consider the data language L_{\neq} of words of the form $(d_1 d_2)^n$ for all $n > 0$ and all integers $d_1 \neq d_2$. A DRA recognizing L_{\neq} would start with an empty memory, then store the 1st data d_1 , then the 2nd data d_2 (after checking $d_1 \neq d_2$). At this point its memory is the list $d_1 d_2$. Then it would alternate between (1) checking whether the next data is d_1 and update the memory to $d_2 d_1$, and (2) checking whether the next data is d_2 and update the memory to $d_1 d_2$.

Passive learning of languages. Language inference has a long history, with applications in (but not limited to) verification [29]. The (passive) inference of regular languages (over a finite alphabet) from positive and negative samples has been studied since the 60s, see e.g. [30] for a survey. Given a sample set $S = (I^+, I^-)$ consisting of a finite set I^+ (resp. I^-) of positive (resp. negative) samples, the goal is to infer a regular language as a DFA consistent with S , i.e. it accepts all words in I^+ and rejects those in I^- . Gold proposed *identification in the limit* of a regular language as a formal notion to characterize the learning capability of inference algorithms [23, 16]. The algorithm RPNI (standing for Regular Positive and Negative Inference) [34] is one of the reference algorithms in passive learning: given a sample set S , it returns in polynomial-time a DFA consistent with S , and for any regular language L , if S is characteristic enough for L , then RPNI returns \mathcal{A}_L . Moreover, there is always a characteristic sample of polysize in the size of \mathcal{A}_L . In other words, RPNI identifies regular languages in polynomial time and data. Since then, there have been many variants of RPNI, for example for regular tree languages [22], regular queries in trees [15] or ω -regular languages [8, 9]. Recently, RPNI was used in combination with reactive synthesis methods to automatically generate reactive systems from LTL specifications and examples [5]. The extension of [5] to the synthesis of data-processing reactive systems is our main motivation for the present work, as the former heavily relies on passive learning.

Despite a rich literature on passive learning for regular languages over *finite alphabets*, and on automata and logics for words over *infinite alphabets*, only few is yet known about passive learning for regular languages over infinite alphabets, to the best of our knowledge. Passive learning for symbolic automata has been investigated in [21]. This is an orthogonal model to register automata: symbolic automata have complex tests over data, but no storage mechanism, while register automata have simple data tests, but registers to store data. Handling data that need to be memorized in a passive learning context is one of the main difficulties of our paper. Let us also mention that there are many contributions in *active* learning for regular data languages, in particular extensions of L^* algorithm [7, 25, 26, 32]. All these extensions however run in exponential time.

Contributions. We prove that any regular data language L can be identified in polynomial time from polynomial data, in the size of the minimal DRA for L . To do so, we design an RPNI-like algorithm, which takes as input a finite sample set $S = (I^+, I^-)$ of positive and negative data words over an infinite domain of data values, and prove that it returns, in polynomial time, a DRA consistent with S , assuming data equality can be tested in polynomial time. We then show that for any regular language L , there exists a characteristic sample S_L , such that for any sample S containing S_L , our algorithm returns the minimal DRA recognizing L (as defined in [31]). To the best of our knowledge, it is the first fully polynomial passive learning algorithm for regular data languages.

Our RPNI algorithm is based on an alternative presentation of RPNI for DFA, compared to the original formulation of [34]. This alternative presentation is similar to the one of [8] for ω -regular languages. We believe this presentation is simpler than the original one, as it does not involve operations such as deterministic state-merging, and is easier to analyse. The idea is to start from a single-state automaton, and to incrementally extend it by adding new transitions, allowing more sample prefixes to be readable by the automaton (taken in length lexicographic order). To generalize the samples, when adding a new transition, existing states are prioritized as a target of the new transition, over creating a fresh new state. A transition can be added if the resulting automaton can still be completed into an automaton consistent with S . We call this S -completeness and show it can be checked in PTIME. In Section 3, we first present this alternative RPNI algorithm for DFA. It is of independent interest, but also helpful to understand its extension to DRA, which is done in Section 4. Briefly, when adding a transition, the algorithm now has to decide whether the incoming data must be stored and which of the stored data can be erased from memory. To do so, and to generalize the sample as much as possible, it tries both to reuse existing states for the target of the new transition, and to erase as many registers as possible, while preserving S -completeness.

In Section 5, we prove that this algorithm identifies any regular data language L from a characteristic sample set of size polynomial in the size of the minimal DRA for L . To prove that the characteristic sample is polynomial, classical automata-theoretic arguments (such as intersection closure) do not apply because they involve exponential blow-up. Instead, we adapt group-theoretic techniques for checking bisimulation of register automata as defined in [33], which exploits symmetries underlying sets of configurations of register automata. As a byproduct of our techniques, we obtain that equivalence of DRA can be checked in CONP.

2 Preliminaries

Data Words and Languages. In this paper, a *data domain* is an infinite countable set \mathcal{D} equipped with equality, whose elements are called *data*. In this paper, we assume an arbitrary well-ordering $<_{\mathcal{D}}$ over \mathcal{D} , so that every subset has a minimal element. It is also assumed that any data has an effective representation, and that data comparison $<_{\mathcal{D}}$ can be tested in polynomial time.

A (*data*) *word* w (or sometimes just *word*) is a finite sequence of data from \mathcal{D} . Given a word w having length $|w| = n$, we denote by $w[i]$ the i th data of w for $1 \leq i \leq |w|$. We define the length-lexicographic order over \mathcal{D}^* : $u <_{lex} v$ if $|u| < |v|$ or $|u| = |v|$, $u = wdu'$, $v = wd'v'$ for some $d <_{\mathcal{D}} d'$ and w, u', v' . For a set $X \subseteq \mathcal{D}^*$, we let $\text{Pref}(X)$ be the set of words u such that u is a prefix (not necessarily strict) of some word in X .

Any (total) function $\mu : \mathcal{D} \rightarrow \mathcal{D}$ can be morphically extended to $\mu : \mathcal{D}^* \rightarrow \mathcal{D}^*$, where $\mu(w)[i] = \mu(w[i])$ for all $w \in \mathcal{D}^*$ and $1 \leq i \leq |w|$. The function μ is called a *data permutation* if μ is bijective. Two data words $w_1, w_2 \in \mathcal{D}^*$ are said to be data-equivalent¹, written as

¹ In the terminology of orbit-finite sets, w_1 and w_2 are said to be in the same orbit [10].

$w_1 \simeq w_2$, if $w_1 = \mu(w_2)$ for some data permutation μ . Note that it implies that $|w_1| = |w_2|$. We denote by $[w]_{\simeq}$ the \simeq -class of any data word w . The following result is immediate (under the previous assumption that data equality is in PTIME):

► **Proposition 1.** *For any data domain \mathcal{D} , \simeq is decidable in PTIME.*

A (data) language L over \mathcal{D} is a set of data words over \mathcal{D} . It is *equivariant* if for all $w \in L$, $[w]_{\simeq} \subseteq L$. For $w_1, w_2 \in \mathcal{D}^*$, we write $w_1 =_L w_2$ when $w_1 \in L \leftrightarrow w_2 \in L$ holds.

Register Automata. Register automata have first been introduced by Kaminsky and Francez under the name *finite memory automata* [27]. In this paper, we use an equi-expressive model defined in [6], which enjoys a canonical, state-minimal and register-minimal form. In this model, the number of available registers may vary over time, but depends on the state, i.e. any configuration in state p has the same number $\lambda(p)$ of stored data. Moreover, registers follow a fixed policy in the way they are used, called *last appearance record*. Thanks to this policy, it is not necessary to give names to registers, and the set of stored data in a configuration is just a data word (of bounded length). Before giving the formal definition, let us informally explain how this model works. Any transition is of the form $t = (p, \alpha, E, q)$ where p, q are states, α is a data word of length $\lambda(p) + 1$, E is a subset of $\{1, \dots, \lambda(p) + 1\}$. The word α is seen as a test of the new data read as input against the registers, via \simeq -equivalence. There can be two types of α : *disequality tests*, which require all the data of α to be pairwise different, and *equality tests*, which require all data to be pairwise different but the last one, which repeats exactly once in α . E.g. over $\mathcal{D} = \mathbb{N}$, $\alpha = 1323$ is an equality test, $\alpha = 2341$ is a disequality test and $\alpha = 222$ is not a valid test. Any configuration in state p is of the form $(p, d_1 \dots d_{\lambda(p)})$ where $d_1 \dots d_{\lambda(p)} \in \mathcal{D}^*$ is the memory content. Reading a new incoming data d , transition t above can be triggered only if $d_1 \dots d_{\lambda(p)} d \simeq \alpha$, and in that case the automaton moves to state q . Some of the data in memory can be dropped, and E specifies which one, by their positions in $\{1, \dots, \lambda(p) + 1\}$. The automaton maintains the following invariant: no data is stored multiple times in memory, and the size of the memory only depends on the state. To maintain the first invariant, if α is an equality test $\alpha_1 \dots \alpha_{\lambda(p)+1}$ with $\alpha_j = \alpha_{\lambda(p)+1}$ for some $j \leq \lambda(p)$, then $j \in E$: this implies that only the *last* occurrence of d_j is kept. For the second invariant, we must have $\lambda(p) + 1 - |E| = \lambda(q)$.

► **Definition 2.** *A (deterministic) register automaton (DRA) over \mathcal{D} is a tuple $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$, where:*

- Q is the set of states, q_0 is the initial state with $\lambda(q_0) = 0$ and $F \subseteq Q$ are the final states.
- $k \in \mathbb{N}$ is the maximum number of stored values, and $\lambda : Q \rightarrow \{0, \dots, k\}$ is called an *availability function*;
- T is a finite set of transitions of the form (p, α, E, q) , where $p, q \in Q$, $\alpha \in \mathcal{D}^{\lambda(p)+1}$ is either an equality or a disequality test and $E \subseteq \{1, \dots, \lambda(p) + 1\}$. It is required that: (i) $\lambda(p) + 1 - |E| = \lambda(q)$, (ii) if $\alpha[i] = \alpha[\lambda(p) + 1]$ for some $i \leq \lambda(p)$, then $i \in E$, and (iii) T is deterministic, i.e. for all states p and \simeq -equivalence class c , there exists at most one transition (p, α, E, q) such that $\alpha \in c$.

For all $0 \leq i \leq k$, we let $Q_i = \lambda^{-1}(i)$ be the set of states with i available registers. Given this notation, we may freely denote a DRA as a tuple $\langle Q_0, \dots, Q_k, T, I, F \rangle$. A *configuration* is a pair $\tau = (q, \sigma)$, where $q \in Q$ and $\sigma \in \mathcal{D}^{\lambda(q)}$. Given two configurations (p, σ) and (q, σ') and a data $a \in \mathcal{D}$, we write $(p, \sigma) \xrightarrow{a}_{\mathcal{A}} (q, \sigma')$ whenever there exists a transition (p, α, E, q) such that $\sigma.d \simeq \alpha$, and $\sigma' = \text{drop}_E(\sigma.d)$ where $\text{drop}_E(\cdot)$ replaces, for all $i \in E$, the i th data of $\sigma.d$ by ϵ , and for all $i \in \{1, \dots, k\} \setminus E$, keeps the i th data.

A run of \mathcal{A} over a data word w is a sequence of configurations $(p_1, \sigma_1) \dots (p_n, \sigma_n)$ such that $n = |w| + 1$, and for all $1 \leq i < n$, $(p_i, \sigma_i) \xrightarrow{w[i]}_{\mathcal{A}} (p_{i+1}, \sigma_{i+1})$. We write $(p, \sigma_1) \xrightarrow{w}_{\mathcal{A}} (p_n, \sigma_n)$ to denote the existence of such a run. The language recognized by the register automaton \mathcal{A} , denoted $\mathcal{L}(\mathcal{A})$, is the set of words w such that $(q_0, \epsilon) \xrightarrow{w}_{\mathcal{A}} (p, \sigma)$ for some $p \in F$.

► **Example 3.** Consider the language of data words $L = \{d_1 d'_1 d_2 d'_2 \dots d_n d'_n \in \mathbb{N}^* \mid \forall 1 \leq i \leq n, d_i = d'_i\}$. The data language L is recognizable by the DRA $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$, where $k = 2$, $Q = \{q_0, q_1\}$, $\lambda(q_0) = 0$, $\lambda(q_1) = 1$, $F = \{q_0\}$ and $T = \{(q_0, 1, \emptyset, q_1), (q_1, 11, \{1, 2\}, q_0)\}$.

The language $L_{\neq} = \{(d_1 d_2)^n \mid n, d_1, d_2 \in \mathbb{N}, n > 0, d_1 \neq d_2\}$ of Introduction is recognizable by $\mathcal{A}' = \langle \{p_0, \dots, p_3\}, 2, \lambda', T', p_0, \{p_2\} \rangle$ where $\lambda'(p_0) = 0$, $\lambda'(p_1) = 1$, $\lambda'(p_2) = \lambda'(p_3) = 2$, and $T' = \{(p_0, 1, \emptyset, p_1), (p_1, 12, \emptyset, p_2), (p_2, 121, \{1\}, p_3), (p_3, 121, \{1\}, p_2)\}$.

We say that a data language L is *regular* if it is recognized by a DRA \mathcal{A} , i.e. $\mathcal{L}(\mathcal{A}) = L$. Note that regular data languages are equivariant, i.e. stable under data renaming, because the property of ρ to be a run on some data word w only depends on the data equalities in w .

3 Warm Up: Passive Learning of DFA

In this section, we recall how passive learning of DFA is achieved by the RPNI algorithm [34]. However we provide an alternative presentation, inspired by a similar presentation in the context of ω -regular languages [8]. RPNI first constructs a tree-like DFA accepting exactly the set of positive samples, and to generalize them, it merges the states of the initial DFA as much as possible and in a specific order, while preserving the rejection of negative samples. In contrast, our approach starts from a single-state DFA and adds transitions incrementally. When adding a new transition, to generalize the positive samples, it tries to reuse an existing state while rejecting negative samples, otherwise it creates a new state. This approach, which we believe is slightly simpler than the original RPNI algorithm, and offers the same guarantees, is easier to generalize to register automata, as it is not based on state-merging. It is indeed not clear how to define such an operation on register automata.

Some useful notions on DFA. Before defining the algorithm, we introduce some notions for DFA. We denote a DFA over an alphabet Σ as a tuple $\mathcal{A} = \langle Q, q_0, F, \delta \rangle$, where Q is the set of states with initial state q_0 , $F \subseteq Q$ are the final states and $\delta : Q \times \Sigma \rightarrow Q$ is the transition function (which might be partial). The language accepted by \mathcal{A} is denoted by $L(\mathcal{A})$. We also denote (if it exists) by $\delta^*(p, w)$ the state reached by \mathcal{A} when reading a word w from a state p . In particular, $\delta^*(p, \epsilon) = p$. Given a set $S \subseteq \Sigma^*$, a word $w \in \Sigma^*$, we define its *residual language* $w^{-1}S = \{w' \in \Sigma^* \mid w.w' \in S\}$.

We now define a relation on DFA, which characterizes when a DFA is a subgraph of another one. Given two DFAs $\mathcal{A}_i = \langle Q_i, q_0^i, F_i, \delta_i \rangle$, $i = 1, 2$, we say that \mathcal{A}_2 *completes* \mathcal{A}_1 , written $\mathcal{A}_1 \preceq \mathcal{A}_2$, if there exists an injective morphism $\Phi : Q_1 \rightarrow Q_2$, i.e. a mapping which preserves the initial state, final states and the transitions: $\Phi(q_0^1) = q_0^2$, $\Phi(F_1) \subseteq F_2$ and for all transitions $(p, \sigma, q) \in \delta_1$, we have $(\Phi(p), \sigma, \Phi(q)) \in \delta_2$. Given a sample set $S = (I^+, I^-)$ where $I^+, I^- \subseteq \Sigma^*$, we say that a DFA \mathcal{A} is *S-consistent* if $I^+ \subseteq L(\mathcal{A})$ and $I^- \cap L(\mathcal{A}) = \emptyset$. We say that \mathcal{A} is *S-completable* if there exists \mathcal{A}' such that $\mathcal{A} \preceq \mathcal{A}'$ and \mathcal{A}' is *S-consistent*.

► **Proposition 4.** *A DFA \mathcal{A} is S-completable iff $I^- \cap L(\mathcal{A}) = \emptyset$ and there do not exist $u_1, u_2, z \in \Sigma^*$ such that $u_1 z \in I^+$, $u_2 z \in I^-$ and $\delta^*(q_0, u_1) = \delta^*(q_0, u_2)$.*

As a corollary, DFA completable can be checked in PTIME, where we define the size $\|\mathcal{A}\|$ of a DFA \mathcal{A} as its number of states plus number of transitions.

► **Corollary 5.** *Given a DFA \mathcal{A} and a (finite) sample set S , it can be checked in time $O(|\mathcal{A}| \cdot |S|)$ whether \mathcal{A} is S -completable.*

Algorithm description. The algorithm is given as Algorithm 1. It takes as input a finite sample set $S = (I^+, I^-)$ of consistent positive and negative samples ($I^+ \cap I^- = \emptyset$), and returns an S -consistent DFA \mathcal{A} . It starts with a single state DFA \mathcal{A} and keeps on extending it with new transitions (possibly creating new states), to be able to read more and more prefixes of words from S . To do so, it adds all the prefixes of words from $I^+ \cup I^-$ (but ϵ) to a waiting list *ToRead*, processed in length-lexicographic order $<_{lex}$. The invariants of the algorithm (preserved by the while-loop at line 3) are:

- (i) any word w in *ToRead* cannot be read fully by \mathcal{A} , i.e. $\delta^*(q_0, w)$ is undefined.
- (ii) \mathcal{A} is S -completable.
- (iii) \mathcal{A} is $(I^+ \setminus \textit{ToRead}, I^-)$ -consistent.

It is easily seen that those invariants are initially true, because the sample set is consistent. At any iteration the algorithm picks a $<_{lex}$ -minimal word $w = ua$ in *ToRead*, with $a \in \Sigma$. Since w is length-minimal, $\delta^*(q_0, u) = p$ exists. The algorithm then adds a new transition from p on reading a . To do so, it calls a function `SET_TRANSITION` which first tries to reuse an existing state q such that adding transition $p \xrightarrow{a} q$ preserves S -compleatability (reusing existing states is how the algorithm *generalizes* the samples). If no such state exists, it creates the fresh state pa and adds the transition $p \xrightarrow{a} pa$. After the transition is added, all words in *ToRead* which can now be read are removed from *ToRead* (thus preserving invariant (i)), and if additionally they are positive, the state they reached are set to be accepting. Invariant (ii) is preserved because `SET_TRANSITION` makes sure the new transition can be added w/o breaking S -compleatability. If invariant (iii) is not preserved, since line 11 makes sure all words in $I^+ \setminus \textit{ToRead}$ are accepted (for the new set *ToRead*), there are some $w \in I^+$ and $w' \in I^-$ which both reach the same accepting state, contradicting S -compleatability and (ii).

The algorithm terminates in a polynomial number of steps with respect to the size of S , and returns an S -consistent DFA, according to invariant (iii). Note that it is not specified in which order states are enumerated at line 15. Different orders may yield different DFA, but this has no influence if the sample is rich enough, as explained in the next paragraph.

Completeness. A way to formalize how well RPNI generalizes the samples is given by a completeness result: for any regular language L , the minimal (not necessarily complete) DFA \mathcal{A}_L recognizing L is output by RPNI, if a small (polynomial size in the size of \mathcal{A}_L) but characteristic enough sample set is provided as input. We obtain the same result for our modified RPNI algorithm. We do not formally prove it because it is a particular case of the same result on DRA, fully proven in Section 5, and the purpose of this section is to convey intuitions easing the comprehension of the DRA setting. We now define a characteristic sample set S_L of polynomial size in the size of \mathcal{A}_L , such that Algo. 1, on input S_L , returns a DFA isomorphic to \mathcal{A}_L . We give the intuitions on how S_L influences the execution of Algorithm 1. For the algorithm to create as many states and transitions as \mathcal{A}_L , *ToRead* needs to contain at least one word per state and transition. Then, when it tries to add a new transition, negative samples are needed to control the target state: by an adequate choice of negative samples, exactly one target state (either existing or fresh) will be picked.

It is convenient to view \mathcal{A}_L as the quotient DFA obtained from the Myhill-Nerode congruence \equiv_L , defined by $u \equiv_L v$ if for all $w \in \Sigma^*$, $uw \in L \leftrightarrow vw \in L$ holds. Let $[u]$ be the class of any word u . For any two different classes c, c' , there exists $w_{c,c'} \in \Sigma^*$ which distinguishes c and c' , in the sense that $u_c w \in L \not\leftrightarrow u_{c'} w \in L$. Then, for u, v and $a \in \Sigma$

■ **Algorithm 1** Alternative formulation of RPNI algorithm for DFA.

Input: A (finite) sample set $S = (I^+, I^-)$ of positive and negative samples over an alphabet Σ : $I^+, I^- \subseteq \Sigma^*$ such that $I^+ \cap I^- = \emptyset$.

Output: An S -consistent DFA $\mathcal{A} = \langle Q, q_0, F, \delta \rangle$

```

1  $q_0 \leftarrow \epsilon$ ;  $Q \leftarrow \{\epsilon\}$ ;  $\delta \leftarrow \emptyset$ ; if  $\epsilon \in I^+$  then  $F \leftarrow \{q_0\}$  else  $F \leftarrow \emptyset$ 
2  $ToRead \leftarrow \text{Prefs}(I^+ \cup I^-) \setminus \{\epsilon\}$ 
3 while  $ToRead \neq \emptyset$  do
4    $u \cdot a \leftarrow$  length-lexicographic minimal word in  $ToRead$ 
5    $p \leftarrow \delta^*(q_0, u)$  // guaranteed to exist
6    $(p, a, q) \leftarrow \text{SET\_TRANSITION}(\mathcal{A}, p, a, S)$ 
7    $\delta \leftarrow \delta \cup \{(p, a, q)\}$ ;  $Q \leftarrow Q \cup \{q\}$ 
8   foreach  $w \in ToRead$  do
9     if  $\delta^*(q_0, w)$  exists then
10        $ToRead \leftarrow ToRead \setminus \{w\}$ 
11       if  $w \in I^+$  then  $F \leftarrow F \cup \{\delta^*(q_0, w)\}$ ;
12 return  $\mathcal{A} = \langle Q, q_0, F, \delta \rangle$ 
13
14 Function SET_TRANSITION( $\mathcal{A}, p, a, S$ ):
   Input: A DFA  $\mathcal{A} = \langle Q, q_0, F, \delta \rangle$ , state  $p \in Q$  and character  $a \in \Sigma$  such that
      $\delta(p, a)$  is undefined and  $\mathcal{A}$  is  $S$ -completable
   Output: A state  $q$  (either in  $Q$  if it exists, otherwise fresh) such that
      $\langle Q, q_0, F, \delta \cup \{p \xrightarrow{a} q\} \rangle$  is  $S$ -completable
15 foreach  $q \in Q$  do
16   if COMPLETABLE( $\langle Q, q_0, F, \delta \cup \{p \xrightarrow{a} q\} \rangle, S$ ) then return  $(p, a, q)$ 
   // check  $S$ -completability, see Cor.5
17 return  $(p, a, pa)$ 

```

such that $[ua] \neq [v]$, samples are needed to forbid the learning algorithm to create the transition $[u] \xrightarrow{a} [v]$. We use the word $w_{[ua],[v]}$ to do so. Formally, a sample set $S = (I^+, I^-)$ is *characteristic for L* if it is L -consistent and there exist $St, Tr, D \subseteq \Sigma^*$ such that:

- St contains for each class $c \in \Sigma^*/\equiv_L$, a $<_{lex}$ -minimal representative u , i.e. $[u] = c$,
- for all $u \in St$ and $a \in \Sigma$, $ua \in Tr$,
- for all $u, v \in St$, $a \in \Sigma$, if $[ua] \neq [v]$ then $uaw_{[ua],[v]}, vw_{[ua],[v]} \in D$,

and such that $(St \cup Tr \cup D) \cap L \subseteq I^+$ and $(St \cup Tr \cup D) \cap \bar{L} \subseteq I^-$.

Given a characteristic sample set S_L , the invariant maintained by Algorithm 1 on the while-loop is that \mathcal{A}_L completes the DFA \mathcal{A} constructed so far. This is ensured by the fact that \mathcal{A} is constructed incrementally by adding new transitions, and by the samples in D which prevents some wrong transitions to be created. Since $ToRead$ initially contains at least one representative sample per states and transitions of \mathcal{A}_L , the final automaton \mathcal{A} returned by Algorithm 1 has the same number of states and transitions as \mathcal{A}_L , and by the invariant, we get that it is isomorphic to \mathcal{A}_L . Moreover, there exists a characteristic sample set of polynomial size in the size of \mathcal{A}_L : there is a representative u_c of any class c of linear length in the number of states of \mathcal{A}_L , while the distinguishing words $w_{c,c'}$ are bounded, using classical pumping arguments, quadratically in the number of states of \mathcal{A}_L . This can even be improved to a linear upper bound [18].

4 Learning Register Automata from Positive and Negative Samples

Let \mathcal{D} be a data domain. A (finite) *sample set* is a pair $S = (I^+, I^-)$ such that $I^+, I^- \subseteq \mathcal{D}^*$ are finite sets of respectively positive and negative samples. S is *consistent* whenever for all $u \in I^+, v \in I^-$, we have $u \not\preceq v$. In this section we introduce an RPNI-like passive learning algorithm for DRA from consistent sample sets. We start by giving some intuition on the key ingredients underlying the algorithmic process. It follows the same structure as the DFA learning algorithm presented before.

The algorithm initially constructs a DRA \mathcal{A} being able just to read the empty word ϵ (and accept it if $\epsilon \in I^+$). Moreover, the non-empty samples prefixes are collected into a set, called *ToRead* (just as for the DFA case of Section 3): namely, *ToRead* maintains the sample prefixes not yet readable by the DRA \mathcal{A} under construction. The algorithm keeps on adding transitions to be able to read such samples prefixes, until *ToRead* is empty (iterating over *ToRead* by increasing samples in $<_{lex}$ order). The following crucial invariants are maintained during the overall execution:

- (i) \mathcal{A} is a DRA able to read each word in $\text{Prefs}(I^+ \cup I^-) \setminus \text{ToRead}$
- (ii) \mathcal{A} accepts (resp. rejects) each sample in $I^+ \setminus \text{ToRead}$ (resp. I^-)
- (iii) It is possible to “complete” \mathcal{A} (adding new states and transitions) into a DRA accepting any sample of I^+ and rejecting all samples of I^- (the latter DRA is said to be *S-consistent*).

Those invariants are clearly met by the initial single-state DRA. We now formalize the notion of being *S-completable* for DRA. As for the DFA case, one needs a notion of embedding of DRA into another. For $i = 1, 2$, let $\mathcal{A}_i = \langle Q_i, k_i, \lambda_i, T_i, q_0^i, F_i \rangle$ be a DRA. We say that \mathcal{A}_2 completes \mathcal{A}_1 , written $\mathcal{A}_1 \preceq \mathcal{A}_2$, whenever there exists an injective mapping $\Phi : Q_1 \rightarrow Q_2$ which preserves the initial state, the availability function, the final states, and the transitions, as follows: $\lambda_1 = \lambda_2 \circ \Phi$, $\Phi(F_1) \subseteq F_2$, $\Phi(q_0^1) = q_0^2$, and for all transitions $(p, \alpha, E, q) \in T_1$, there exists $\beta \simeq \alpha$ such that $(\Phi(p), \beta, E, \Phi(q)) \in T_2$. In particular, observe that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$. We say that \mathcal{A}_1 is *S-completable* whenever there exists an *S-consistent* DRA \mathcal{A}_2 completing \mathcal{A}_1 . This notion is decidable in PTIME. To prove it, we first give a characterization of *S-completability*, which can be proved in the same line as the DFA case:

► **Proposition 6.** *A DRA \mathcal{A} is S-completable for a sample set $S = (I^+, I^-)$, iff $L(\mathcal{A}) \cap I^- = \emptyset$ and there do not exist words $w \in I^+, z \in I^-$, state q and words σ, σ' such that: $w = w_1 w_2 \wedge z = z_1 z_2 \wedge (q_0, \epsilon) \xrightarrow{w_1}_{\mathcal{A}} (q, \sigma) \wedge (q_0, \epsilon) \xrightarrow{z_1}_{\mathcal{A}} (q, \sigma') \wedge \sigma w_2 \simeq \sigma' z_2$.*

By inspecting I^+, I^- and computing the states reached by their prefixes in \mathcal{A} , it is not difficult to decide the latter characterization in PTIME:

► **Corollary 7.** *It is decidable in PTIME whether for a given DRA \mathcal{A} is S-completable.*

The prefixes in *ToRead* are considered in $<_{lex}$ -increasing order and used to add a transition to \mathcal{A} allowing their processing without violating the three invariants above. This can be done since \mathcal{A} is *S-completable*. Therefore, by the end of the algorithm \mathcal{A} is an *S-consistent* DRA (due to the first two invariants and the fact that *toRead* is empty).

With this intuition in mind, we go in more details introducing the pseudocode of our DRA passive learning algorithm. A careful reader may notice a clear symmetry between the DFA and DRA learning processes presented, and may wonder how we take care of the registers when creating new transitions. This will be made clear in the pseudocode presentation below.

Pseudocode of the DRA Passive Learning Algorithm. The pseudocode in Algorithm 2 illustrates the procedure `DRA_PASSIVE_LEARN(S)`, which takes as input a consistent finite sample set $S = (I^+, I^-)$ and returns an S -consistent DRA \mathcal{A} . A full execution of the algorithm is provided as Example 8.

A while-loop follows the initialization phase in lines (1)-(2), controlled by a guard checking whether the set of samples prefixes *ToRead* is not empty. In that case, the DRA under construction is not yet capable to process all the words in S . Hence, a $<_{lex}$ -minimal word ua is extracted from *ToRead* at line (4). Invariant (i) and the minimal length of ua ensures that \mathcal{A} can read u but not ua . As \mathcal{A} is deterministic, it admits a unique run $\rho : \langle q_0, \epsilon \rangle \xrightarrow{u}_{\mathcal{A}} \langle q, \sigma \rangle$ processing u . The S -completeness of \mathcal{A} ensures that it is possible to add a transition from state q such that ua is now readable, while preserving the invariants. To generalize the samples, the choice of that transition is done in such a way that it tries to keep the least amount of data in memory and tries to reuse existing states. Such a choice is delegated to the procedure `SET_TRANSITION` at line (6), which returns a new transition $t = (q, \sigma a, E, p)$. Before describing this procedure, we finish the overview of Algorithm 2. Line (7) augments \mathcal{A} by adding to T the new transition t . If the target state q' is new, line (8) updates consistently the set of states of the DRA under construction and the availability function. Finally, the for-loop at line (9) updates *ToRead* by removing the prefixes that can now be read, and updates the final states consistently based on positive samples which can now be read by \mathcal{A} .

We now describe the procedure `SET_TRANSITION` whose pseudocode is given in Algorithm 3. It takes as input a consistent sample set $S = (I^+, I^-)$, an S -completable DRA \mathcal{A} , a state q , and a word $\sigma \cdot a \in \mathcal{D}^{\lambda(q)+1}$ such that \mathcal{A} has no transition be able to read a from configuration (q, σ) . It computes a set of registers E to be erased, and a new target state p , and returns transition $(q, \sigma a, E, p)$, with the guarantee that adding this new transition to \mathcal{A} preserves S -completeness, while trying to generalize the samples. To compute E , it first tries to determine if some stored values can be dropped, while preserving S -completeness. Then it attempts to reuse an existing state p as the target of a transition $(q, \sigma a, E, p)$, otherwise creates a fresh new state. When computing E , it tries to erase as many stored values as possible, which informally allows for more generalization of the samples. Another fundamental reason for doing so is because it is needed to be able to prove that our algorithm identifies regular data languages in the limit. Since the characteristic sample somehow encodes the behaviours of the minimal automaton, which stores the least amount of data in its configurations, the algorithm as well has to follow the same policy, to be able to output the minimal canonical automaton for the language. We now give more details.

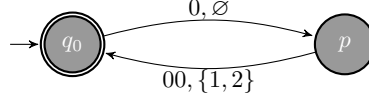
If a is registered in the word σ at position j (i.e. $\sigma(j) = a$) then the set of erasable registers E is initialized to $\{j\}$ otherwise to the empty set. Line (3) initializes the set R (of registers to check for erasability) to $\{1, \dots, \lambda(q) + 1\} \setminus E$. The loop at line (4) (taken $|R|$ times) extracts one-by-one the registers in R and checks whether their values can be safely erased. It relies on a function `Choose(R)` which picks a register from R . In each iteration the current register h chosen from R is recognized as erasable only if the DRA obtained by adding the transition $t = (q, \sigma a, E \cup \{h\}, f)$ towards a fresh new state f does not compromise S -completeness. In that case h is added to E (and never removed). The task of checking if adjoining the transition t to \mathcal{A} leads to an S -completable DRA \mathcal{A}' is performed through a call to a function `COMPLETABLE(\mathcal{A}', I^+, I^-)`, which exists by Corollary 7.

Once the set of erasable registers E for the new transition is defined, the function `SET_TRANSITION` proceeds with a loop through the states $p \in Q$ such that $\lambda(p) = \lambda(q) + 1 - |E|$ at line (8), aiming at identifying the target of the new transition among existing states. To check that p can be reused, the algorithm calls `COMPLETABLE(\mathcal{A}', I^+, I^-)`

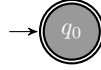
10:10 Passive Learning of Regular Data Languages in Polynomial Time and Data

to determine whether the DRA \mathcal{A}' obtained by adding the transition $(q, \sigma a, E, q')$ to \mathcal{A} is S -completable. In that case the function returns the transition $(q, \sigma a, E, p)$. If no vertex is identified as a safe target for the new transition within the loop at line (8), a fresh state $f \notin Q$ is created and the function returns the transition $(q, \sigma a, E, f)$ at line (11).

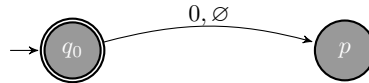
► **Example 8** (RPNI Execution for DRA). Consider the language $L = \{\epsilon\} \cup \{d_1 d'_1 d_2 d'_2 \dots d_n d'_n \in \mathbb{N}^* \mid \forall 1 \leq i \leq n, d_i = d'_i\}$ recognizable by the the DRA $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$, where $k = 2$, $Q = \{q_0, p\}$, $\lambda(q_0) = 0$, $\lambda(p) = 1$, $F = \{q_0\}$ and $T = \{(q_0, 0, \emptyset, p), (p, 00, \{1, 2\}, q_0)\}$, as depicted below:



We provide a complete example of execution of the proposed DRA learner, given the sample-set $S = (I^+ = \{\epsilon, 00, 11\}, I^- = \{0, 011, 01\})$, which eventually returns the latter DRA. The RPNI algorithm builds an initial DRA \mathcal{A} composed by the only initial state q_0 , which is inserted in the final states since $\epsilon \in I^+$. Then, the set $ToRead = \{0, 1, 00, 01, 11, 011\}$ of non-empty sample-prefixes is constructed. The initial DRA is:



The first execution of the main loop extracts the \langle_{lex} -minimum sample $ua = 0$ from $ToRead$ and calls $SET_TRANSITION(\mathcal{A}, q_0, \sigma a = 0, I^+, I^-)$ to build a transition t out from q_0 (reached by \mathcal{A} upon reading $u = \epsilon$) processing the value $a = 0$. $SET_TRANSITION(\mathcal{A}, q_0, \sigma a = 0, I^+, I^-)$ detects that a needs to be remembered, given that the DRA \mathcal{A}' obtained by adding to \mathcal{A} the transition (q_0, a, \emptyset, p) (toward the fresh state p) is not completable since $00 \in I^+$, $01 \in I^-$. Therefore, the set of erasable registers for the transition $(q_0, a, E, _)$ is set to the empty set (i.e. $E = \emptyset$). There is no state in \mathcal{A} with $(\lambda(q_0) + 1) - |E| = (0 + 1) - 0 = 1$ registers. Hence, the second loop at line 8 does not perform any iteration and $SET_TRANSITION(\mathcal{A}, q_0, \sigma a = 0, I^+, I^-)$ returns the transition $t = (q_0, a, \emptyset, p)$ toward the fresh state p . The DRA under construction is then updated to $\mathcal{A} = \langle \{q_0, p\}, 1, \lambda, \{t\}, \{q_0\} \rangle$, where $\lambda(q_0) = 0$, $\lambda(p) = 1$. The final for-loop at line 9 extracts from $ToRead$ the sample 0 and the sample 1 that are now readable by \mathcal{A} and does not label p as final because there is no positive sample readable by \mathcal{A} leading to p . Therefore, at the beginning of the second iteration of the main loop at line 3, we have that $\mathcal{A} = \langle \{q_0, p\}, 1, \lambda, \{t\}, \{q_0\} \rangle$ and $ToRead = \{00, 01, 11, 011\}$.



The second execution of the main loop extracts the \langle_{lex} -minimum sample $ua = 00$ from $ToRead$ and calls $SET_TRANSITION(\mathcal{A}, p, \sigma a = 00, I^+, I^-)$ to build a transition t out from p (reached by \mathcal{A} upon reading $u = 0$) processing the value $a = 0$. Line 5 of $SET_TRANSITION$ inserts the index 1 in E since $\sigma(1) = a = 0$ (avoiding the storage of duplicates). The automaton \mathcal{A}' obtained adding to \mathcal{A} the transition $(p, 00, E = \{1, 2\}, s)$, where s is a fresh state is completable. Therefore E is set to $\{1, 2\}$ at the end of the first (and only) iteration of the loop at line 4. Since \mathcal{A} contains only one state with $(\lambda(p) + 1) - |E| = (1 + 1) - 2 = 0$ registers, also the loop at line 8 iterates only once, attempting

at redirecting $(p, 00, E = \{1, 2\}, s)$ toward q_0 . Such a redirection succeeds since the DRA \mathcal{A}' obtained by adding to \mathcal{A} the transition $(p, 00, E = \{1, 2\}, q_0)$ is completable. Hence $\text{SET_TRANSITION}(\mathcal{A}, p, \sigma a = 00, I^+, I^-)$ returns $t' = (p, 00, \{1, 2\}, q_0)$. The DRA under construction is then updated to $\mathcal{A} = \langle \{q_0, p\}, 1, \lambda, \{t, t'\}, \{q_0\} \rangle$, where $\lambda(q_0) = 0, \lambda(p) = 1$. Since \mathcal{A} is able to read all the samples provided as input, accepting them if they belong to I^+ , the final for-loop at line 9 extracts all the samples from *ToRead* and does not modify the set of final states. The algorithm then terminates and returns a DRA recognizing L .

► **Remark 9.** In the algorithm SET_TRANSITIONS , there are several places where choices can be made: at line 5 when a register is chosen, and later at line 8 when target states p are enumerated. Different implementations of those choice and enumeration functions might result in differing DRA in general. However, our learning algorithm's guarantees are independent of those implementations, namely: 1) the algorithm returns an S -consistent DRA, and 2) for a regular language L , given a characteristic sample it returns a minimal DRA for L .

■ **Algorithm 2** Algorithm $\text{DRA_PASSIVE_LEARN}(S)$ for learning a deterministic register automaton from a set of positive and negative samples $S = (I^+, I^-)$.

Input: A finite consistent sample set $S = (I^+, I^-)$.
Output: An S -consistent DRA $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$, i.e. $I^+ \subseteq L(\mathcal{A})$ and $I^- \cap L(\mathcal{A}) = \emptyset$.

```

/* define an initial S-completable DRA A */
1 Q ← {ε}; λ(ε) = 0; T ← ∅; q₀ ← ε; F ← ∅ ;if ε ∈ I⁺ then F ← {q₀}
2 ToRead ← (prefs(I⁺ ∪ I⁻)) \ {ε}
3 while ToRead is not empty do
4   w = ua ← <uex>-minimal word in ToRead
5   ⟨q, σ⟩ ← configuration such that ⟨q₀, ε⟩  $\xrightarrow{u}$   $\mathcal{A}$  ⟨q, σ⟩
6   (q, σa, E, p) ← SET_TRANSITION(ℳ, q, σa, I⁺, I⁻)
7   T ← T ∪ {(q, σa, E, p)}
8   if p ∉ Q then Q ← Q ∪ {p}; λ(p) ← λ(q) + 1 - |E|
9   for w ∈ ToRead do
10    if ⟨q₀, ε⟩  $\xrightarrow{w}$   $\mathcal{A}$  ⟨s, σ⟩ then
11     ToRead ← ToRead \ {w}
12     if ∃w' ∈ I⁺, w' ≃ w then F ← F ∪ {s}
13 return ℳ = ⟨Q, k = max_{q ∈ Q} λ(q), λ, T, q₀, F⟩

```

We now prove the correctness of our passive DRA learning algorithm. Lemma 10 below proves the validity of the crucial invariants briefly introduced in the previous section.

► **Lemma 10.** *The following invariants hold at line (3) (entrance of the main loop) of the algorithm $\text{DRA_PASSIVE_LEARN}(S)$, where $S = (I^+, I^-)$:*

- Inv1: \mathcal{A} is a DRA accepting (resp. rejecting) all words in $I^+ \setminus \text{ToRead}$ (resp. in I^-)
- Inv2: \mathcal{A} can read all words in $\text{Prefs}(I^+ \cup I^-) \setminus \text{ToRead}$
- Inv3: \mathcal{A} is S -completable

Sketch. The invariants are clearly true before the while-loop has been entered for the first time (for the 3rd invariant, it is because S is consistent, so we can always built an DRA which accepts exactly I^+ and its \simeq -equivalent words, and rejects all words from I^- and their \simeq -equivalent words).

■ **Algorithm 3** Function $\text{SET_TRANSITION}(\mathcal{A}, q, \sigma a, I^+, I^-)$ used by DRA learner to complete the new transition $(q, \sigma \cdot a, _, _)$ defining the erasable registers and the target node.

```

1 Function SET_TRANSITION( $\mathcal{A}, q, \sigma \cdot a, I^+, I^-$ ):
  | Input:  $S$ -completable DRA  $\mathcal{A} = \langle Q, k, \lambda, T, q_0, F \rangle$ , state  $q \in Q$ , and word
  |    $\sigma \cdot a \in \mathcal{D}^{\lambda(q)+1}$  such that  $T$  does not contain any transition  $(q, \sigma \cdot a, \_, \_)$ 
  | Output: New  $t = (q, \sigma a, E, p)$  s.t.  $\langle Q, k, \lambda, T \cup \{t\}, q_0, F \rangle$  is  $S$ -completable
  | /* Compute some set  $E \subseteq \{1, \dots, \lambda(q) + 1\}$  of erasable registers */
2  $E \leftarrow \emptyset; i \leftarrow \lambda(q)$ ;
3 if  $\exists j \leq i : \sigma(j) = a$  then  $E \leftarrow \{j\}; R \leftarrow \{1 \dots i + 1\} \setminus \{j\}$  else  $R \leftarrow \{1 \dots i + 1\}$ ;
4 while  $R \neq \emptyset$  do
5   |  $h \leftarrow \text{Choose}(R); R \leftarrow R \setminus \{h\}; E \leftarrow E \cup \{h\}$ ;
6   |  $\mathcal{A}' \leftarrow \langle Q \cup \{f\}, k, \lambda \cup \{f \mapsto i + 1 - |E|\}, T \cup \{(q, \sigma a, E, f)\}, q_0, F \rangle$  for  $f$  fresh;
7   | if  $\neg(\text{COMPLETABLE}(\mathcal{A}', I^+, I^-))$  then  $E \leftarrow E \setminus \{h\}$ ;
  | /* Pick some  $p \in Q$ , if it exists, as target of the new transition */
8 foreach  $p \in Q$  such that  $\lambda(p) = \lambda(q) + 1 - |E|$  do
9   |  $\mathcal{A}' \leftarrow \langle Q, k, \lambda, T \cup \{(q, \sigma a, E, p)\}, q_0, F \rangle$ ;
10  | if  $\text{COMPLETABLE}(\mathcal{A}', I^+, I^-)$  then return  $(q, \sigma a, E, p)$ ;
  | /* Otherwise create a fresh new state as target */
11 return  $(q, \sigma a, E, f)$  for  $f$  a fresh state ;

```

The inductive step is rather simple. For Inv2, this is precisely the essence of our algorithm: it picks any word $ua \in \text{ToRead}$ which cannot be read fully (while u can) and completes \mathcal{A} with one transition allowing to read ua . Inv3 is clear because calls to COMPLETABLE in SET_TRANSITION make sure that S -completeness is preserved. For Inv1, the loop at line (9) updates the accepting states and so it makes sure that all words of I^+ which can be read are accepted. The only possible issue is that it could now accept words from I^- . It is not possible since SET_TRANSITION ensures that adding the new transition preserves S -completeness (given an S -completable DRA, which is ensured by the induction hypothesis). If a word of I^- is now accepted, it means that some accepting state is now reachable by a positive word and a negative word, contradicting S -completeness, ensured by Inv3. ◀

On the ground of Lemma 10, we are ready to prove the correctness and complexity of our DRA passive learning algorithm.

► **Theorem 11.** *The algorithm $\text{DRA_PASSIVE_LEARN}(S)$ returns a DRA \mathcal{A} consistent with $S = (I^+, I^-)$ (i.e. $L(\mathcal{A}) \supseteq I^+ \wedge L(\mathcal{A}) \cap I^- = \emptyset$) in time polynomial w.r.t. the size of S .*

Proof. Correctness is trivial by Inv1 and the fact that ToRead is eventually empty when the algorithm terminates. For complexity, note that the algorithm takes as input a consistent sample set. Consistency can be checked in PTIME by Lemma 1. Note that all the loops in DRA_PASSIVE_LEARN and SET_TRANSITION are iterated only a polynomial number of times in the size of their inputs. Moreover, COMPLETABLE runs in PTIME by Cor. 7. ◀

► **Remark 12.** Note that correctness of SET_TRANSITION is trivial since it calls COMPLETABLE to make sure that the returned transition can be added while preserving S -completeness. The fact that SET_TRANSITION tries to erase some registers and to reuse existing states is important for *generalizing* the samples, not for correctness, and important for proving that our learning algorithm identifies regular data languages in the limit. Moreover, $\langle u_{ex}$ -minimality at line 4 could be replaced by just length minimality while preserving correctness, but it is important to prove completeness.

5 Identification in the Limit of the Class of Regular Data Languages

In this section we prove that the proposed RPNI algorithm for DRA identifies in the limit the class of regular data languages, according to the definition of [23, 16, 8]. Namely, given a regular data language L , we show that there exists a characteristic sample S_L such that $\text{DRA_PASSIVE_LEARN}(S)$ learns a DRA recognizing L whenever $S_L \subseteq S$. We then show that it can be chosen of polynomial size in the size of the minimal DRA recognizing L . As for the DFA case, we rely on the existence of a unique minimal DRA for L , as shown in [6]. We recall this notion here.

Memorable data and canonical register automata. Any regular data language L admits a unique minimal DRA, up to DRA isomorphism², as shown in [6]. Minimality is both for the number of states, as well as the number of stored data, in the sense that when the canonical automaton reads a prefix, it stores the least amount of data. The key notion towards canonicity and minimality is based on the notion of *memorable data* with respect to L . Given a word $w \in \mathcal{D}^*$ and two data $a, b \in \mathcal{D}$, we let $w[a/b] = \mu_{a/b}(w)$ where $\mu_{a/b}(a) = b$ and $\mu_{a/b}(d) = d$ for all $d \neq a$. Intuitively, a data a in a word w is memorable if for some continuation u , renaming data a in u by some data b which does not occur in wu has an influence on the membership into L .

► **Definition 13** (Memorable Values [6]). *A data $a \in \mathcal{D}$ in a word w is L -memorable for a language L if there exist $u \in \mathcal{D}^*$ and $b \in \mathcal{D}$ such that: $wu \simeq (wu)[a/b]$ and $wu \neq_L w(u[a/b])$.*

Given a word w and a language L on \mathcal{D} , we let $\text{mem}_L(w)$ denote the finite sequence of all L -memorable data of w ordered according to the positions of their last occurrences in w : a occurs before b in $\text{mem}_L(w)$ if they are both memorable in w , and the last occurrence of a in w is before the last occurrence of b in w . Intuitively, $\text{mem}_L(w)$ represents the data that necessarily need to be stored by any DRA recognizing L . We now define an equivalence $\equiv_L \subseteq \mathcal{D}^* \times \mathcal{D}^*$ which is of finite index iff L is DRA-recognizable. We let $w \equiv_L w'$ if $|\text{mem}_L(w)| = |\text{mem}_L(w')|$ and for all words u, u' , if $\text{mem}_L(w)u \simeq \text{mem}_L(w')u'$ then $wu =_L w'u'$. When \equiv_L is of finite index, it is used to define a DRA recognizing L (the canonical DRA \mathcal{A}_L) that is minimal amongst all DRA recognizing L [6].

► **Theorem 14** ([6]). *For any DRA-recognizable language L , there exists a unique DRA \mathcal{A}_L (up to isomorphism) recognizing L , which is both state-minimal and register-minimal (in the sense that the number of stored values in any configuration is minimal).*

We recall the construction of $\mathcal{A}_L = \langle Q, k, \lambda, T, q_0, F \rangle$. The set of states is $Q = \mathcal{D}/\equiv_L$ with for all $u \in \mathcal{D}^*$, $\lambda([u]) = |\text{mem}_L(u)|$ and $k = \max_{q \in Q} \lambda(q)$. The initial state is $q_0 = [e]$ and $F = \{[u] \mid u \in L\}$. Finally, for all classes $c \in \mathcal{D}/\equiv_L$, pick a representative u such that $[u] = c$. For all $d \in \mathcal{D}$, pick a word³ α such that $\text{mem}_L(u)d \simeq \alpha$, then set $([u], \alpha, E, [ud]) \in T$ where E is defined such that $\text{mem}_L(ud) = \text{drop}_E(\text{mem}_L(u)d)$. The later automaton is well-defined [6]. Even though there are infinitely many minimal automata up to isomorphism, in the sequel we refer to \mathcal{A}_L as a representative of one of them.

Characteristic samples. As for the DFA case, one needs samples to account for the states and transitions of the minimal automaton, and to distinguish equivalence classes. In addition, samples are also needed to witness memorable data. A sample set $S = (I^+, I^-)$ is said to be L -consistent if $I^+ \subseteq L$ and $I^- \cap L = \emptyset$.

² Two DRA $\mathcal{A}_1, \mathcal{A}_2$ are isomorphic if $\mathcal{A}_1 \preceq \mathcal{A}_2$ and $\mathcal{A}_2 \preceq \mathcal{A}_1$, see Section 4

³ Different choices for the representative u and α yield isomorphic DRA.

► **Definition 15** (Characteristic sample). *Let L be a regular data language. A sample set $S = (I^+, I^-)$ is characteristic for L if it is L -consistent and there exist $St, Tr, Mem, D \subseteq \mathcal{D}^*$ such that:*

- *St contains, for each class $c \in \mathcal{D}/\equiv_L$, a $<_{lex}$ -minimal representative u , i.e. $[u] = c$,*
 - *for all $w \in St, a \in mem_L(w), d \notin mem_L(w)$ s.t. d is $<_{\mathcal{D}}$ -minimal: $wa, wd \in Tr$,*
 - *for all $w \in Tr$, all $a \in mem_L(w)$, there exists $b \in \mathcal{D}$ and $u \in \mathcal{D}^*$ such that $wu \simeq (wu)[a/b]$, $wu \not\equiv_L w(u[a/b])$ and $wu, w(u[a/b]) \in Mem$,*
 - *for all $w \in St, z \in Tr$ such that $|mem_L(w)| = |mem_L(z)|$ and $w \not\equiv_L z$, there exist w', z' such that $mem_L(w)w' \simeq mem_L(z)z'$, $ww' \not\equiv_L zz'$ and $ww', zz' \in D$,*
- and $(St \cup Tr \cup Mem \cup D) \cap L \subseteq I^+$ and $(St \cup Tr \cup Mem \cup D) \cap \bar{L} \subseteq I^-$.*

We now establish the following theorem:

► **Theorem 16.** *Let L be a regular data language and let S_L be a characteristic sample for L . Then $\text{DRA_PASSIVE_LEARN}(S_L)$ returns a DRA isomorphic to the minimal DRA \mathcal{A}_L .*

Sketch. The main arguments are however rather intuitive and similar to the DFA case, except that additional samples are needed to account for memorable data. In the definition of characteristic sample, St and Tr allow the learner to identify the states and the transitions of \mathcal{A}_L . Instead, the learner uses the samples in Mem within the first loop of SET_TRANSITION to avoid dropping necessary registers. Finally, the learner uses the samples in D within the second loop of SET_TRANSITION to set correctly the target of new transitions avoiding wrong redirections toward old states.

The proof in Theorem 16 uses an inductive argument on the number of iterations of the main loop in $\text{RA_Passive_Learning}$ to establish that, whenever the guard-condition at line (3) is checked, the DRA \mathcal{A} under construction (given a characteristic sample for L) is isomorphic to a portion of the minimal DRA \mathcal{A}_L . In other words \mathcal{A}_L completes \mathcal{A} , i.e. $\mathcal{A} \preceq \mathcal{A}_L$. When processing a new prefix $w = ua$ in ToRead , where w leads to a configuration (q, σ) of the so far constructed DRA, we know by IH that $\sigma = mem_L(u)$. The call to SET_TRANSITION adds a new transition $(q, \sigma a, E, p)$, and we have to prove that $(q, \sigma a, E, p)$ can be embedded into a transition $(q^{\otimes}, \alpha^{\otimes}, E^{\otimes}, p^{\otimes})$ of \mathcal{A}_L . By IH and the definition of \mathcal{A}_L , we know that u leads to configuration $(q^{\otimes}, \sigma = mem_L(u))$ by \mathcal{A}_L . Therefore, $\alpha^{\otimes} \simeq mem_L(u)a$. Again by definition of \mathcal{A}_L , E^{\otimes} drops only unmemorable data of ua . By definition of characteristic samples, there are samples in Mem that prevent dropping memorable data of ua , so the first phase of SET_TRANSITIONS will keep only the necessary data, i.e. $E = E^{\otimes}$. Finally, the samples in D ensures that $(q, \sigma a, E)$ can be directed towards a unique state p . For any other state, calls to COMPLETABLE are failing. This state p is then embedded into state p^{\otimes} of \mathcal{A}_L to show that \mathcal{A} extended with the new transition $(q, \sigma a, E, p)$ is completable by \mathcal{A}_L .

When $\text{DRA_PASSIVE_LEARN}(S_L)$ returns, it yields a DRA \mathcal{A} where $\mathcal{A} \preceq \mathcal{A}_L$. Moreover, since in S , there is at least one representative sample per state and transition of \mathcal{A}_L , \mathcal{A} has as many states and transitions as \mathcal{A}_L , and therefore $\mathcal{A}_L \preceq \mathcal{A}$, i.e., \mathcal{A} and \mathcal{A}_L are isomorphic. ◀

Characteristic samples of polynomial size. We conclude this section by showing that given a regular data language L , there exists a characteristic sample S_L (see Definition 15) of polynomial size in the size of the minimal DRA \mathcal{A}_L for L . Let us give a brief overview of the proof. For the samples representing the states and transitions of $\mathcal{A}(L)$ (sets St and Tr), it is rather easy, via pumping arguments, to show that samples of lengths polynomial in the number of states of \mathcal{A}_L are sufficient. It relies on the next lemma, proved by similar arguments as in the case of finite memory automata [27].

► **Lemma 17.** *Let \mathcal{A} be a DRA with n states such that $L(\mathcal{A}) \neq \emptyset$. Then there exists w such that $w \in L(\mathcal{A})$ and $|w|$ is bounded by n .*

Given a data word u , we let $u^{-1}L = \{v \mid uv \in L\}$ be the residual language of L by u . Bounding *Mem* is similar to bounding *D*. We recall that for *Mem*, we need two samples of the form wu and $w(u[a/b])$ for any $w \in St$ and $a \in mem_L(w)$, such that $wu \neq_L w(u[a/b])$. We show that $wu \neq_L (w[a/b])u$ since b is fresh in wu . Since samples $w \in St$ have been bounded already, it remains to bound u , i.e. show that the inequality $w^{-1}L \neq (w[a/b])^{-1}L$ is witnessed by a polysize word u . Similarly, for *D*, given $w \in St$ and $z \in Tr$, we have to bound w', z' such that $mem_L(w)w' \simeq mem_L(z)z'$ and $ww' \neq_L zz'$. We prove that the latter is equivalent to the existence of a data permutation τ such that $z' = \tau(w')$ and $\tau(w)z' \neq_L zz'$. So, we only need to bound z' satisfying $\tau(w)z' \neq_L zz'$, i.e. show that the inequality $(\tau(w))^{-1}L \neq z^{-1}L$ is witnessed by a polysize word z' . In general, for a DRA \mathcal{A} recognizing a language K and a configuration κ , we let $\kappa^{-1}K$ be the set of words on which \mathcal{A} has an accepting run starting in κ . The polysize of *Mem* and *D* is then a consequence of:

► **Theorem 18.** *Given a DRA \mathcal{A} recognizing a language K and two configurations κ_1, κ_2 , if $\kappa_1^{-1}K \neq \kappa_2^{-1}K$, then there is $w \in \mathcal{D}^*$ of polynomial length such that $w \in \kappa_1^{-1}K \not\leftrightarrow w \in \kappa_2^{-1}K$.*

A low-hanging consequence of the latter theorem is that DRA equivalence is solvable in CONP, which is new for this model of DRA. Note that for DFA, proving a result like Theorem 18 is easy by using Boolean operations on DFA and short witness for DFA non-emptiness. However in the data setting, intersection of DRA involves an exponential blow-up, for instance to maintain that registers should hold distinct values⁴. Instead, to prove Theorem 18, we show that given two configurations of a DRA, if they are not *bisimilar*, then this is witnessed by a polynomial word. We rely on group-theoretic methods as developed for fresh register automata in [33], which we adapt to the DRA model defined in this paper. Unfortunately, it does not seem that DRA can be transformed in PTIME into any of the models of [33], thus preventing us from directly applying those results. We reprove them with a slight adaptation to our setting. We now give an overview of the procedure.

The space of configurations of a DRA form a labelled transition system (where the data are labels of the transitions). Bisimulation is defined classically, see e.g. [4], and is seen as an Attacker/Defender game: Attacker picks a data and transition, Defender replies by a transition on the same data, etc. In the deterministic case, a strategy for Attacker is just a data word. We prove that if Attacker has a strategy, i.e. a word, it has a small strategy (of poly-length in the size of the DRA). To show the latter, the concrete bisimulation relation, which is infinite, is finitely but completely abstracted, based on the observation that the data themselves are not important, only equalities between data matter. Based on this, any pair of configurations $(q_1, \alpha_1), (q_2, \alpha_2) \in Q \times \mathcal{D}^\ell$ is abstracted as a triple (q_1, π, q_2) where $\pi : [1, \ell] \leftrightarrow [1, \ell]$ is a partial permutation s.t. $\pi(i) = j$ iff $\alpha_1[i] = \alpha_2[j]$. A relation is then defined over such triples, called *symbolic bisimulation*, which reflects in an abstract way the moves of Attacker and Defender in the concrete bisimulation game. It is then shown that two configurations are bisimilar iff their abstraction as a triple is in the symbolic bisimulation (in other words, the abstraction is sound and complete).

The symbolic bisimulation relation can be computed as the greatest fixpoint of a monotonic function F , generating a decreasing chain of intermediate relations $\mathcal{U}_0 \supseteq \dots \supseteq \mathcal{U}_n = \mathcal{U}_{n+1}$ converging to the symbolic bisimulation relation in n steps. From this chain, it is possible to

⁴ We conjecture the exponential intersection blow-up is unavoidable.

extract for any non-bisimilar configurations, a word strategy for Attacker of length at most n . The group-theoretic arguments come into play to bound polynomially n , as introduced in [33]. All the sets \mathcal{U}_i bear a lot of symmetries: for example if $(q_1, \pi, q_2) \in \mathcal{U}_i$ for some i , then $(q_2, \pi^{-1}, q_1) \in \mathcal{U}_i$ (closure by inverse). In fact, this is precisely where the fact that registers hold distinct data is crucial: it implies that π is a permutation and not an arbitrary relation, and so it can be inverted. The sets \mathcal{U}_i enjoy other closures (such as composition) allowing one to show that some characteristic subset of permutations of \mathcal{U}_i form a subgroup of the symmetric group, ordered by the subgroup relation for increasing i . The final argument follows from [3]: chains of subgroups of the symmetric group are linearly bounded.

6 Future work

While we have shown that our learning algorithm runs in PTIME, there are interesting questions related to implementation and in particular efficient data structures, for instance to deal with sample prefixes, and to quickly check for completability. As mentioned in Remark 9, it is worth investigating heuristics for choosing target states of new created transitions when there are multiple candidates, for example adaptations of evidence-based heuristics [17] to the context of data languages. The same question arises when erasing registers: there might be several possible sets E .

An interesting future direction, related to the implementation of our algorithm, is to make it incremental, in the sense that it does not have to restart the whole procedure from scratch whenever a new example is added to the set of samples, in the spirit of [18].

Finally, a natural continuation is to investigate active learning for the register automata model of this paper. As mentioned in the introduction, all known results about active learning for regular data languages have exponential time complexity, but this model has not been investigated yet, to the best of our knowledge.

References

- 1 Parosh Aziz Abdulla, C. Aiswarya, and Mohamed Faouzi Atig. Data communicating processes with unreliable channels. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 166–175. ACM, 2016. doi:10.1145/2933575.2934535.
- 2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmet Kara, and Othmane Rezine. Verification of dynamic register automata. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPICs*, pages 653–665. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.FSTTCS.2014.653.
- 3 László Babai. On the length of subgroup chains in the symmetric group. *Communications in Algebra*, 14(9):1729–1736, 1986. doi:10.1080/00927878608823393.
- 4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 5 Mrudula Balachander, Emmanuel Filiot, and Jean-François Raskin. LTL reactive synthesis with a few hints. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II*, volume 13994 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2023. doi:10.1007/978-3-031-30820-8_20.
- 6 Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In Alberto H. F. Laender and Laks V. S. Lakshmanan, editors, *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Buenos Aires, Argentina, May 17-20, 2010*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. URL: <https://ceur-ws.org/Vol-619/paper11.pdf>.

- 7 Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines using domains with equality tests. In José Luiz Fiadeiro and Paola Inverardi, editors, *Fundamental Approaches to Software Engineering, 11th International Conference, FASE 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4961 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2008. doi:10.1007/978-3-540-78743-3_24.
- 8 León Bohn and Christof Löding. Constructing deterministic ω -automata from examples by an extension of the RPNI algorithm. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPICs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.MFCS.2021.20.
- 9 León Bohn and Christof Löding. Passive learning of deterministic büchi automata by combinations of dfas. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 114:1–114:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.114.
- 10 Mikolaj Bojanczyk. Orbit-finite sets and their algorithms (invited talk). In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 1:1–1:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.1.
- 11 Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.
- 12 Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 7–16. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.51.
- 13 Benedikt Bollig. An automaton over data words that captures EMSO logic. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2011. doi:10.1007/978-3-642-23217-6_12.
- 14 Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Model checking languages of data words. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7213 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 2012. doi:10.1007/978-3-642-28729-9_26.
- 15 Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducer. *Mach. Learn.*, 66(1):33–67, 2007. doi:10.1007/s10994-006-9613-8.
- 16 Colin de la Higuera. Characteristic sets for polynomial grammatical inference. *Mach. Learn.*, 27(2):125–138, 1997. doi:10.1023/A:1007353007695.
- 17 Colin de la Higuera, José Oncina, and Enrique Vidal. Identification of DFA: data-dependent vs data-independent algorithms. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, September 25-27, 1996, Proceedings*, volume 1147 of *Lecture Notes in Computer Science*, pages 313–325. Springer, 1996. doi:10.1007/BFb0033365.
- 18 Pierre Dupont. Incremental regular inference. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, pages 222–237. Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/BFb0033357.

- 19 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. *Formal Methods Syst. Des.*, 61(2):290–337, 2022. doi:10.1007/s10703-023-00435-w.
- 20 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: <https://lmcs.episciences.org/7279>.
- 21 Dana Fisman, Hadar Frenkel, and Sandra Zilles. Inferring symbolic automata. *Log. Methods Comput. Sci.*, 19(2), 2023. doi:10.46298/lmcs-19(2:5)2023.
- 22 Pedro García and Jose Oncina. Inference of recognizable tree sets. *Tech. rep., Departamento de Sistemas Informáticos y Computación, Universidad de Alicante. DSIC-II/47/93*, 1993.
- 23 E. Mark Gold. Language identification in the limit. *Inf. Control.*, 10(5):447–474, 1967. doi:10.1016/S0019-9958(67)91165-5.
- 24 Radu Grigore and Nikos Tzevelekos. History-register automata. *Log. Methods Comput. Sci.*, 12(1), 2016. doi:10.2168/LMCS-12(1:7)2016.
- 25 Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2012. doi:10.1007/978-3-642-27940-9_17.
- 26 Malte Isberner, Falk Howar, and Bernhard Steffen. Learning register automata: from languages to program structures. *Mach. Learn.*, 96(1-2):65–98, 2014. doi:10.1007/s10994-013-5419-7.
- 27 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 28 Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 25:1–25:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.25.
- 29 Martin Leucker. Learning meets verification. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7-10, 2006, Revised Lectures*, volume 4709 of *Lecture Notes in Computer Science*, pages 127–151. Springer, 2006. doi:10.1007/978-3-540-74792-5_6.
- 30 Damián López and Pedro García. On the inference of finite state automata from positive and negative data. *Topics in Grammatical Inference*, pages 73–112, 2016.
- 31 Amaldev Manuel, Anca Muscholl, and Gabriele Puppis. Walking on data words. *Theory Comput. Syst.*, 59(2):180–208, 2016. doi:10.1007/s00224-014-9603-3.
- 32 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 613–625. ACM, 2017. doi:10.1145/3009837.3009879.
- 33 Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Bisimilarity in fresh-register automata. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, pages 156–167, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.24.
- 34 Jose Oncina and Pedro García. Inferring regular languages in polynomial update time. *World Scientific*, January 1992. doi:10.1142/9789812797902_0004.
- 35 Nikos Tzevelekos. Fresh-register automata. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 295–306. ACM, 2011. doi:10.1145/1926385.1926420.

A Completeness of DFA

Before proving Proposition 4, we define the notion of prefix-tree acceptor of a finite set S , as a tree-shaped DFA accepting exactly S . For any finite set $S \subseteq \Sigma^*$, $\text{PTA}(S) = (Q = \text{Pref}(S), q_0 = \varepsilon, \delta, F = S)$, where $\forall wa \in \text{Pref}(S) : \delta(w, a) = wa$. Clearly, $L(\text{PTA}(S)) = S$.

► **Proposition 4.** *A DFA \mathcal{A} is S -completable iff $I^- \cap L(\mathcal{A}) = \emptyset$ and there do not exist $u_1, u_2, z \in \Sigma^*$ such that $u_1z \in I^+$, $u_2z \in I^-$ and $\delta^*(q_0, u_1) = \delta^*(q_0, u_2)$.*

Proof. (\implies) Clearly, if $I^- \cap L(\mathcal{A}) \neq \emptyset$, then \mathcal{A} is not completable, since by definition of completable, final states have to be preserved.

Now, let $w_1 = u_1z \in I^+$ and $w_2 = u_2z \in I^-$ such that $\delta_{\mathcal{A}}^*(q_0, u_1) = \delta_{\mathcal{A}}^*(q_0, u_2)$, then any S -consistent extension \mathcal{A}^{ext} of \mathcal{A} have the following property: $\delta_{\mathcal{A}^{\text{ext}}}^*(q_0, w_1) = \delta_{\mathcal{A}^{\text{ext}}}^*(q_0, w_2)$ and therefore, $w_1 \in L(\mathcal{A}^{\text{ext}})$ iff $w_2 \in \mathcal{A}^{\text{ext}}$, hence \mathcal{A}^{ext} is not S -consistent. Contradiction.

(\impliedby) We construct an S -consistent extension \mathcal{A}^{ext} of \mathcal{A} . For every state q of \mathcal{A} , let L_q be the set of words w such that $q_0 \xrightarrow{w} q$. For all letter $a \in \Sigma$ such that $\delta(q, a)$ is undefined, let $I_{q,a}^+ = (L_q a)^{-1} I^+$ be the set of words v such that $uav \in I^+$ for some $u \in L_q$. We construct $T_{q,a} = \text{PTA}(I_{q,a}^+)$ with root node t_ε and add a transition from q to the initial state of $T_{q,a}$ on reading a . The set of accepting states is set to be the accepting states of \mathcal{A} plus the states reached by words from I^+ . We make such construction for any pair (q, a) such that $\delta(q, a)$ is undefined. This results in an extension \mathcal{A}^{ext} of \mathcal{A} , which is now able to read any word of I^+ .

We now show \mathcal{A}^{ext} is S -consistent. The inclusion $I^+ \subseteq \mathcal{A}^{\text{ext}}$ is immediate by construction. Let us show $I^- \cap L(\mathcal{A}^{\text{ext}}) = \emptyset$. Suppose, for the sake of contradiction, that there is some $w' \in I^- \cap L(\mathcal{A}^{\text{ext}})$. Let $p = \delta_{\mathcal{A}^{\text{ext}}}^*(q_0, w')$. We consider two cases:

1. $p \notin Q_{\mathcal{A}}$, i.e. p is a state of some added PTA $T_{q,a}$. Hence there is some $w \in I^+$ such that $\delta_{\mathcal{A}^{\text{ext}}}^*(q_0, w) = p$. By construction of \mathcal{A}^{ext} (in particular because $T_{q,a}$ is tree-shaped), it implies that w and w' can be factorized as $w = u_1z$ and $w' = u_2z$ for some $u_1, u_2 \in \Sigma^*$, such that $\delta_{\mathcal{A}}^*(q_0, u_1) = \delta_{\mathcal{A}}^*(q_0, u_2) = q$ exists and in \mathcal{A}^{ext} , there exists a transition from q to the initial state of $T_{q,a}$ where a is the first letter of z . This contradicts our assumption, since $w = u_1z \in I^+$ while $w' = u_2z \in I^-$.
2. $p \in Q_{\mathcal{A}}$: since $I^- \cap L(\mathcal{A}) = \emptyset$, then p was not accepting in \mathcal{A} . Hence there must be some positive example $w \in I^+$ such that $q_0 \xrightarrow{w} p$. We then immediately get a contradiction, by taking $u_1 = w$, $u_2 = w'$ and $z = \varepsilon$. ◀

► **Corollary 5.** *Given a DFA \mathcal{A} and a (finite) sample set S , it can be checked in time $O(|\mathcal{A}| \cdot |S|)$ whether \mathcal{A} is S -completable.*

Proof. We check the characterization given by Proposition 4. Checking $I^- \cap L(\mathcal{A}) = \emptyset$ is done in $O(|I^-| \cdot |\mathcal{A}|)$. For the second condition, we check the non-existence of u_1, u_2, z as in Proposition 4. Let $\text{Suff}(S)$ be the set of suffixes of words of S . For any word $z \in \text{Suff}(S)$, the objective is to compute two sets P_z^+, P_z^- , where for $s \in \{+, -\}$, $P_z^s = \{\delta^*(q_0, u) \mid u \in \Sigma^* \wedge uz \in I^s\}$ (this set can be empty). Then, it suffices to check that no node $z \in \text{Suff}(S)$ satisfies $P_z^+ \cap P_z^- \neq \emptyset$.

To compute the sets P_z^+ and P_z^- for all $z \in \text{Suff}(S)$ in $O(|\mathcal{A}| \cdot |S|)$, the algorithm first builds a suffix tree based on S , whose set of nodes is $\text{Suff}(S)$. The root is ε , and if $u \in \Sigma^*$ and $\sigma u \in \Sigma^*$ are two nodes of the tree, there is an edge from u to σu . Therefore, if a node z is a common ancestor of two nodes u_1 and u_2 , then z is a common suffix of u_1 and u_2 . Constructing the tree is done in time $O(|S|)$.

The tree is then processed bottom-up, exploiting the following claim:

▷ **Claim.** For all $z \in \text{Suff}(S)$, all $s \in \{+, -\}$, $P_z^s = \{\delta(q, \sigma) \mid \sigma z \in \text{Suff}(S) \wedge q \in P_{\sigma z}^s\} \cup X$ where $X = \{q_0\}$ if $z \in I^s$, otherwise $X = \emptyset$.

Proof of the claim. Let denote by Q_z^s the rhs of the above equality. We show that $P_z^s = Q_z^s$. Let $p \in P_z^s$. Then $p = \delta^*(q_0, u)$ for some $uz \in I^s$. If $u = \epsilon$, then $p = q_0$ and $z \in I^s$, so $p \in X$ and $p \in Q_z^s$. If $u = u'\sigma$ for some u' , then $u'\sigma z \in I^s$ and $\sigma z \in \text{Suff}(S)$. Let $q = \delta^*(q_0, u')$. Then $p = \delta(q, \sigma)$ and we can conclude since $q \in P_{\sigma z}^s$ by definition of $P_{\sigma z}^s$.

Conversely, let $p \in Q_z^s$. There are two cases: either $p = q_0$ and $z \in I^s$, we get immediately that $p \in P_z^s$, or $p \in \{\delta(q, \sigma) \mid \sigma z \in \text{Suff}(S) \wedge q \in P_{\sigma z}^s\}$. Since $q \in P_{\sigma z}^s$, $q = \delta^*(q_0, u)$ for some u such that $u\sigma z \in I^s$. As $p = \delta(q, \sigma)$, we get that $p = \delta^*(q_0, u\sigma)$ and since $u\sigma z \in I^s$, we can conclude that $p \in P_z^s$. ◀

The algorithm is then immediate by processing the tree bottom-up, applying δ on the sets computed for the children of any node currently processed. The overall computation is in time $O(\|A\| \cdot \|S\|)$. ◀

B Completeness for DRA

► **Proposition 6.** A DRA \mathcal{A} is S -completable for a sample set $S = (I^+, I^-)$, iff $L(\mathcal{A}) \cap I^- = \emptyset$ and there do not exist words $w \in I^+$, $z \in I^-$, state q and words σ, σ' such that: $w = w_1 w_2 \wedge z = z_1 z_2 \wedge (q_0, \epsilon) \xrightarrow{w_1}_{\mathcal{A}} (q, \sigma) \wedge (q_0, \epsilon) \xrightarrow{z_1}_{\mathcal{A}} (q, \sigma') \wedge \sigma w_2 \simeq \sigma' z_2$.

Proof. (\Leftarrow) Let $W = \{s_1, \dots, s_n\} = I^+ \cup I^-$ and for all $0 \leq i \leq n$, let $W_i = \{s_1, \dots, s_i\}$, $I_i^+ = I^+ \cap W_i$, $I_i^- = I^- \cap W_i$ and $S_i = (I_i^+, I_i^-)$. Note that $S_0 = (\emptyset, I^-)$ and $S_n = S$. We show by induction on i how to build a sequence of DRA $\mathcal{A}_0 = \mathcal{A} \preceq \mathcal{A}_1 \preceq \dots \preceq \mathcal{A}_n$ such for each $0 \leq i \leq n$, \mathcal{A}_i is S_i -consistent and there does not exist $w \in I^+$, $z \in I^-$ and state q and words σ, σ' such that:

$$w = w_1 w_2 \wedge z = z_1 z_2 \wedge (q_0, \epsilon) \xrightarrow{w_1}_{\mathcal{A}_i} (q, \sigma) \wedge (q_0, \epsilon) \xrightarrow{z_1}_{\mathcal{A}_i} (q, \sigma') \wedge \sigma w_2 \simeq \sigma' z_2 \quad (1)$$

The base case $i = 0$ is clear by hypothesis, since $\mathcal{A}_0 = \mathcal{A}$. Let $i > 1$ and assume by IH that \mathcal{A}_{i-1} is S_{i-1} -consistent. The DRA $\mathcal{A}_i = \langle Q_i, k_i, \lambda_i, T_i, q_0^i, F_i \rangle$ is obtained from $\mathcal{A}_{i-1} = \langle Q_{i-1}, k_{i-1}, \lambda_{i-1}, T_{i-1}, q_0^{i-1}, F_{i-1} \rangle$ as follows. We consider two cases:

- (1) If \mathcal{A}_{i-1} reads s_i reaching a configuration (q, σ) then let q be a final state iff $s_i \in I^+$. By contradiction suppose that \mathcal{A}_i is not S_i -consistent. Then, there exists $s_{j < i}$ such that both reading s_i and reading s_j , \mathcal{A}_{i-1} gets to a configuration where the first component is q , say $(q, \sigma), (q, \delta)$, however only one between s_i, s_j is a positive sample. This contradicts the fact that \mathcal{A}_{i-1} respects condition (1), since $\sigma \simeq \delta$ (by definition of DRA).
- (2) Otherwise, let decompose s_i into $s_i = s_i^1 s_i^2$ where s_i^1 is the longest prefix of s_i which can be read by \mathcal{A}_{i-1} , and let (q, σ) be the configuration reached by \mathcal{A}_{i-1} upon reading s_i^1 . To obtain \mathcal{A}_i , we complete \mathcal{A}_{i-1} with new transitions from state q , so that \mathcal{A}_i can only read s_i^2 from q (and all \simeq -equivalent words). Let $a_1 \dots a_m = s_i^2$, $\sigma_0 = \sigma$, and for all $1 \leq k \leq m$, $\sigma_k = \text{drop}_{E_k}(\sigma_{k-1} a_k)$ where $E_k = \{x\}$ such that $a_k = \sigma_{k-1}[x]$ if it exists, otherwise $E_k = \emptyset$. We now construct \mathcal{A}_i . Let $p_1, \dots, p_m \notin Q_{i-1}$ be fresh new states. Let $Q_i = Q_{i-1} \cup \{p_1 \dots p_m\}$ and $p_0 = q$, $T_i = T_{i-1} \cup \{(p_k, \sigma_k \cdot a_{k+1}, E_{k+1}, p_{k+1}) \mid 0 \leq j < m\}$, $q_0^i = q_0^{i-1}$, $F_i = F_{i-1} \cup \{p_m\}$ if $s_i \in I^+$, otherwise $F_i = F_{i-1}$.

We first show that \mathcal{A}_i meets Condition (1). If it is not the case, then, there exists a positive sample $u_1 = w_1 z_1$ and a negative sample $u_2 = w_2 z_2$ and runs $(q_0^i, \epsilon) \xrightarrow{w_1}_{\mathcal{A}_i} (t, \lambda_1)$ and $(q_0^i, \epsilon) \xrightarrow{w_2}_{\mathcal{A}_i} (t, \lambda_2)$ such that $\lambda_1 z_1 \simeq \lambda_2 z_2$. Since \mathcal{A}_{i-1} satisfies Condition (1), necessarily, state t is a newly added state, so one of $\{p_1, \dots, p_m\}$, say $t = p_j$ for some j . So, w_1 and w_2 can be further decomposed into $w_1' w_1''$ and $w_2' w_2''$ with $|w_1'| = |w_2'|$, and the runs into:

$$\begin{aligned} (q_0^i, \epsilon) &\xrightarrow{w_1'}_{\mathcal{A}_{i-1}} (q, \lambda_1) \xrightarrow{w_1''}_{\mathcal{A}_i} (p_j, \lambda_1) \\ (q_0^i, \epsilon) &\xrightarrow{w_2'}_{\mathcal{A}_{i-1}} (q, \lambda_2) \xrightarrow{w_2''}_{\mathcal{A}_i} (p_j, \lambda_2) \end{aligned}$$

By definition of \mathcal{A}_i , $\sigma a_1 \dots a_j \simeq \lambda_1' w_1''$ and $\sigma a_1 \dots a_j \simeq \lambda_2' w_2''$ (where σ, a_1, \dots, a_m have been used before to define \mathcal{A}_i), hence $\lambda_1' w_1'' \simeq \lambda_2' w_2''$. Now, again by definition of \mathcal{A}_i , λ_1 (resp. λ_2) is made of exactly one occurrence of each data appearing in $\lambda_1' w_1''$ (resp. $\lambda_2' w_2''$). From this fact, the fact that $\lambda_1 z_1 \simeq \lambda_2 z_2$ and $\lambda_1' w_1'' \simeq \lambda_2' w_2''$, we get that $\lambda_1' w_1'' z_1 \simeq \lambda_2' w_2'' z_2$, contradicting the fact that \mathcal{A}_{i-1} respects Condition (1).

It remains to show that \mathcal{A}_i is S_i -consistent. By construction, \mathcal{A}_i accepts all words in I_i^+ , because \mathcal{A}_i accepts all words in I_{i-1}^+ , by IH, and we have not removed accepting states, only added potentially one accepting state to accept s_i . We show that \mathcal{A}_i rejects all words in I_i^- . If it were not the case, then let $s \in I_i^-$ accepted by \mathcal{A}_i . As \mathcal{A}_{i-1} is S_{i-1} -consistent, it is necessarily the case that s has an accepting run towards a new accepting state, and this new accepting state can only be p_m , and $s_i \in I^+$. From this we immediately get that Condition (1) is not satisfied, which is a contradiction.

(\Rightarrow) Let $\mathcal{A}' = (Q', k', \lambda', T', q_0', F')$ be the S -consistent DRA that completes \mathcal{A} and let $\Phi : Q \rightarrow Q'$ the injective function witnessing it. If $L(\mathcal{A}) \cap I^- \neq \emptyset$, since Φ must preserve accepting states, then $L(\mathcal{A}') \cap I^- \neq \emptyset$, which contradicts that \mathcal{A}' is S -consistent. On the other hand, suppose that there exist $w = w_1 w_2 \in I^+, z = z_1 z_2 \in I^-$ such that $(q_0, \epsilon) \xrightarrow{w_1}_{\mathcal{A}} (q, \sigma) \wedge (q_0, \epsilon) \xrightarrow{z_1}_{\mathcal{A}} (q, \sigma') \wedge \sigma w_2 \simeq \sigma' z_2$. Then, \mathcal{A}' admits runs $(\Phi(q_0), \epsilon) \xrightarrow{w_1}_{\mathcal{A}'} (\Phi(q), \sigma) \xrightarrow{w_2}_{\mathcal{A}'} (p, \delta) \wedge (\Phi(q_0), \epsilon) \xrightarrow{z_1}_{\mathcal{A}'} (\Phi(q), \sigma') \xrightarrow{z_2}_{\mathcal{A}'} (p, \delta')$. If $p \in F'$, then \mathcal{A}' accepts both w and z , otherwise it rejects them, contradicting its S -consistency. \blacktriangleleft

Left-Linear Rewriting in Adhesive Categories

Paolo Baldan  

Department of Mathematics, University of Padua, Italy

Davide Castelnovo  

Department of Mathematics, University of Padua, Italy

Andrea Corradini  

Department of Computer Science, University of Pisa, Italy

Fabio Gadducci  

Department of Computer Science, University of Pisa, Italy

Abstract

When can two sequential steps performed by a computing device be considered (causally) independent? This is a relevant question for concurrent and distributed systems, since independence means that they could be executed in any order, and potentially in parallel. Equivalences identifying rewriting sequences which differ only for independent steps are at the core of the theory of concurrency of many formalisms. We investigate the issue in the context of the double pushout approach to rewriting in the general setting of adhesive categories. While a consolidated theory exists for linear rules, which can consume, preserve and generate entities, this paper focuses on left-linear rules which may also “merge” parts of the state. This is an apparently minimal, yet technically hard enhancement, since a standard characterisation of independence that – in the linear case – allows one to derive a number of properties, essential in the development of a theory of concurrency, no longer holds. The paper performs an in-depth study of the notion of independence for left-linear rules: it introduces a novel characterisation of independence, identifies well-behaved classes of left-linear rewriting systems, and provides some fundamental results including a Church-Rosser property and the existence of canonical equivalence proofs for concurrent computations. These results properly extends the class of formalisms that can be modelled in the adhesive framework.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Semantics and reasoning

Keywords and phrases Adhesive categories, double-pushout rewriting, left-linear rules, switch equivalence, local Church-Rosser property

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.11

Related Version *Full Version:* <https://arxiv.org/abs/2407.06181>

Funding The research has been partially supported by the European Union – NextGenerationEU under the National Recovery and Resilience Plan (NRRP) – Call PRIN 2022 PNRR – Project P2022HXNSC “Resource Awareness in Programming: Algebra, Rewriting, and Analysis”, by the Italian MUR – Call PRIN 2022 – Project 20228KXFN2 “Spatio-Temporal Enhancement of Neural nets for Deeply Hierarchical Automated Logic” and by the University of Pisa – Call PRA 2022 – Project 2022_99 “Formal Methods for the Healthcare Domain based on Spatial Information”.

1 Introduction

One of the key pay-off of concurrency theory is the idea that the behaviour of a computational device can be modelled by abstracting its sequences of steps via a suitable equivalence. Intuitively, the equivalence captures when steps are causally unrelated and thus could be executed in any order, and possibly in parallel. Two seminal contributions to this line of research have been Mazurkiewicz’s traces [37] and Winskel’s event structures [39]. Concerning



© Paolo Baldan, Davide Castelnovo, Andrea Corradini, and Fabio Gadducci;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

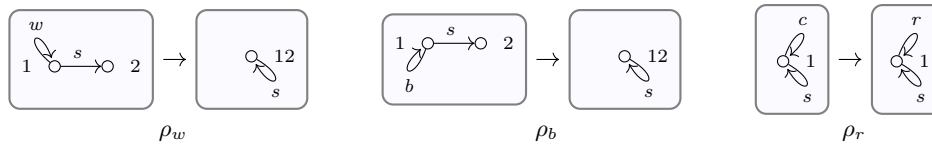
Editors: Rupak Majumdar and Alexandra Silva; Article No. 11; pp. 11:1–11:24



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

11:2 Left-Linear Rewriting in Adhesive Categories



■ **Figure 1** Rewriting rules for the coffee example.

formalisms based on the rewriting paradigm, concurrency often boils down to having a notion of independence between two consecutive steps. Noteworthy examples are the interchange law [38] in term rewriting and permutation equivalence [35] in λ -calculus.

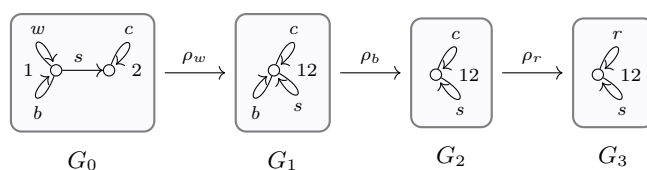
Among rewriting-based formalisms, those manipulating graph-like structures proved useful in many settings for representing the dynamics of distributed systems: the graph is used to represent the entities and their relations within states, while rewriting steps, modifying the graph, model computation steps that result in changes to the system state. The DPO (double-pushout) approach [25] is nowadays considered a standard one for these structures, thanks to its locality (rule application has a local effect on a state, differently from more recent approaches like SqPO [17] or PBPO⁺ [40]) and to its flexibility: the rules allow to specify that, in a rewriting step, some entities in the current state are removed, some others are needed for the rewriting step to occur but remain unaltered, and some new entities can be generated. Concerning concurrency, the chosen notion is switch (also referred to as shift) equivalence, and independence is formally captured by what is called the Church-Rosser theorem of DPO rewriting [16, Chapter 3, Section 3.4].

A noteworthy advantage of DPO rewriting is that it can be formulated over adhesive categories and their variants [26, 32], which include, among others, toposes [31] and string diagrams [9]. Operating within the framework of adhesive categories allows one to devise the foundations of a rewriting theory that can be then instantiated to the context of interest, thus avoiding the need of repeatedly proving similar ad-hoc results for each specific setting.

So far, most of the theoretical results focused on *linear rules*: intuitively, a step is required only to consume, preserve and generate entities. However, in some situations it is also necessary to “merge” parts of the state. This happens in the context of string diagrams mentioned above, but also in graphical implementations of nominal calculi where, as a result of name passing, the received name is identified with a local one at the receiver [18, 27], or in the visual modelling of bonding in biological/chemical processes [41]. A similar mechanism is at the core of e-graphs (equality graphs), used for rewrite-driven compiler optimisations [42]: rather than merging nodes corresponding to equal expressions, the idea – conceptually and, as we will see, also mathematically similar – is to maintain an equivalence over the nodes.

Technically, to acquire the expressiveness needed for capturing fusions, i.e. the merge of some elements in the state, requires to move from linear to left-linear rewriting rules. While left-linear rules have been considered in various instances (mainly in categories of graphs) and some results concerning the corresponding rewriting theory have been put forward [6, 23, 24], the phenomena arising when devising a theory of independence and concurrency for rewriting with left-linear rules in the general setting of adhesive and quasi-adhesive categories have not been systematically explored, even if an investigation in the setting of quasi-topoi, considering non-linear rules yet restricting their applicability, has been presented in [7, 8].

Just to get some basic intuition about left-linear rewriting systems, their capabilities and the problems that can arise, assume we want to model a situation in which a coffee comes with a bag of white sugar and a bag of brown sugar. The situation is represented by the graph G_0 in Fig. 2: the loop c is the coffee, the two loops b and w represent the white and



■ **Figure 2** A sequence of rewriting steps.

brown sugar. The corresponding nodes are connected to the coffee node by an s -labelled edge to indicate that they can be possibly added to the coffee. The available rules are in Fig. 1. Rules ρ_w and ρ_b model the addition of white and brown sugar, respectively, to the coffee. Note that adding one kind of sugar, say x , to the coffee is realised by deleting loop x and merging the corresponding node with the coffee node so that a self-loop s is created. Once some sugar has been added, the coffee is ready for drinking. This is represented by rule ρ_r , which checks for the presence of sugar, represented by the loop s , in the coffee, and brings it to the r state, ready for drinking.

A rewriting sequence is depicted in Fig. 2, applying ρ_w , ρ_b , and ρ_r in sequence, thus producing a coffee with both white and brown sugar, ready for drinking. A first interesting observation is that after applying rule ρ_w to G_0 , thus producing graph G_1 , we can still apply rule ρ_b , with a non-injective match. Intuitively, once the coffee is ready for drinking because some sugar has been added, adding more sugar will not change its state. Indeed, the rules could also be applied in reverse order. Should they be considered independent? Moreover, since rule ρ_r just requires the presence of sugar, it could be applied immediately after ρ_w . This might lead to think that ρ_b and ρ_r are independent. However, this is not completely clear, since in absence of ρ_w , the application of ρ_b would be essential to enable ρ_r .

In this paper we perform a systematic exploration of these phenomena at the level of \mathcal{M} -adhesive categories, which encompass a large family of DPO-based formalisms.

As an initial step, we observe that for left-linear rules some fundamental properties that play a role in the concurrency theory of linear rewriting need to be deeply revised, or plainly do not hold. Firstly, the notion of sequential independence and the Church-Rosser theorem, ensuring that sequential independent steps can be performed in reverse order, does not hold out of the box. Even worse, given two independent steps it might be possible to switch them in several different ways. As we will observe, this prevents the development of a sensible theory of concurrency, since switches are used as a basis for equating sequences of rewriting steps that differ only in the order of independent steps.

We single out a class of left-linear rewriting systems, referred to as *well-switching rewriting systems*, where sequential independence ensures existence and uniqueness of the switch, and argue that this is the right setting for dealing with left-linear rewriting systems.

On the one hand, they capture most categories of interest for rewriting. Indeed, we identify general classes of left-linear rewriting systems on graph-like structures that are well-switching. As a sanity check, we prove that linear rewriting systems are well-switching.

On the other hand, we argue that a concurrency theory of rewriting can be developed for well-switching rewriting systems, recovering, sometimes in weakened form, most of the results that hold in the linear case. The development faces a main technical obstacle. In the literature on linear rewriting systems, a central result is the characterisation of switch equivalence of sequences of steps based on what are called processes or consistent permutations [2, 4, 15, 30]. In turn this is the key to derive some fundamental properties for linear rewriting, namely

1. *globality of independence*: if two independent steps are moved forward or backward due to the application of switchings to a sequence, they remain independent;
2. *consistency of switching*: when transforming sequences of rewriting steps into equivalent ones by switching independent steps, the result does not depend on the order in which switchings are applied, but only on the associated permutation.

Properties (1) and (2) allow one to deduce a canonical form for equivalence proofs between sequences of rewriting steps seen as concurrent computations.

The fact that for left-linear rewriting systems the characterisation of switch equivalence via consistent permutations fails, and thus cannot be used for proving (1) and (2), leads to asking if these properties hold. We show that much of the theory can be retained: globality of independence holds in a weaker form, conceptually linked to the fact that fusions introduce a form of disjunctive causality, while consistency of switching holds unchanged. Finally, we prove that a canonical form for switching sequences can also be recovered. These results represent a solid basis for developing a satisfactory theory of concurrency for left-linear rules.

Synopsis. In §2 we recall the basic definitions of DPO rewriting for \mathcal{M} -adhesive categories. In §3 we present the key novelties of our proposal, introducing the standard notion of independence and a novel axiomatic notion of switchability, showing that the former fails to imply the latter for left-linear rewriting systems. We then define well-switching rewriting systems, a subclass of left-linear rewriting systems, and we prove that the classical results for linear rewriting systems are recovered for such a class. Finally, in §4 we outline directions for future work. Appendix A recalls basic results on adhesive categories and Appendix B presents *strong enforcing* rewriting systems, a class of systems where the local Church-Rosser Theorem holds. Appendix C of the extended version provides the proofs of our results.

2 \mathcal{M} -adhesive categories and rewriting systems

This section recalls the basic theory of \mathcal{M} -adhesive categories [1, 20, 21, 30, 32]. Given a category \mathbf{X} we will not distinguish notationally between \mathbf{X} and its class of objects, so “ $X \in \mathbf{X}$ ” means that X is an object of \mathbf{X} . We let $\text{Mor}(\mathbf{X})$, $\text{Mono}(\mathbf{X})$ and $\text{Reg}(\mathbf{X})$ denote the class of all arrows, monos and regular monos of \mathbf{X} , respectively. Given an integer $n \in \mathbb{Z}$, $[0, n]$ denotes the set of natural numbers less than or equal to n ; in particular, $[0, n] = \emptyset$ if $n < 0$.

2.1 \mathcal{M} -adhesivity

The key property of \mathcal{M} -adhesive categories is the *Van Kampen condition* [10, 31, 32]. Let \mathbf{X} be a category. A subclass \mathcal{A} of $\text{Mor}(\mathbf{X})$ is called

- *stable under pushouts (pullbacks)* if for every pushout (pullback) square as the one below, if $m \in \mathcal{A}$ ($n \in \mathcal{A}$) then $n \in \mathcal{A}$ ($m \in \mathcal{A}$);
- *closed under composition* if $h, k \in \mathcal{A}$ implies $h \circ k \in \mathcal{A}$ whenever h and k are composable.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 m \downarrow & & \downarrow n \\
 C & \xrightarrow{g} & D
 \end{array}$$

► **Definition 2.1** (Van Kampen property). *Let \mathbf{X} be a category and consider the diagram below. Given a class of arrows $\mathcal{A} \subseteq \text{Mor}(\mathbf{X})$, we say that the bottom square is an \mathcal{A} -Van Kampen square if*

1. *it is a pushout square;*
2. *whenever the cube above has pullbacks as back and left faces and the vertical arrows belong to \mathcal{A} , then its top face is a pushout if and only if the front and right faces are pullbacks.*

Pushout squares that enjoy only the “if” half of item (2) above are called \mathcal{A} -stable.

A $\text{Mor}(\mathbf{X})$ -Van Kampen square is called Van Kampen and a $\text{Mor}(\mathbf{X})$ -stable square stable.

$$\begin{array}{ccccc}
 & & A' & \xrightarrow{f'} & B' \\
 & m' \swarrow & | & \searrow n' & \\
 C' & \xrightarrow{a} & D' & & \\
 & \swarrow c & \downarrow a & \searrow d & \downarrow b \\
 & & A & \xrightarrow{f} & B \\
 & \swarrow c & \downarrow d & \searrow n & \\
 C & \xrightarrow{g} & D & &
 \end{array}$$

We can now define \mathcal{M} -adhesive categories.

► **Definition 2.2** (\mathcal{M} -adhesive category). *Let \mathbf{X} be a category and \mathcal{M} a subclass of $\text{Mono}(\mathbf{X})$ including all isomorphisms, closed under composition, and stable under pullbacks and pushouts. The category \mathbf{X} is said to be \mathcal{M} -adhesive if*

1. *it has \mathcal{M} -pullbacks, i.e. pullbacks along arrows of \mathcal{M} ;*
2. *it has \mathcal{M} -pushouts, i.e. pushouts along arrows of \mathcal{M} ;*
3. *\mathcal{M} -pushouts are \mathcal{M} -Van Kampen squares.*

A category \mathbf{X} is said to be strictly \mathcal{M} -adhesive if \mathcal{M} -pushouts are Van Kampen squares.

We write $m: X \rightarrow Y$ to denote that an arrow $m: X \rightarrow Y$ belongs to \mathcal{M} .

► **Remark 2.3.** *Adhesivity and quasiadhesivity [28,32] coincide with strict $\text{Mono}(\mathbf{X})$ -adhesivity and strict $\text{Reg}(\mathbf{X})$ -adhesivity, respectively.*

► **Example 2.4.** *Every category \mathbf{X} is $\text{I}(\mathbf{X})$ -adhesive, where $\text{I}(\mathbf{X})$ is the class of isomorphisms.*

\mathcal{M} -adhesivity is well-behaved with respect to the construction of slice and functor categories [36], as shown by the following theorems [19,32].

► **Theorem 2.5.** *Let \mathbf{X} be an \mathcal{M} -adhesive category. Given an object X the category \mathbf{X}/X is \mathcal{M}/X -adhesive with $\mathcal{M}/X := \{m \in \text{Mor}(\mathbf{X}/X) \mid m \in \mathcal{M}\}$. Similarly, X/\mathbf{X} is X/\mathcal{M} -adhesive with $X/\mathcal{M} := \{m \in \text{Mor}(X/\mathbf{X}) \mid m \in \mathcal{M}\}$.*

Moreover for every small category \mathbf{Y} , the category $\mathbf{X}^{\mathbf{Y}}$ of functors $\mathbf{Y} \rightarrow \mathbf{X}$ is $\mathcal{M}^{\mathbf{Y}}$ -adhesive, where $\mathcal{M}^{\mathbf{Y}} := \{\eta \in \text{Mor}(\mathbf{X}^{\mathbf{Y}}) \mid \eta_Y \in \mathcal{M} \text{ for every } Y \in \mathbf{Y}\}$.

We can list various examples of \mathcal{M} -adhesive categories (see [12,13,33]).

► **Example 2.6.** **Set** is adhesive, and, more generally, every topos is adhesive [33]. By the closure properties above, every presheaf $[\mathbf{X}, \mathbf{Set}]$ is adhesive, thus the category **Graph** = $[E \rightrightarrows V, \mathbf{Set}]$ is adhesive where $E \rightrightarrows V$ is the two objects category with two morphisms $s, t: E \rightarrow V$. Similarly, various categories of hypergraphs can be shown to be adhesive, such as term graphs and hierarchical graphs [14]. Note that the category **sGraphs** of simple graphs, i.e. graphs without parallel edges, is $\text{Reg}(\mathbf{sGraphs})$ -adhesive [8] but not quasiadhesive.

A number of properties of adhesive categories that play a role in the theory of rewriting generalise to \mathcal{M} -adhesive categories. These include \mathcal{M} -pushout-pullback decomposition and uniqueness of pushouts complements (details and proofs are in Appendix A).

2.2 DPO rewriting systems derivations

\mathcal{M} -adhesive categories are the right context in which to perform abstract rewriting using what is called the Double-Pushout (DPO) approach.

► **Definition 2.7** ([29, 30]). *Let \mathbf{X} be an \mathcal{M} -adhesive category. A left \mathcal{M} -linear rule ρ is a pair of arrows $\rho = (l : K \rightarrow L, r : K \rightarrow R)$ such that l belongs to \mathcal{M} . The rule ρ is \mathcal{M} -linear if $r \in \mathcal{M}$ too. The object L is the left-hand side, R the right-hand side, and K the interface.*

A left-linear rewriting system is a pair (\mathbf{X}, \mathbf{R}) where \mathbf{X} is an \mathcal{M} -adhesive category and \mathbf{R} a set of left \mathcal{M} -linear rules. It is called \mathcal{M} -linear if every rule in \mathbf{R} is \mathcal{M} -linear.

Given two objects G and H and a rule $\rho = (l, r)$ in \mathbf{R} , a direct derivation \mathcal{D} from G to H applying rule ρ is a diagram as the one below, in which both squares are pushouts. The arrow m is called the match of the derivation, while h is its co-match. We denote a direct derivation \mathcal{D} from G to H as $\mathcal{D} : G \Rightarrow H$.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ m \downarrow & & \downarrow k & & \downarrow h \\ G & \xleftarrow{f} & D & \xrightarrow{g} & H \end{array}$$

A derivation can now be defined simply as a sequence of direct derivations.

► **Definition 2.8** (Derivation). *Let \mathbf{X} be an \mathcal{M} -adhesive category and (\mathbf{X}, \mathbf{R}) a left-linear rewriting system. Given $G, H \in \mathbf{X}$, a derivation $\underline{\mathcal{D}}$ with source G and target H , written $\underline{\mathcal{D}} : G \Rightarrow H$, is defined as a sequence $\{\mathcal{D}_i\}_{i=0}^n$ of direct derivations such that $\mathcal{D}_i : G_i \Rightarrow G_{i+1}$ is a direct derivation for every index i and $G_0 = G$, $G_{n+1} = H$. Moreover, we have an empty derivation $G : G \Rightarrow G$ for each $G \in \mathbf{X}$. We denote by $\text{lg}(\underline{\mathcal{D}})$ the length of a derivation $\underline{\mathcal{D}}$.*

Given a derivation $\underline{\mathcal{D}} = \{\mathcal{D}_i\}_{i=0}^n$, we define $r(\underline{\mathcal{D}}) = \{\rho_i\}_{i=0}^n$ as the sequence of rules such that every \mathcal{D}_i applies rule $\rho_i \in \mathbf{R}$.

Derivations naturally compose: given $\underline{\mathcal{D}} : G \Rightarrow H$ and $\underline{\mathcal{E}} : H \Rightarrow F$, we can consider their composition $\underline{\mathcal{D}} \cdot \underline{\mathcal{E}} : G \Rightarrow F$, defined in the obvious way.

Often, we are interested in derivations only up to some notion of coherent isomorphism between them. This leads us to the following definition.

► **Definition 2.9** (Abstraction equivalence). *Let \mathbf{X} be an \mathcal{M} -adhesive category and (\mathbf{X}, \mathbf{R}) a left-linear rewriting system. An abstraction equivalence between derivations $\underline{\mathcal{D}}$ and $\underline{\mathcal{D}}'$ with the same length and $r(\underline{\mathcal{D}}) = r(\underline{\mathcal{D}}')$ is a family of isomorphisms $\{\phi_X\}_{X \in \{G_i, D_i\}}$ such that the diagram below commutes for $i \in [0, \min(0, \text{lg}(\underline{\mathcal{D}}) - 1)]$. We say that $\underline{\mathcal{D}}$ and $\underline{\mathcal{D}}'$ are abstraction equivalent, written $\underline{\mathcal{D}} \equiv_a \underline{\mathcal{D}}'$, if there exists an abstraction equivalence between them.*

$$\begin{array}{ccccc} G'_i & \xleftarrow{f'_i} & D'_i & \xrightarrow{g'_i} & G'_{i+1} \\ m'_i \uparrow & & k'_i \uparrow & & h'_i \uparrow \\ L_i & \xleftarrow{l_i} & K_i & \xrightarrow{r_i} & R_i \\ m_i \downarrow & & \downarrow k_i & & \downarrow h_i \\ G_i & \xleftarrow{f_i} & D_i & \xrightarrow{g_i} & G_{i+1} \end{array} \quad \begin{array}{c} \phi_{G_{i+1}} \\ \phi_{D_i} \end{array}$$

\mathcal{M} -adhesivity ensures the uniqueness of the result of applying a rule to an object: two direct derivations using the same match are abstraction equivalent (see Proposition C.1 of the extended version).

3 Independence in rewriting

In this section we discuss the notion of independence between two consecutive rewriting steps, a fundamental ingredient of the rewriting theory, which comes into play when viewing a sequence of steps as a concurrent computation. We observe that moving from linear to left-linear rules leads to the failure of various basic properties of the classical notion of independence used in the linear setting and we single out a framework in which these can be re-established, possibly in a weakened form.

3.1 Sequentially independent and switchable derivations

Sequential independence is the canonical notion of independence in the DPO approach.

► **Definition 3.1** (Sequential independence). *Let \mathbf{X} be an \mathcal{M} -adhesive category and $(\mathbf{X}, \mathcal{R})$ a left-linear rewriting system. Let also $\underline{\mathcal{D}} = \{\mathcal{D}_i\}_{i=0}^1$ be a derivation of length 2, with \mathcal{D}_i using rule $\rho_i = (l_i, r_i)$. We say that \mathcal{D}_0 and \mathcal{D}_1 are sequentially independent if there is a pair of arrows $i_0: R_0 \rightarrow D_1$ and $i_1: L_1 \rightarrow D_0$ such that the following diagram commutes. In this case (i_0, i_1) is called an independence pair.*

$$\begin{array}{ccccccc}
 L_0 & \xleftarrow{l_0} & K_0 & \xrightarrow{r_0} & R_0 & \xrightarrow{\quad} & L_1 & \xleftarrow{l_1} & K_1 & \xrightarrow{r_1} & R_1 \\
 m_0 \downarrow & & k_0 \downarrow & & & & & & & & \downarrow k_1 & & \downarrow h_1 \\
 G_0 & \xleftarrow{f_0} & D_0 & \xrightarrow{g_0} & G_1 & \xleftarrow{f_1} & D_1 & \xrightarrow{g_1} & G_2
 \end{array}$$

Intuitively, the existence of the independence pair captures the possibility of switching the application of the two rules: $f_0 \circ i_1$ is a match for ρ_1 in G_0 and the existence of $i_0: R_0 \rightarrow D_1$ means that ρ_1 does not delete items in the image of K_0 , thus ρ_0 can be applied after ρ_1 .

► **Remark 3.2.** It is worth mentioning that if $(\mathbf{X}, \mathcal{R})$ is a linear rewriting system then two derivations \mathcal{D}_0 and \mathcal{D}_1 can have at most one independence pair. Indeed if (i_0, i_1) and (i'_0, i'_1) are independence pairs, then

$$g_0 \circ i_1 = g_0 \circ i'_1 \quad f_1 \circ i_0 = f_1 \circ i'_0$$

But in linear systems g_0 and f_1 are both monos, entailing $i_0 = i'_0$ and $i_1 = i'_1$.

We next formalise what it means for a derivation being the switch of another.

► **Definition 3.3** (Switch). *Let \mathbf{X} be an \mathcal{M} -adhesive category and $(\mathbf{X}, \mathcal{R})$ a left-linear rewriting system. Let also $\underline{\mathcal{D}} = \{\mathcal{D}_i\}_{i=0}^1$ be a derivation made of two sequentially independent derivations $\mathcal{D}_0, \mathcal{D}_1$ with independence pair (i_0, i_1) , as in the diagram on the left below. A switch of $\underline{\mathcal{D}}$ along (i_0, i_1) is a derivation $\underline{\mathcal{E}} = \{\mathcal{E}_i\}_{i=0}^1$, between the same objects and using the same rules in reverse order, as in the diagram on the right below*

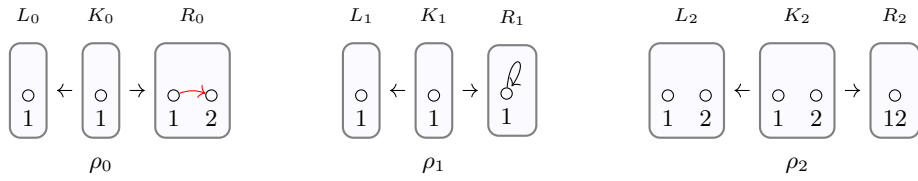
$$\begin{array}{ccccccc}
 L_0 & \xleftarrow{l_0} & K_0 & \xrightarrow{r_0} & R_0 & \xrightarrow{\quad} & L_1 & \xleftarrow{l_1} & K_1 & \xrightarrow{r_1} & R_1 \\
 m_0 \downarrow & & k_0 \downarrow & & & & & & & & \downarrow k_1 & & \downarrow h_1 \\
 G_0 & \xleftarrow{f_0} & D_0 & \xrightarrow{g_0} & G_1 & \xleftarrow{f_1} & D_1 & \xrightarrow{g_1} & G_2
 \end{array}
 \quad
 \begin{array}{ccccccc}
 L_1 & \xleftarrow{l_1} & K_1 & \xrightarrow{r_1} & R_1 & \xrightarrow{\quad} & L_0 & \xleftarrow{l_0} & K_0 & \xrightarrow{r_0} & R_0 \\
 m'_0 \downarrow & & k'_0 \downarrow & & & & & & & & \downarrow k'_1 & & \downarrow h'_1 \\
 G_0 & \xleftarrow{f'_0} & D'_0 & \xrightarrow{g'_0} & G_1 & \xleftarrow{f'_1} & D'_1 & \xrightarrow{g'_1} & G_2
 \end{array}$$

such that there is an independence pair (i'_0, i'_1) between \mathcal{E}_0 and \mathcal{E}_1 and

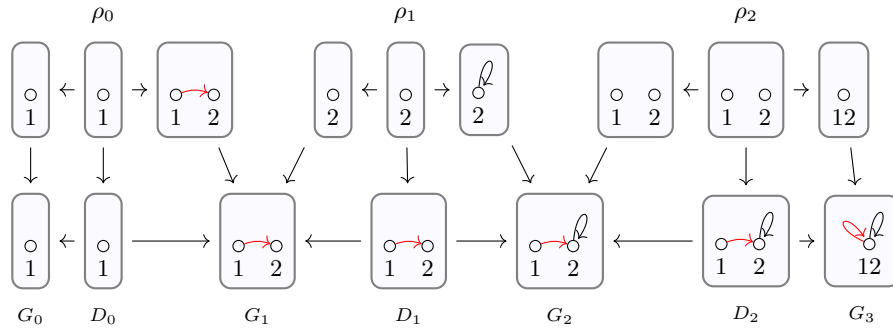
$$m_0 = f'_0 \circ i'_1 \quad h_1 = g'_1 \circ i'_0 \quad m'_0 = f_0 \circ i_1 \quad h'_1 = g_1 \circ i_0.$$

If a switch of \mathcal{D}_0 and \mathcal{D}_1 exists we say that they are switchable.

11:8 Left-Linear Rewriting in Adhesive Categories



■ Figure 3 A rewriting system in **Graph**.

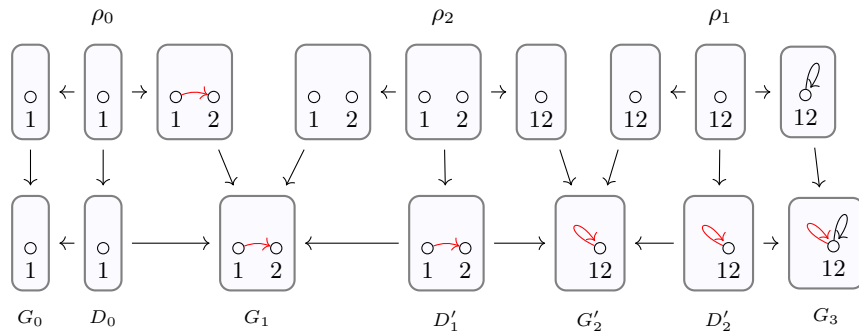


■ Figure 4 The derivation \mathcal{D} .

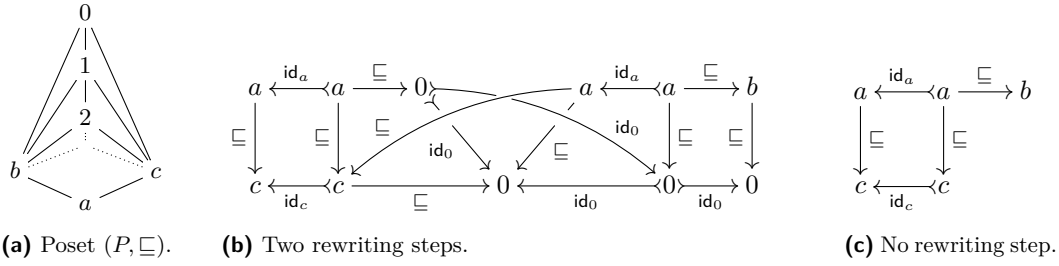
It can be shown that switches along the same independence pair are unique up to abstraction equivalence (see Proposition C.3 of the extended version).

► **Example 3.4.** Consider a rewriting system in **Graph**, the category of directed graphs and graph morphisms, which is adhesive. The set of rules $R = \{\rho_0, \rho_1, \rho_2\}$ is in Fig. 3, where numbers are used to represent the morphisms from the interface to the left- and right-hand sides. Rules ρ_0 and ρ_1 are linear: ρ_0 generates a new node and a new edge, while ρ_1 creates a self-loop edge. Instead, ρ_2 is left-linear but not linear, as it “merges” two nodes.

A derivation \mathcal{D} consisting of three steps $\mathcal{D}_0, \mathcal{D}_1,$ and \mathcal{D}_2 , each applying the corresponding rule ρ_i , is depicted in Fig. 4. Note that the numbers in ρ_1 were changed consistently to represent the vertical morphisms. Steps \mathcal{D}_1 and \mathcal{D}_2 are clearly sequential independent via the independence pair $(R_1 \rightarrow D_2, L_2 \rightarrow D_1)$, mapping nodes according to their numbering. A switch of \mathcal{D} along such independence pair is the derivation \mathcal{E} in Fig. 5. Instead, in \mathcal{D} the first two steps applying ρ_0 and ρ_1 are not sequential independent, intuitively because ρ_1 uses the node produced by ρ_0 for attaching a self-loop.



■ Figure 5 The derivation \mathcal{E} .



■ **Figure 6** Sequential independence does not imply switchability.

As we mentioned, for linear rewriting systems it is canonical to identify derivations that are equal “up to switching”, i.e. that differ only in the order of independent steps. The same notion can be given for left-linear rewriting systems by relying on the notion of switch.

We recall some notations on permutations. A *permutation* on $[0, n]$ is a bijection $\sigma : [0, n] \rightarrow [0, n]$. It is a *transposition* ν if there are $i, j \in [1, n]$, $i \neq j$ such that $\sigma(i) = j$, $\sigma(j) = i$, and $\sigma(k) = k$ otherwise; it is denoted as (i, j) . Given a permutation σ , an *inversion* for σ is a pair (i, j) such that $i < j$ and $\sigma(j) < \sigma(i)$; $\text{inv}(\sigma)$ denotes the set of inversions of σ .

Switch equivalence is now defined as the equivalence relating derivations that can be obtained one from another by a sequence of switches. Moreover, intermediate graphs can be taken up to isomorphism according to abstraction equivalence.

► **Definition 3.5** (Switch equivalence). *Let (\mathbf{X}, R) be a left-linear rewriting system. Let also $\mathcal{D}, \mathcal{E} : G \Rightarrow H$ be derivations with the same length, $\mathcal{D} = \{\mathcal{D}_i\}_{i=0}^n$ and $\mathcal{E} = \{\mathcal{E}_i\}_{i=0}^n$. If $\mathcal{D}_i \cdot \mathcal{D}_{i+1}$ is a switch of $\mathcal{E}_i \cdot \mathcal{E}_{i+1}$ for some $i \in [0, n-1]$ and $\mathcal{D}_j = \mathcal{E}_j$ for each $j \notin \{i, i+1\}$ then we write $\mathcal{D} \rightsquigarrow_{(i, i+1)} \mathcal{E}$. A switching sequence is a sequence $\{\mathcal{D}_k\}_{k=0}^m$ of derivations such that $\mathcal{D}_0 \rightsquigarrow_{\nu_1} \mathcal{D}_1 \rightsquigarrow_{\nu_2} \dots \rightsquigarrow_{\nu_m} \mathcal{D}_m$ with $\nu_k = (i_k, i_k + 1)$.*

Let us denote by $\nu_{k,h}$ the composition $\nu_h \circ \nu_{h-1} \circ \dots \circ \nu_k$. We say that the switching sequence consists of inversions if for all $k \in [0, m]$ the transposition ν_k is an inversion for $\nu_{1,m}$.

Two derivations $\mathcal{D}, \mathcal{E} : G \Rightarrow H$ are switch equivalent, written $\mathcal{D} \equiv^{sh} \mathcal{E}$, if there is a switching sequence $\{\mathcal{D}_k\}_{k=0}^m$ such that $\mathcal{D} \equiv_a \mathcal{D}_0 \rightsquigarrow_{\nu_1} \mathcal{D}_1 \rightsquigarrow_{\nu_2} \dots \rightsquigarrow_{\nu_m} \mathcal{D}_m \equiv_a \mathcal{E}$. To point out a chosen permutation of rewriting steps, we will also write $\mathcal{D} \equiv_{\sigma}^{sh} \mathcal{E}$, where σ is the composition of the transposition appearing in a chosen switching sequence.

► **Remark 3.6.** The possibility of an empty switching sequence assures that two abstraction equivalent derivations are switch equivalent.

3.2 Church-Rosser: Sequential independence and switchability

The fact that sequential independence implies switchability always holds for linear rules (see Proposition C.8 of the extended version). The result is so indispensable that typically, in the literature, it is not even stated, in the sense that switchability is not introduced axiomatically as in Definition 3.3, but it is based on the explicit construction of a switch.

For left-linear rewriting systems sequential independence does not imply switchability (while the converse implication clearly holds), as shown by the next example.

► **Example 3.7.** Consider the poset (P, \sqsubseteq) in Fig. 6a where $P = \mathbb{N} \cup \{a, b, c\}$ and \sqsubseteq is defined by $m \sqsubseteq n$ if $m \geq n$, $a \sqsubseteq x$ for all $x \in P$ and $b, c \sqsubseteq n$ for all $n \in \mathbb{N}$. Let \mathbf{X} be the category associated with this order, which by Example 2.4 is $\mathbf{l}(\mathbf{X})$ -adhesive. Consider a rewriting system whose set of rules contains the following

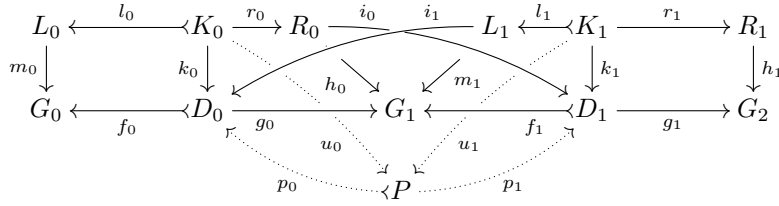
$$a \xleftarrow{\text{id}_a} \langle a \xrightarrow{\sqsubseteq} 0 \quad a \xleftarrow{\text{id}_a} \langle a \xrightarrow{\sqsubseteq} b$$

11:10 Left-Linear Rewriting in Adhesive Categories

We can then consider the derivation $\mathcal{D} = \{\mathcal{D}_i\}_{i=0}^1$ in Fig. 6b. Note that the two direct derivations are sequential independent. However there is no switch since the rule applied by \mathcal{D}_1 cannot be applied to c . In fact, there is a unique morphism $a \rightarrow c$, yielding the diagram in Fig. 6c. But b and c do not have a supremum in the poset underlying \mathbf{X} , thus the arrows $a \rightarrow b$ and $a \rightarrow c$ do not have a pushout. Hence we do not get a direct derivation from c .

The conditions guaranteeing switchability are inspired by the notion of *canonical filler* [30].

► **Definition 3.8** (strong independence pair). *Let \mathbf{X} be an \mathcal{M} -adhesive category and $(\mathbf{X}, \mathcal{R})$ a left-linear rewriting system. Let also (i_0, i_1) be an independence pair between two direct derivations $\mathcal{D}_0, \mathcal{D}_1$ as in the solid part of the diagram below*



Consider the pullback of g_0 and f_1 , which yields $p_i : P \rightarrow D_i$ for $i \in \{0, 1\}$ and the mediating arrows $u_i : K_i \rightarrow P$ for $i \in \{0, 1\}$ into the pullback object (see Proposition C.4 of the extended version for details). We say that (i_0, i_1) is a strong independence pair if the first two squares depicted below are pushouts and if the pushout of $r_1 : K_1 \rightarrow R_1$ and $u_1 : K_1 \rightarrow P$ exists

$$\begin{array}{ccc}
 K_0 \xrightarrow{r_0} R_0 & K_1 \xrightarrow{l_1} L_1 & K_1 \xrightarrow{r_1} R_1 \\
 u_0 \downarrow & \downarrow i_0 & u_1 \downarrow & \downarrow j_0 \\
 P \xrightarrow{p_1} D_1 & P \xrightarrow{p_0} D_0 & P \xrightarrow{q_1} Q_1
 \end{array}$$

We can now prove a Local Church-Rosser Theorem for strong independence pairs.

► **Proposition 3.9** (Local Church-Rosser Theorem). *Let (i_0, i_1) be a strong independence pair between \mathcal{D}_0 and \mathcal{D}_1 . Then \mathcal{D}_0 and \mathcal{D}_1 are switchable.*

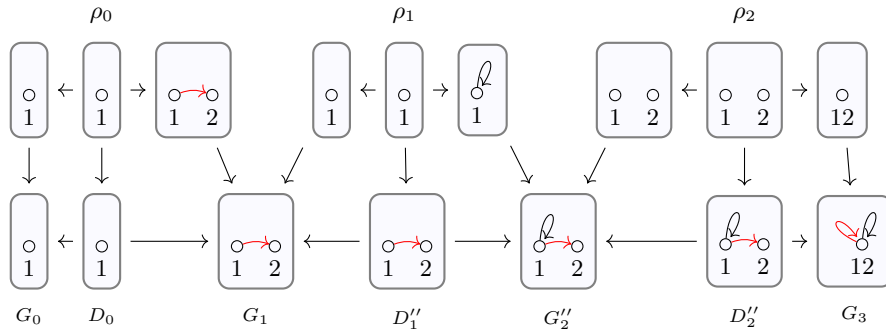
The correspondence between sequential independence and switchability is fundamental. We name the class of rewriting systems where this property holds.

► **Definition 3.10** (Strong enforcing rewriting systems). *A left-linear rewriting system is strong enforcing if every independence pair between two direct derivations is strong.*

We can identify a large class of adhesive categories such that all left-linear rewriting systems over such categories are strong enforcing. This class includes **Set** and it is closed under comma and functor category constructions. As such, it includes essentially all categories (e.g., presheaves over set) that are typically considered for modelling purposes. Notably, it contains the category **Graph** of directed graphs. This is a natural generalisation from adhesive to \mathcal{M} -adhesive categories of a class studied in [6] (see Appendix B for details).

Still, there are \mathcal{M} -adhesive rewriting systems that are not strong enforcing.

► **Example 3.11** (Non-strong enforcing left-linear rewriting system). In light of Proposition 3.9, Example 3.7 provides an example of an independence pair that is not strong. This gives an example of a left-linear rewriting system in a \mathcal{M} -adhesive category that is quite pathological since $\mathcal{M} = \mathbf{l}(\mathbf{X})$. However, this is expected, since all natural examples seem to belong to the well-behaved class mentioned above.



■ **Figure 7** The derivation $\underline{\mathcal{D}}'$.

► **Remark 3.12.** By Proposition 3.9 the existence of a strong independence pair entails switchability, which in turn entails sequential independence by construction. Strong enforcing rewriting systems are exactly those rewriting systems in which these three notions coincide.

Consistency with the theory of linear rewriting systems is ensured by the fact that all linear rewriting systems are strong enforcing (see Proposition C.8 of the extended version).

3.3 Well-switching rewriting systems

Even if we work in strong enforcing rewriting systems where sequential independence ensures switchability, when dealing with left-linear rules there is a further, possibly more serious issue, namely that there can be more than one independence pair between the same derivations (cfr. Remark 3.2). This hinders the very idea of using sequential independence as a basis of a theory of concurrency for rewriting systems, since exchanges performed using different independence pairs may lead to derivations that are not abstraction equivalent, thus equating computations that should definitively be taken apart, as shown in the example below.

► **Example 3.13.** Consider the derivation $\underline{\mathcal{E}}$ from Example 3.4 (see Fig. 5). The last two steps are sequential independent, but one easily sees that there are two distinct independence pairs, as the left-hand side of ρ_1 can be mapped either to node 1 or to node 2 in D'_1 . Correspondingly, there are two switches of $\underline{\mathcal{E}}$: one is the derivation $\underline{\mathcal{D}}$ in Fig. 4 we started from, the other is the derivation $\underline{\mathcal{D}}'$ in Fig. 7.

As a consequence, $\underline{\mathcal{D}}$ and $\underline{\mathcal{D}}'$ would be switch equivalent, but this is not acceptable when viewing equivalence classes of derivations as concurrent computations: in $\underline{\mathcal{D}}$ the first two steps are not sequential independent, while in $\underline{\mathcal{D}}'$ they are, intuitively because in $\underline{\mathcal{D}}$ rule ρ_1 uses the node generated by ρ_0 (adding a self-loop to it), while in $\underline{\mathcal{D}}'$ rule ρ_1 uses the node that was in the initial graph. Also observe that the graphs G_2 and G'_2 produced after two steps in $\underline{\mathcal{D}}$ and $\underline{\mathcal{D}}'$ are not isomorphic. From the technical point of view, the property of being switch equivalent is not closed by prefix, and this prevents deriving a sensible concurrent semantics: In fact $\underline{\mathcal{D}} = \mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2$ and $\underline{\mathcal{D}}' = \mathcal{D}'_0 \cdot \mathcal{D}'_1 \cdot \mathcal{D}'_2$ are switch equivalent, while if we consider the first two steps, derivations $\mathcal{D}_0 \cdot \mathcal{D}_1$ and $\mathcal{D}'_0 \cdot \mathcal{D}'_1$ are not switch equivalent.

For these reasons we believe a theory of rewriting for left-linear rules in adhesive categories should be developed for systems where the uniqueness of the independence pair is ensured.

► **Definition 3.14 (Well-switching rewriting systems).** A left-linear rewriting system (\mathbf{X}, \mathbf{R}) is well-switching if it is strong enforcing and, for every derivation $\underline{\mathcal{D}} := \{\mathcal{D}_i\}_{i=0}^1$, there is at most one independence pair between \mathcal{D}_0 and \mathcal{D}_1 .

11:12 Left-Linear Rewriting in Adhesive Categories

Clearly, linear rewriting systems are well-switching (see Proposition C.10 of the extended version). Moreover, we next observe that various classes of rewriting systems, comprising all the ones used in modelling the applications to the encoding of process calculi or of bio and chemical systems mentioned in the introduction, are actually well-switching.

A first class consists of those rewriting systems over possibly hierarchical graphical structures obtained as algebras of suitable signatures where rules are constrained not to merge elements of top level sorts in the hierarchy (for graphs, nodes can be merged while edges cannot). The idea here is to consider graph structures as presheaves on categories in which there are objects that play the role of *roots*, i.e. objects that are not the codomain of any arrow besides the identity.

► **Definition 3.15** (Root-preserving graphical rewriting systems). *Let \mathbf{X} be a category, an object $X \in \mathbf{X}$ is a root if the only arrow with codomain X is id_X . The category $\mathbf{X}\text{-Graph}$ of \mathbf{X} -graphs is the category $\mathbf{Set}^{\mathbf{X}}$. A root-preserving graphical rewriting system is a left-linear rewriting system $(\mathbf{X}\text{-Graph}, \mathbf{R})$ such that for each rule $(l: K \rightarrow L, r: K \rightarrow R)$ in \mathbf{R} it holds*

1. *for every $X \in \mathbf{X}$ and $x \in L(X)$, there exists a root Y and an arrow $f: Y \rightarrow X$ such that x belongs to the image of $L(f): L(Y) \rightarrow L(X)$;*
2. *$r: K \rightarrow R$ is mono on the roots, i.e. for every root $X \in \mathbf{X}$ the component $r_X: K(X) \rightarrow R(X)$ is injective.*

For instance, the category **Graph** can be obtained by taking as \mathbf{X} the category generated by $E \rightrightarrows V$. In this case E is the only root, hence, condition 1 asks that in the left-hand side of each rule there are no isolated nodes, while condition 2 asks that the morphism $r: K \rightarrow R$ is injective on edges, i.e. it can only merge nodes.

► **Lemma 3.16.** *All root-preserving graphical rewriting systems are well-switching.*

Another interesting class of well-switching rewriting systems is given by e-graphs.

► **Example 3.17** (E-graphs). Consider the category **EGraphs**, where objects are (directed) graphs endowed with an equivalence over nodes, and arrows are graph morphisms that preserve the equivalence, as considered in [5], closely related to e-graphs [42]. Formally, **EGraphs** can be seen as the subcategory of the presheaf $[E \rightrightarrows V \rightarrow Q, \mathbf{Set}]$ where objects are constrained to have the component $V \rightarrow Q$ surjective.

Explicitly, an e-graph G is a triple $\langle s_G, t_G, q_G \rangle$ where $s_G, t_G: E_G \rightrightarrows V_G$ provides the graphical structure, while the surjective function $q_G: V_G \rightarrow Q_G$ maps each node to the corresponding equivalence class. Notice that the inclusion functor into $[E \rightrightarrows V \rightarrow Q, \mathbf{Set}]$ creates pullbacks and pushouts [36], so that they are computed component-wise.

A morphism in **EGraph** is mono if the components over E and V are mono, i.e. if it is mono as a morphism in **Graph**. It is regular mono if also the component on Q is mono, i.e. if it reflects equivalence classes besides preserving them. This characterisation of regular monos and the fact that pullbacks and pushouts are computed component-wise allows us to prove quasi-adhesivity of **EGraphs** at once. Moreover, one can deduce that every rewriting system (\mathbf{X}, \mathbf{R}) that is left-linear with respect to $\text{Reg}(\mathbf{EGraphs})$ is strong enforcing: this is done exploiting again the inclusion functor into $[E \rightrightarrows V \rightarrow Q, \mathbf{Set}]$.

Left-linear rewriting systems with respect to $\text{Reg}(\mathbf{EGraphs})$ are well-switching. They have been used in [5] for the graphical implementation of nominal calculi, where, differently from [27], as a result of name passing the received name is not merged with a local one, but put in the same equivalence class, better tracing the causal dependencies among reductions.

3.4 A canonical form for switch equivalences

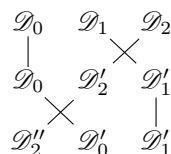
As discussed in the introduction, in the linear case a number of basic properties of switch equivalence, i.e. the globality of independence, the consistency of switching, and the existence of a canonical form for equivalence proofs, can be derived from a characterisation of switch equivalence in terms of consistent permutations or processes.

When dealing with left-linear rules, such a characterisation fails. Leaving aside the formal details (the interested reader can refer to Definition C.12 of the extended version), the intuition is quite simple. In the linear case one proves that two derivations using the same rules in different orders are switch equivalent when the colimits of the two derivations, seen as diagrams, are isomorphic and the isomorphism properly commutes with the embeddings of the rules. When considering left-linear derivations, this is no longer true. Intuitively this is due to the fact that a rewriting step can merge parts of an object thus making the colimit little informative. This effect can be seen for derivations \mathcal{D} in Fig. 4 and \mathcal{D}' in Fig. 7. They are not abstraction equivalent but the colimit of both derivations is a single node with two self-loops, in a way that all commutativity requirements are trivially satisfied.

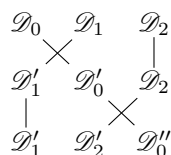
In this section we show that, despite the failure of such characterisation, actually much of the theory can be retained, taking a different and sensibly more complex route for the proofs: globality of independence holds in a weak form, conceptually linked to the fact that fusions introduce a form of disjunctive causality, while consistency of switching holds unchanged. Finally, a canonical form for switching sequences can also be recovered.

We start by proving two lemmas dealing with derivations of length 3. Despite being technical, these lemmas are fundamental building blocks for obtaining our results.

The first lemma is at the core of globality for independence. It shows that if we have a derivation consisting of three rewriting steps, say $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2$ where \mathcal{D}_0 and \mathcal{D}_1 are independent, then if we switch \mathcal{D}_2 to the first position, obtaining a derivation $\mathcal{D}_2'' \cdot \mathcal{D}_0' \cdot \mathcal{D}_1'$, as in the diagram below where vertical lines represent permutations, the steps \mathcal{D}_0' and \mathcal{D}_1' are still independent.

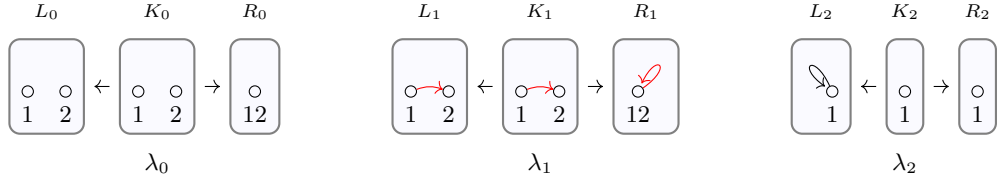


If instead \mathcal{D}_1 and \mathcal{D}_2 are independent in $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2$ and we switch \mathcal{D}_0 to the last position, obtaining a derivation $\mathcal{D}_1' \cdot \mathcal{D}_2' \cdot \mathcal{D}_0''$, as in the diagram below, the steps \mathcal{D}_1' and \mathcal{D}_2' could be no longer independent. Intuitively, this is due to the fact that dealing with left-linear rewriting systems introduces disjunctive forms of causality. Hence, if in $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2$ we have that \mathcal{D}_0 and \mathcal{D}_1 are disjunctive causes of \mathcal{D}_2 , at least one of the two is needed to enable the application of \mathcal{D}_2 and this explain why in $\mathcal{D}_1' \cdot \mathcal{D}_2' \cdot \mathcal{D}_0''$ we have that \mathcal{D}_1' and \mathcal{D}_2' are no longer independent. This is exemplified below.

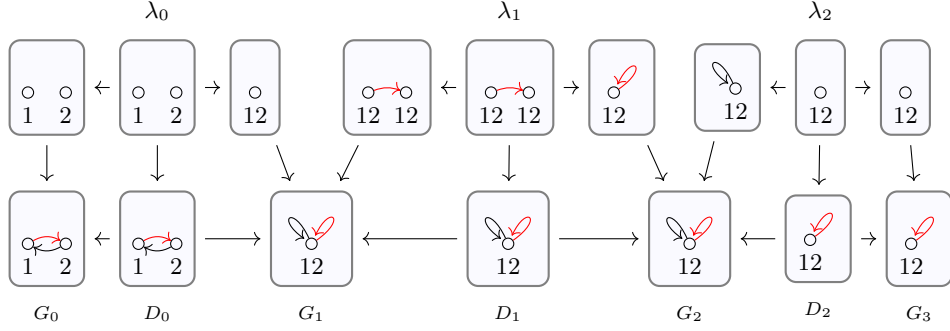


► **Example 3.18.** Consider the rewriting system in **Graph** consisting of the rules λ_0 , λ_1 , and λ_2 in Fig. 8. Consider the derivation \mathcal{F} in Fig. 9. Note that rule λ_2 needs the black self-loop on node 12 to be applied. This can be generated either by λ_0 or by λ_1 merging nodes 1 and 2,

11:14 Left-Linear Rewriting in Adhesive Categories



■ **Figure 8** A new rewriting system in **Graph**.



■ **Figure 9** The derivation \mathcal{F} .

hence at least one of them must precede the application of λ_2 . Indeed, in \mathcal{F} it is easily seen that the second and the third step applying λ_1 and λ_2 are independent. However, we could switch twice the application of λ_0 , bringing it in the last position, obtaining a derivation \mathcal{F}' as depicted in Fig. 10, where the applications of λ_1 and λ_2 are no longer independent.

► **Lemma 3.19** (Globality of independence). *Let \mathbf{X} be an \mathcal{M} -adhesive category and (\mathbf{X}, \mathbf{R}) a well-switching rewriting system. Let $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2$ be a three-steps derivation.*

1. *If \mathcal{D}_0 and \mathcal{D}_1 are switchable and there is a switching sequence $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2 \rightsquigarrow_{(1,2)} \mathcal{D}_0 \cdot \mathcal{D}'_2 \cdot \mathcal{D}'_1 \rightsquigarrow_{(0,1)} \mathcal{D}''_2 \cdot \mathcal{D}'_0 \cdot \mathcal{D}'_1$ then in the last derivation \mathcal{D}'_0 and \mathcal{D}'_1 are switchable;*
2. *Suppose that \mathcal{D}_1 and \mathcal{D}_2 are switchable, hence $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2 \rightsquigarrow_{(1,2)} \mathcal{D}_0 \cdot \mathcal{D}'_2 \cdot \mathcal{D}'_1$ and \mathcal{D}_0 and \mathcal{D}'_2 are switchable. If there is a switching sequence $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2 \rightsquigarrow_{(0,1)} \mathcal{D}'_1 \cdot \mathcal{D}'_0 \cdot \mathcal{D}_2 \rightsquigarrow_{(1,2)} \mathcal{D}'_1 \cdot \mathcal{D}'_2 \cdot \mathcal{D}''_0$ then in the last derivation \mathcal{D}'_1 and \mathcal{D}'_2 are switchable.*

The second lemma captures the essence of the consistency of switchings. If in a derivation of three rewriting steps, say $\mathcal{D}_0 \cdot \mathcal{D}_1 \cdot \mathcal{D}_2$, we invert them leading to a sequence reordered as $\mathcal{D}_2 \cdot \mathcal{D}_1 \cdot \mathcal{D}_0$, we obtain the same derivation, independently from the order of switchings.

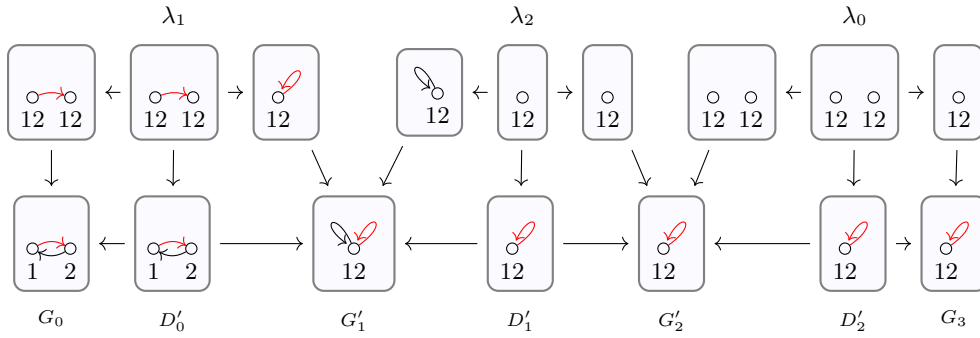
► **Lemma 3.20** (Consistency of switchings). *Let \mathbf{X} be an \mathcal{M} -adhesive category and (\mathbf{X}, \mathbf{R}) a well-switching rewriting system. Consider a derivation $\underline{\mathcal{D}} = \{\mathcal{D}_i\}_{i=0}^2$ and suppose that we have the following two switching sequences*

$$\underline{\mathcal{D}} \rightsquigarrow_{(0,1)} \underline{\mathcal{D}}' \rightsquigarrow_{(1,2)} \underline{\mathcal{D}}'' \rightsquigarrow_{(0,1)} \underline{\mathcal{D}}''' \qquad \underline{\mathcal{D}} \rightsquigarrow_{(1,2)} \underline{\mathcal{E}}' \rightsquigarrow_{(0,1)} \underline{\mathcal{E}}'' \rightsquigarrow_{(1,2)} \underline{\mathcal{E}}'''$$

Then $\underline{\mathcal{D}}'''$ and $\underline{\mathcal{E}}'''$ are abstraction equivalent.

The lemmas above open the way to the key result of this section, which establishes a canonical form for switching derivations.

The first result shows that when equating two derivation sequences by performing a series of switchings, we can limit ourselves to switchings that invert the order of steps that in the target derivation has to be reversed, i.e. we can limit ourselves to applying inversions.



■ **Figure 10** The derivation $\underline{\mathcal{F}}'$.

► **Theorem 3.21** (No need of useless switches). *Let $\underline{\mathcal{D}}$ and $\underline{\mathcal{D}}'$ be derivation sequences. If $\underline{\mathcal{D}} \equiv^{sh} \underline{\mathcal{D}}'$ then there is a switching sequence $\underline{\mathcal{D}} \equiv_a \underline{\mathcal{D}}_0 \rightsquigarrow_{\nu_1} \underline{\mathcal{D}}_1 \rightsquigarrow_{\nu_2} \dots \rightsquigarrow_{\nu_n} \underline{\mathcal{D}}_n \equiv_a \underline{\mathcal{D}}'$ consisting only of inversions.*

While inversions can be performed in any order in the case of linear rewriting systems, this is no longer true for left-linear rewriting systems due to the fact that, as already observed, globality of independence holds in a weak form. For instance, if we get back to Example 3.18 and consider derivation $\underline{\mathcal{F}}$ in Fig. 9, the last two steps applying rules λ_1 and λ_2 can be switched, thus leading to a derivation, call it $\underline{\mathcal{F}}''$, applying $\lambda_0, \lambda_2, \lambda_1$. This $\underline{\mathcal{F}}''$ is switch equivalent to $\underline{\mathcal{F}}'$ in Fig. 10, hence starting from $\underline{\mathcal{F}}''$ we can obtain $\underline{\mathcal{F}}'$ via a switching sequence, but we cannot start by switching λ_0 and λ_2 , even if this is an inversion.

Still, we can identify a canonical form for switching derivations: at each step one can apply a switch to the inversion with largest index.

► **Corollary 3.22** (Canonical form). *Let $\underline{\mathcal{D}}$ and $\underline{\mathcal{D}}'$ be derivation sequences. If $\underline{\mathcal{D}} \equiv_{\sigma}^{sh} \underline{\mathcal{D}}'$ then there is a switching sequence*

$$\underline{\mathcal{D}} \equiv_a \underline{\mathcal{D}}_0 \rightsquigarrow_{\nu_1} \underline{\mathcal{D}}_1 \rightsquigarrow_{\nu_2} \dots \rightsquigarrow_{\nu_n} \underline{\mathcal{D}}_n \equiv_a \underline{\mathcal{D}}'$$

where for $i \in [1, n]$ we have $\nu_i = (k, k + 1)$ with $k = \max\{j \mid (j, j + 1) \in \text{inv}(\nu_{i,n})\}$.

4 Conclusions and further work

We performed an in-depth investigation of the notion of independence between rewriting steps in the setting of left-linear rewriting systems over \mathcal{M} -adhesive categories, which encompasses most of the structures commonly used in analysing and modelling applications.

We showed that the canonical notion of independence adopted for linear systems does not enjoy some properties that are essential for developing a sensible theory of concurrency, notably a Church-Rosser theorem, and we identified a subclass of left-linear rewriting systems, which we call well-switching, as an appropriate setting where many key results can be re-established. Specifically, a Church-Rosser theorem, i.e. the possibility of performing independent steps in any order, is fully recovered. Moreover, the switch construction and correspondingly the switch equivalence, at the core of a theory of concurrency for rewriting systems, enjoy a number of fundamental properties: a weak form of globality of independence, consistency of switching and the existence of a canonical form for switch equivalence proofs.

The class of well-switching \mathcal{M} -adhesive rewriting systems is large. Generalising a result in [6], we showed that one of its defining properties (the fact that sequential independence implies switchability) holds for a general class of \mathcal{M} -adhesive categories that include all

presheaves over **Set**. Moreover, the second property, uniqueness of switching, is ensured when rewriting possibly hierarchical graphical structures, as long as elements belonging to the sorts of the roots are not merged (e.g., one can consider rewriting systems over directed graphs where only nodes are merged, as it happens in most of the approaches to the modelling of calculi and biological systems that we cited in the introduction, as well as in string diagrams).

In the DPO approach to rewriting, the notion of sequential independence, identifying consecutive rewriting steps that can be performed in reverse order is typically closely linked to that of *parallel independence* [21]. Parallel independence relates two rewriting steps starting from the same object, which can be applied in any order producing, up to isomorphism, the same resulting object. Parallel independent steps, when applied in sequence, lead to sequential independent steps and, conversely, from two sequential independent steps, as a byproduct of the switch construction, one can get two parallel independent steps.

Under mild additional conditions, one can have a notion of *parallel rule* in a way that two parallel independent steps can be also combined in a single rewriting step. Even if this is not discussed explicitly in the paper, our theory can be easily accommodated to encompass a notion of parallel independence and parallel rule application, enjoying the properties outlined above. To ensure this, however, one must require that the underlying category \mathbf{X} has binary coproducts and that the class \mathcal{M} is closed under them.

The results in this paper open the way to the development of a concurrent semantics for left-linear rewriting systems over \mathcal{M} -adhesive categories. In [3] the authors argue that for computational systems that allow one to express merging or fusions, the right semantical models are weak prime domains, a generalisation of prime domains, a staple in concurrency theory, and connected event structures, their event-based counterpart. The mentioned paper discusses only the case of graph rewriting and does not focus into the problems induced by the occurring of “merges” to the theory of rewriting (the systems considered there are implicitly assumed to be well-switching). We plan to consolidate the statement that weak prime domains are “the” model for systems with fusions by showing that they allow to provide a semantics for left-linear rewriting systems in \mathcal{M} -adhesive categories.

On the practical side, the present paper just surveyed the possibility of modelling e-graphs. This appears to be quite interesting as concurrent rewriting on such structures is an active area of research [34]. We plan to make the correspondence precise and investigate how the analysis techniques enabled by a concurrent semantics of rewriting can impact on e-graphs.

References

- 1 G. G. Azzi, A. Corradini, and L. Ribeiro. On the essence and initiality of conflicts in \mathcal{M} -adhesive transformation systems. *Journal of Logical and Algebraic Methods in Programming*, 109:100482, 2019.
- 2 P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation systems. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 3: Concurrency*, pages 107–187. World Scientific, 1999.
- 3 Paolo Baldan, Andrea Corradini, and Fabio Gadducci. Domains and event structures for fusions. In *LICS 2017*, pages 1–12. IEEE Computer Society, 2017.
- 4 Paolo Baldan, Andrea Corradini, Tobias Heindel, Barbara König, and Pawel Sobocinski. Processes for adhesive rewriting systems. In Luca Aceto and Anna Ingólfssdóttir, editors, *FOSSACS 2006*, volume 3921 of *LNCS*, pages 202–216. Springer, 2006.
- 5 Paolo Baldan, Fabio Gadducci, and Ugo Montanari. Concurrent rewriting for graphs with equivalences. In Christel Baier and Holger Hermanns, editors, *CONCUR 2006*, volume 4137 of *LNCS*, pages 279–294. Springer, 2006.

- 6 Paolo Baldan, Fabio Gadducci, and Pawel Sobocinski. Adhesivity is not enough: Local Church-Rosser revisited. In Filip Murlak and Piotr Sankowski, editors, *MFCS 2011*, volume 6907 of *LNCS*, pages 48–59. Springer, 2011.
- 7 Nicolas Behr, Russ Harmer, and Jean Krivine. Concurrency theorems for non-linear rewriting theories. In Fabio Gadducci and Timo Kehrer, editors, *ICGT 2021*, volume 12741 of *LNCS*, pages 3–21. Springer, 2021.
- 8 Nicolas Behr, Russ Harmer, and Jean Krivine. Fundamentals of compositional rewriting theory. *Journal of Logical and Algebraic Methods in Programming*, 135:100893, 2023.
- 9 F. Bonchi, F. Gadducci, A. Kissinger, P. Sobociński, and F. Zanasi. String diagram rewrite theory I: rewriting with Frobenius structure. *Journal of the Association for Computing Machinery*, 69(2):1–58, 2022.
- 10 R. Brown and G. Janelidze. Van Kampen theorems for categories of covering morphisms in lextensive categories. *Journal of Pure and Applied Algebra*, 119(3):255–263, 1997.
- 11 Aurelio Carboni, Stephen Lack, and Robert FC Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84(2):145–158, 1993.
- 12 D. Castelnovo. *Fuzzy algebraic theories and \mathcal{M}, \mathcal{N} -adhesive categories*. PhD thesis, University of Udine, 2023.
- 13 D. Castelnovo, F. Gadducci, and M. Miculan. A new criterion for \mathcal{M}, \mathcal{N} -adhesivity, with an application to hierarchical graphs. In P. Bouyer and L. Schröder, editors, *FOSSACS 2022*, volume 13242 of *LNCS*, pages 205–224. Springer, 2022.
- 14 Davide Castelnovo, Fabio Gadducci, and Marino Miculan. A simple criterion for \mathcal{M}, \mathcal{N} -adhesivity. *Theoretical Computer Science*, 982:114280, 2024.
- 15 A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
- 16 A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - Part I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations. Volume 1: Foundations*, pages 163–246. World Scientific, 1997.
- 17 Andrea Corradini, Tobias Heindel, Frank Hermann, and Barbara König. Sesqui-pushout rewriting. In Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg, editors, *ICGT 2006*, volume 4178 of *LNCS*, pages 30–45. Springer, 2006.
- 18 S. Crafa, D. Varacca, and N. Yoshida. Event structure semantics of parallel extrusion in the π -calculus. In L. Birkedal, editor, *FOSSACS 2012*, volume 7213 of *LNCS*, pages 225–239. Springer, 2012.
- 19 H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.
- 20 H. Ehrig, U. Golas, A. Habel, L. Lambers, and F. Orejas. \mathcal{M} -adhesive transformation systems with nested application conditions. Part 2: Embedding, critical pairs and local confluence. *Fundamenta Informaticae*, 118(1-2):35–63, 2012.
- 21 H. Ehrig, U. Golas, A. Habel, L. Lambers, and F. Orejas. \mathcal{M} -adhesive transformation systems with nested application conditions. Part 1: Parallelism, concurrency and amalgamation. *Mathematical Structures in Computer Science*, 24(4):240406, 2014.
- 22 H. Ehrig, A. Habel, J. Padberg, and U. Prange. Adhesive high-level replacement categories and systems. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *ICGT 2004*, *LNCS*, pages 144–160. Springer, 2004.
- 23 Hartmut Ehrig, Annegret Habel, and Francesco Parisi-Presicce. Basic results for two types of high-level replacement systems. In Michel Bauderon and Andrea Corradini, editors, *GET-GRATS Closing Workshop 2001*, volume 51 of *ENTCS*, pages 127–138. Elsevier, 2001.
- 24 Hartmut Ehrig and Hans-Jörg Kreowski. Parallelism of manipulations in multidimensional information structures. In Antoni W. Mazurkiewicz, editor, *MFCS 1976*, volume 45 of *LNCS*, pages 284–293. Springer, 1976.

- 25 Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *SWAT 1973*, pages 167–180. IEEE Computer Society, 1973.
- 26 Hartmut Ehrig and Ulrike Prange. Weak adhesive high-level replacement categories and systems: A unifying framework for graph and Petri net transformations. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen*, volume 4060 of *LNCS*, pages 235–251. Springer, 2006.
- 27 F. Gadducci. Graph rewriting and the π -calculus. *Mathematical Structures in Computer Science*, 17(3):1–31, 2007.
- 28 R. Garner and S. Lack. On the axioms for adhesive and quasiadhesive categories. *Theory and Applications of Categories*, 27(3):27–46, 2012.
- 29 A. Habel and D. Plump. \mathcal{M}, \mathcal{N} -adhesive transformation systems. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *ICGT 2012*, volume 7562 of *LNCS*, pages 218–233. Springer, 2012.
- 30 T. Heindel. *A category theoretical approach to the concurrent semantics of rewriting*. PhD thesis, Universität Duisburg–Essen, 2009.
- 31 P.T. Johnstone, S. Lack, and P. Sobociński. Quasitoposes, quasiadhesive categories and Artin glueing. In T. Mossakowski, U. Montanari, and M. Haverdaen, editors, *CALCO 2007*, volume 4624 of *LNCS*, pages 312–326. Springer, 2007.
- 32 S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *RAIRO-Theoretical Informatics and Applications*, 39(3):511–545, 2005.
- 33 S. Lack and P. Sobociński. Toposes are adhesive. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *ICGT 2006*, volume 4178 of *LNCS*, pages 184–198. Springer, 2006.
- 34 Henrich Lauko, Lukás Korencik, and Peter Goodman. On the optimization of equivalent concurrent computations. *CoRR*, abs/2208.06295, 2022.
- 35 J.J. Lévy. Optimal reductions in the lambda-calculus. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980.
- 36 S. MacLane. *Categories for the working mathematician*. Springer, 2013.
- 37 Antoni W. Mazurkiewicz. Trace theory. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets 1986*, volume 255 of *LNCS*, pages 279–324. Springer, 1986.
- 38 J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
- 39 M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
- 40 Roy Overbeek, Jörg Endrullis, and Aloïs Rosset. Graph rewriting and relabeling with PBPO⁺: A unifying theory for quasitoposes. *Journal of Logical and Algebraic Methods in Programming*, 133:100873, 2023.
- 41 I. Phillips, I. Ulidowski, and S. Yuen. Modelling of bonding with processes and events. In Gerhard W. Dueck and D. Michael Miller, editors, *RC 2013*, volume 7948 of *LNCS*, pages 141–154. Springer, 2013.
- 42 Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–29, 2021.

A Properties of \mathcal{M} -adhesive categories

This first appendix is devoted to the proofs of a few well-known results about \mathcal{M} -adhesive categories. Observe that our notion of \mathcal{M} -adhesivity follows [20, 21] and is different from the one of [1]. What is called \mathcal{M} -adhesivity in the latter paper corresponds to our strict \mathcal{M} -adhesivity. Moreover, in [1] the class \mathcal{M} is assumed to be only stable under pullbacks. However, if \mathcal{M} contains all split monos, then stability under pushouts can be deduced from the other axioms [12, Prop. 5.1.21].

A.1 Some results on \mathcal{A} -stable and \mathcal{A} -Van Kampen squares

We start proving some general results regarding \mathcal{A} -Van Kampen and \mathcal{A} -stable squares. Let us begin recalling some classical results about pullbacks and pushouts [36].

- **Lemma A.1.** *Let \mathbf{X} be a category, and consider the diagram below, then the following hold*
1. *if the right square is a pullback, then the whole rectangle is a pullback if and only if the left square is one;*
 2. *if the left square is a pushout, then the whole rectangle is a pushout if and only if the right square is one.*

$$\begin{array}{ccccc}
 X & \xrightarrow{f} & Y & \xrightarrow{g} & Z \\
 a \downarrow & & \downarrow b & & \downarrow c \\
 A & \xrightarrow{h} & B & \xrightarrow{k} & C
 \end{array}$$

The following proposition establishes a key property of \mathcal{A} -Van Kampen squares with a mono as a side: they are not only pushouts, but also pullbacks [8, 22, 32].

- **Proposition A.2.** *Let \mathcal{A} be a class of arrows stable under pushouts and containing all the isomorphisms. If the square below is \mathcal{A} -Van Kampen and $m: A \rightarrow C$ is mono and belongs to \mathcal{A} , then the square is a pullback and n is a monomorphism.*

$$\begin{array}{ccc}
 A & \xrightarrow{g} & B \\
 m \downarrow & & \downarrow n \\
 C & \xrightarrow{f} & D
 \end{array}$$

The previous proposition allows us to establish the following result.

- **Lemma A.3.** *Let \mathcal{A} be a class of arrows stable under pullbacks, pushouts and containing all isomorphisms. Suppose that the left square below is \mathcal{A} -Van Kampen, while the vertical faces in the right cube are pullbacks. Suppose moreover that $m: A \rightarrow C$ and $d: D' \rightarrow D$ are mono and that d belongs to \mathcal{A} . Then $d \leq n$ if and only if $c \leq m$.*

$$\begin{array}{ccc}
 A & \xrightarrow{g} & B \\
 m \downarrow & & \downarrow n \\
 C & \xrightarrow{f} & D
 \end{array}
 \qquad
 \begin{array}{ccccc}
 & & A' & \xrightarrow{g'} & B' \\
 & m' \swarrow & \downarrow f' & \swarrow n' & \\
 C' & \xrightarrow{a} & D' & & \\
 c \downarrow & & \downarrow d & & \downarrow b \\
 C & \xrightarrow{m} & A & \xrightarrow{g} & B \\
 & & \downarrow d & & \swarrow n \\
 & & D & &
 \end{array}$$

- **Remark A.4.** Recall that, given two monos $m : M \rightarrow X$ and $n : N \rightarrow X$ with the same codomain, $m \leq n$ means that there exists a, necessarily unique and necessarily mono, $k : M \rightarrow N$ fitting in the triangle below.

Notice, moreover, that if $m \leq n$ and $n \leq m$, then the arrow $k : M \rightarrow N$ is an isomorphism.

$$\begin{array}{ccc}
 M & \xrightarrow{k} & N \\
 m \searrow & & \swarrow n \\
 & X &
 \end{array}$$

Finally, we show that \mathcal{A} -stable pushouts enjoy a *pullback-pushout decomposition* property.

► **Proposition A.5.** *Let \mathbf{X} be a category and \mathcal{A} a class of arrows stable under pullbacks. Suppose that, in the diagram below, the whole rectangle is an \mathcal{A} -stable pushout and the right square a pullback. If the arrow k is in \mathcal{A} and it is a monomorphism, then both squares are pushouts.*

$$\begin{array}{ccccc} X & \xrightarrow{f} & Y & \xrightarrow{g} & Z \\ a \downarrow & & \downarrow b & & \downarrow c \\ A & \xrightarrow{h} & B & \xrightarrow{k} & C \end{array}$$

A.2 Useful properties of \mathcal{M} -adhesive categories

We are now going to apply the results of the previous section to \mathcal{M} -adhesive categories in order to establish some *high-level replacement properties* [19, 21, 22]. A first important result that can be immediately established, with the aid of Proposition A.2, is the following one.

► **Proposition A.6.** *Let \mathbf{X} be an \mathcal{M} -adhesive category. Then \mathcal{M} -pushouts are pullbacks.*

From Proposition A.6, in turn, we can derive the following corollaries.

► **Corollary A.7.** *In a \mathcal{M} -adhesive category \mathbf{X} , every $m \in \mathcal{M}$ is a regular mono.*

The following result now follows at once noticing that a regular monomorphism which is also epic is automatically an isomorphism.

► **Corollary A.8.** *If \mathbf{X} is an \mathcal{M} -adhesive categories, then every epimorphism in \mathcal{M} is an isomorphism. In particular, every adhesive category \mathbf{X} is balanced: if a morphism is monic and epic, then it is an isomorphism.*

► **Lemma A.9** (*\mathcal{M} -pushout-pullback decomposition*). *Let \mathbf{X} be an \mathcal{M} -adhesive category and suppose that, in the diagram below, the whole rectangle is a pushout and the right square a pullback. Then the following statements hold*

1. *if a belongs to \mathcal{M} and k is a mono, then both squares are pushouts and pullbacks;*
2. *if f and k are in \mathcal{M} , then both squares are pushouts and pullbacks.*

$$\begin{array}{ccccc} X & \xrightarrow{f} & Y & \xrightarrow{g} & Z \\ a \downarrow & & \downarrow b & & \downarrow c \\ A & \xrightarrow{h} & B & \xrightarrow{k} & C \end{array}$$

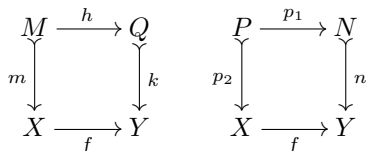
Let us turn our attention to pushout complements.

► **Definition A.10** (*Pushout complement*). *Let $f: X \rightarrow Y$ and $g: Y \rightarrow Z$ be two composable arrows in a category \mathbf{X} . A pushout complement for the pair (f, g) is a pair (h, k) with $h: X \rightarrow W$ and $k: W \rightarrow Z$ such that the square below is a pushout.*

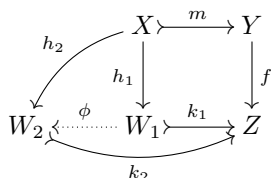
$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ h \downarrow & & \downarrow g \\ W & \xrightarrow{k} & Z \end{array}$$

Working in an \mathcal{M} -adhesive category guarantees that pushout complements are unique.

► **Lemma A.11.** *Let $f: X \rightarrow Y$ be an arrow in an \mathcal{M} -adhesive category \mathbf{X} and suppose that the left square below is a pushout while the right one is a pullback, with $m: M \rightarrow X$ and $n: N \rightarrow Y$ in \mathcal{M} . Then $n \leq k$ if and only if $p_2 \leq m$.*



► **Corollary A.12** (Uniqueness of pushout complements). *Let \mathbf{X} be a \mathcal{M} -adhesive category. Given $m: X \rightarrow Y$ in \mathcal{M} and $f: Y \rightarrow Z$, let (h_1, k_1) and (h_2, k_2) be pushout complements of m and f depicted below. Then there exists a unique isomorphism $\phi: W_1 \rightarrow W_2$ making the diagram below commutative.*

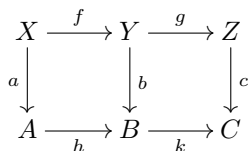


B \mathcal{M} -adhesivity is not enough

In Section 3 we introduced the notion of strong enforcing left-linear rewriting system, a class that contains all the linear rewriting systems. This result can be further refined: in [6] a class \mathbb{B} of (quasi-)adhesive category is defined for which the local Church-Rosser Theorem holds for left-linear rewriting system. In our language, this means that every left-linear rewriting system based on a category in \mathbb{B} is strong enforcing. This section is devoted to repropose, and slightly generalise, the results of [6] to our context.

► **Definition B.1.** *Let \mathcal{M} be a class of monos in a category \mathbf{X} , closed under composition, containing all isomorphisms and stable under pullbacks and pushouts. Suppose that the diagram below is given and the whole rectangle is a pushout. We say that \mathbf{X} satisfies*

- *the \mathcal{M} -mixed decomposition property if, whenever k belongs to \mathcal{M} and the right half of the diagram below is a pullback, then the left one is a pushout;*
- *the \mathcal{M} -pushout decomposition property if whenever a, b and c belongs to \mathcal{M} and the right half of the diagram below is a pushout, then its left half is a pushout too.*



The class of categories of type \mathbb{B} is closed under the same constructions of Theorem 2.5. This can be deduced at once from the fact that in such categories pullbacks and pushouts are computed component-wise.

► **Lemma B.2.** *Let \mathbf{X} be a category satisfying the \mathcal{M} -mixed and \mathcal{M} -pushout decomposition properties, then the following hold*

1. *for every object X , \mathbf{X}/X satisfies the \mathcal{M}/X -mixed and the \mathcal{M}/X -pushout decomposition properties, while X/\mathcal{M} -adhesive satisfies their X/\mathcal{M} variants, where*

$$\mathcal{M}/X := \{m \in \mathcal{A}(\mathbf{X}/X) \mid m \in \mathcal{M}\} \qquad X/\mathcal{M} := \{m \in \mathcal{A}(X/\mathbf{X}) \mid m \in \mathcal{M}\}$$

11:22 Left-Linear Rewriting in Adhesive Categories

2. for every small category \mathbf{Y} , the category $\mathbf{X}^{\mathbf{Y}}$ satisfies the $\mathcal{M}^{\mathbf{Y}}$ -mixed and the $\mathcal{M}^{\mathbf{Y}}$ -pushout decomposition properties, where

$$\mathcal{M}^{\mathbf{Y}} := \{\eta \in \mathcal{A}(\mathbf{X}^{\mathbf{Y}}) \mid \eta_Y \in \mathcal{M} \text{ for every } Y \in \mathbf{Y}\}$$

The following result shows that the mixed and pushout decomposition properties guarantee that every independence pair is strong.

► **Theorem B.3.** *Let \mathbf{X} be an \mathcal{M} -adhesive category with all pushouts and satisfying the \mathcal{M} -mixed and the \mathcal{M} -pushout decomposition properties. Then every left-linear rewriting system on \mathbf{X} is strong enforcing.*

Proof. Let $\mathcal{D} = \{\mathcal{D}_i\}_{i=0}^1$ be the derivation made by two sequentially independent derivations

$$\begin{array}{ccccccc} L_0 & \xleftarrow{l_0} & K_0 & \xrightarrow{r_0} & R_0 & & L_1 & \xleftarrow{l_1} & K_1 & \xrightarrow{r_1} & R_1 \\ m_0 \downarrow & & k_0 \downarrow & & h_0 \searrow & & i_1 \swarrow & & i_0 \swarrow & & k_1 \downarrow & & h_1 \downarrow \\ G_0 & \xleftarrow{f_0} & D_0 & \xrightarrow{g_0} & G_1 & \xleftarrow{f_1} & D_1 & \xrightarrow{g_1} & G_2 \end{array}$$

We have to show that (i_0, i_1) is a strong independence pair. Now, by hypothesis \mathbf{X} has all pushouts, thus the only thing to show is that the squares below are pushouts (the third one is the usual pullback of $f_1: D_1 \rightarrow G_0$ along $g_0: D_0 \rightarrow G_1$)

$$\begin{array}{ccc} K_0 \xrightarrow{r_0} R_0 & K_1 \xrightarrow{l_1} L_1 & P \xrightarrow{p_0} D_0 \\ u_0 \downarrow & u_1 \downarrow & p_1 \downarrow \\ P \xrightarrow{p_1} D_1 & P \xrightarrow{p_0} D_0 & D_1 \xrightarrow{f_1} G_1 \end{array}$$

To see this, consider the following two diagrams

$$\begin{array}{ccc} K_0 \xrightarrow{u_0} P \xrightarrow{p_0} D_0 & & K_1 \xrightarrow{u_1} P \xrightarrow{p_1} D_1 \\ r_0 \downarrow & p_1 \downarrow & l_1 \downarrow \\ R_0 \xrightarrow{i_0} D_1 \xrightarrow{f_1} G_1 & & L_1 \xrightarrow{i_1} D_0 \xrightarrow{g_0} G_1 \\ h_0 \curvearrowright & & m_1 \curvearrowright \end{array}$$

The thesis now follows from the \mathcal{M} -mixed and the \mathcal{M} -pushout decomposition property. ◀

Our next step is to identify sufficient conditions for a category \mathbf{X} to satisfy the mixed and \mathcal{M} -pushout decomposition properties.

► **Definition B.4.** *Let \mathbf{X} be an \mathcal{M} -adhesive category, the pair $(\mathbf{X}, \mathcal{M})$ is of type \mathbb{B} if*

1. every arrow in \mathcal{M} is a coproduct coprojection;
2. \mathbf{X} has all pushouts;
3. \mathbf{X} has strict initial objects and for every X the unique arrow $?_X: 0 \rightarrow X$ belongs to \mathcal{M} ;
4. all pushouts are \mathcal{M} -stable.

► **Remark B.5.** It is worth to examine more closely conditions 1 and 3 of the above definition.

- Let $m_0: X_0 \rightarrow Y$ be in \mathcal{M} , the first condition means that there exists $m_1: X_1 \rightarrow Y$ such that $(Y, \{m_i\}_{i=0}^1)$ is a coproduct. This, together with property 3, entails that every coprojection in a coproduct is in \mathcal{M} . This follows since any coproduct $(X_1 + X_2, \{\iota_{X_i}\}_{i=0}^1)$ fits in a pushout diagram as the one below.

- \mathbf{X} has strict initial objects if it has initial objects and every arrow $f : X \rightarrow 0$ is an isomorphism. Notice that if initial objects are strict, then $?_X : 0 \rightarrow X$ is mono for every X : indeed for every pair $f, g : Y \rightrightarrows 0$ then, by strictness, Y is initial and so $f = g$.

$$\begin{array}{ccc}
 0 & \xrightarrow{?_{X_1}} & X_1 \\
 \downarrow ?_{X_2} & & \downarrow \iota_{X_1} \\
 X_2 & \xrightarrow{\iota_{X_2}} & X_1 + X_2
 \end{array}$$

► **Example B.6.** The category **Set** of sets and functions is of type \mathbb{B} . Similarly, the category **Inj** of sets and injective functions is quasiadhesive and, with regular monos, of type \mathbb{B} .

Categories of type \mathbb{B} satisfy a property resembling *extensivity* [11].

► **Proposition B.7.** Let $(\mathbf{X}, \mathcal{M})$ be a pair of type \mathbb{B} . Then for every diagram as the one below, in which the bottom row is a coproduct cocone and the vertical arrows are in \mathcal{M} , the top row is a coproduct if and only if the two squares are pullbacks.

$$\begin{array}{ccccc}
 A & \xrightarrow{f} & C & \xleftarrow{g} & B \\
 \downarrow r & & \downarrow s & & \downarrow t \\
 X & \xrightarrow{\iota_X} & X + Y & \xleftarrow{\iota_Y} & Y
 \end{array}$$

Proof. Consider the cube below, in which the back faces are pullbacks. The bottom faces is an \mathcal{M} -Van Kampen pushout, thus the top face is a pushout if and only if the front faces are pullbacks. By strictness of 0 , $a : I \rightarrow 0$ is an isomorphism, so that I is initial, therefore the \mathcal{M} -Van Kampen condition reduces to the request that $(C, \{f, g\})$ is a coproduct cocone if and only if the front faces are pullbacks, as claimed.

$$\begin{array}{ccccc}
 & & I & \xrightarrow{k} & B \\
 & \swarrow h & \downarrow & \searrow g & \downarrow t \\
 A & \xrightarrow{f} & C & & \\
 \downarrow r & & \downarrow s & & \\
 X & \xrightarrow{\iota_X} & X + Y & \xrightarrow{\iota_Y} & Y \\
 & \swarrow ?_X & \downarrow ?_Y & \swarrow \iota_Y & \\
 & 0 & & &
 \end{array}$$

◀

The previous result entails the following one, needed to show that in any pair $(\mathbf{X}, \mathcal{M})$ of type \mathbb{B} , the category \mathbf{X} satisfies the \mathcal{M} -mixed and \mathcal{M} -pushout decomposition properties.

► **Proposition B.8.** Let $(\mathbf{X}, \mathcal{M})$ be a pair of type \mathbb{B} , and suppose that the square below is an \mathcal{M} -pullback. Then there exists $E \in \mathbf{X}$, $e : E \rightarrow C$ in \mathcal{M} , $\phi : E \rightarrow B_1$ such that $(C, \{m, e\})$ is a coproduct and $g = f + \phi$. Moreover, such a square is a pushout if and only if ϕ is an isomorphism.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B_0 \\
 \downarrow m & & \downarrow \iota_{B_0} \\
 C & \xrightarrow{g} & B_0 + B_1
 \end{array}$$

11:24 Left-Linear Rewriting in Adhesive Categories

► **Lemma B.9.** *Let $(\mathbf{X}, \mathcal{M})$ be a pair of type \mathbb{B} , then \mathbf{X} satisfies the \mathcal{M} -mixed and \mathcal{M} -pushout decomposition properties.*

Proof. \mathcal{M} -mixed decomposition property. This follows at once from Proposition A.5.

\mathcal{M} -pushout-decomposition property. Using Propositions B.7 and B.8 and the fact that arrows in \mathcal{M} are coproduct coprojections, we can reduce to prove the property for diagrams as the one below. By hypothesis and Proposition B.8, φ and $\varphi \circ \phi$ are both isomorphisms, therefore ϕ is an isomorphism too and we can conclude again using Proposition B.8.

$$\begin{array}{ccccc}
 X & \xrightarrow{f} & Y & \xrightarrow{g} & Z \\
 \downarrow \iota_X & & \downarrow \iota_Y & & \downarrow \iota_Z \\
 X + A & \xrightarrow{f+\phi} & Y + B & \xrightarrow{g+\varphi} & Z + C \\
 & \searrow & \xrightarrow{(g \circ f) + (\varphi \circ \phi)} & &
 \end{array}$$

◀

► **Definition B.10.** *Let \mathbf{X} be an \mathcal{M} -adhesive category, we say that the pair $(\mathbf{X}, \mathcal{M})$ is of type \mathbb{B}^+ if (at least) one of the following holds*

1. $(\mathbf{X}, \mathcal{M})$ is of type \mathbb{B} ;
2. \mathbf{X} is \mathbf{Y}/Y and $\mathcal{M} = \mathcal{N}/Y$ for some $(\mathbf{Y}, \mathcal{N})$ of type \mathbb{B}^+ and $Y \in \mathbf{Y}$;
3. \mathbf{X} is Y/\mathbf{Y} and $\mathcal{M} = Y/\mathcal{N}$ for some $(\mathbf{Y}, \mathcal{N})$ of type \mathbb{B}^+ and $Y \in \mathbf{Y}$;
4. \mathbf{X} is $\mathbf{Y}^{\mathbf{A}}$ and $\mathcal{M} = \mathcal{N}^{\mathbf{A}}$ for some category \mathbf{A} and $(\mathbf{Y}, \mathcal{N})$ of type \mathbb{B}^+ .

► **Example B.11.** The pair $(\mathbf{Graph}, \text{Mono}(\mathbf{Graph}))$ of graphs and their monos is of type \mathbb{B}^+ but not of type \mathbb{B} . Indeed, not every monomorphism of graphs is a coproduct coprojection.

From Lemmas B.2 and B.9 we can now deduce at once the following.

► **Corollary B.12.** *For every $(\mathbf{X}, \mathcal{M})$ of type \mathbb{B}^+ , the category \mathbf{X} has the \mathcal{M} -mixed and \mathcal{M} -pushout decomposition properties.*

Using Theorem B.3 we finally get the main result of our appendix.

► **Corollary B.13.** *Let $(\mathbf{X}, \mathcal{M})$ be a pair of type \mathbb{B}^+ , then every left-linear rewriting system is strong enforcing.*

History-Determinism vs Fair Simulation

Udi Boker   

Reichman University, Herzliya, Israel

Thomas A. Henzinger   

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Karoliina Lehtinen   

CNRS, LIS, Aix-Marseille Univ., France

Aditya Prakash   

University of Warwick, Coventry, UK

Abstract

An automaton \mathcal{A} is history-deterministic if its nondeterminism can be resolved on the fly, only using the prefix of the word read so far. This mild form of nondeterminism has attracted particular attention for its applications in synthesis problems. An automaton \mathcal{A} is guidable with respect to a class C of automata if it can fairly simulate every automaton in C , whose language is contained in that of \mathcal{A} . In other words, guidable automata are those for which inclusion and simulation coincide, making them particularly interesting for model-checking.

We study the connection between these two notions, and specifically the question of when they coincide. For classes of automata on which they do, deciding guidability, an otherwise challenging decision problem, reduces to deciding history-determinism, a problem that is starting to be well-understood for many classes.

We provide a selection of sufficient criteria for a class of automata to guarantee the coincidence of the notions, and use them to show that the notions coincide for the most common automata classes, among which are ω -regular automata and many infinite-state automata with safety and reachability acceptance conditions, including vector addition systems with states, one-counter nets, pushdown-, Parikh-, and timed-automata.

We also demonstrate that history-determinism and guidability do not always coincide, for example, for the classes of timed automata with a fixed number of clocks.

2012 ACM Subject Classification Theory of computation \rightarrow Formal languages and automata theory

Keywords and phrases History-Determinism

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.12

Related Version *Full Version*: <https://arxiv.org/abs/2407.08620> [3]

Funding *Udi Boker*: Israel Science Foundation grant 2410/22

Thomas A. Henzinger: ERC-2020-AdG 101020093 (VAMOS)

Karoliina Lehtinen: ANR QUASY 23-CE48-0008-01

Aditya Prakash: Chancellors' International Scholarship from the University of Warwick and Centre for Discrete Mathematics and Its Applications (DIMAP)

1 Introduction

Language inclusion between automata is a key problem in verification: given an automaton representing a program and another representing a specification, language inclusion of the former in the latter captures precisely whether all executions of the program satisfy the specification. Unfortunately, in the presence of nondeterminism, inclusion is algorithmically hard. For instance, for regular automata it is PSPACE-hard on both finite and infinite words.



© Udi Boker, Thomas A. Henzinger, Karoliina Lehtinen, and Aditya Prakash;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 12; pp. 12:1–12:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

12:2 History-Determinism vs Fair Simulation

Fair simulation is a more syntactic approximation of inclusion, defined by the simulation game [12]. In this game, one player, in the role of the spoiler, builds, transition by transition, a run in one of the automata, say \mathcal{A} , while the other, in the role of the duplicator, chooses at each turn a matching transition in the other automaton, say \mathcal{B} . The second player's task is to build a run that is accepting if the first player's run is accepting. If the duplicator has a winning strategy, then \mathcal{B} is said to simulate \mathcal{A} , which, in particular, implies that \mathcal{A} 's language is included in \mathcal{B} 's language. Simulation, due to its local and syntactic nature, is generally easier to check than inclusion: for instance it is in PTIME for nondeterministic Büchi automata. As a result, automata for which language inclusion and simulation coincide are particularly well-suited for model-checking. We call such automata *guidable*, after a similar notion used previously as an alternative to determinism for tree automata [9]. Despite their clear usefulness for model checking, guidable automata have so far been mostly used as a tool, but not studied much in their own right, with the notable exception of [17].

Guidability is not easy to decide: it is contingent on an automaton simulating a potentially infinite number of language-included automata. We would like to have, whenever possible, a characterisation that is more amenable to algorithmic detection.

Deterministic automata are of course always guidable, and so are *history-deterministic* automata. These are mildly nondeterministic automata, in which nondeterministic choices are permitted, but can only depend on the word read so far, rather than the future of the word. These automata have received a fair bit of attention recently due, in particular, to their applications in synthesis problems [6]. In general, they offer an interesting compromise between the power of nondeterministic automata and the better algorithmic properties of deterministic ones. In particular, they can simulate all equivalent, or language-contained, automata as they only need the history to resolve nondeterministic choice in the best possible way – in other words, they are guidable. In fact, at first it might appear that history-determinism and guidability should coincide; indeed, this is the case if we consider guidability with respect to all labelled transition systems [13, Theorem 4]. However, there are also classes of automata for which this is not the case.

Guidability and history-determinism coinciding on a class C of automata is equivalent to the description “for every automaton $\mathcal{A} \in C$ that is not history-deterministic, there is some $\mathcal{A}' \in C$ that is language-included in \mathcal{A} but that \mathcal{A} does not simulate” (D). Then it is easy enough to hand-pick automata to build classes where guidability and history-determinism do not coincide (for example, a class of inclusion-incomparable automata that are not all history-deterministic). However, as we will see, there are also more natural classes of automata, such as timed automata with a bounded number of clocks, for which guidability and history-determinism differ.

The characterisation (D) is too abstract to be much use for analysing the usual automata classes we are interested in. We therefore prove that several more concrete criteria (Theorem 1) imply that guidability and history-determinism coincide, and use each of these in a comprehensive analysis of standard automata classes. Roughly, each of the criteria describes some sufficient closure properties which guarantee the existence of \mathcal{A}' from description (D). If some automaton can simulate another automaton that is sufficiently difficult to simulate (for example a deterministic one, since they simulate all equivalent automaton), then it must be history-deterministic; as a result, a class of automata having sufficient closure properties (such as closure under determinisation), implies that guidability and history-determinism coincide. The challenge is to identify, for a variety of different classes of automata C , an automaton that is sufficiently difficult to simulate, while remaining in C .

In order to discuss our sufficient criteria in more detail, we need to start with a couple of key notions, the first of which is the *1-token game* and the second is the *1-token ghost*.

History-determinism of an automaton is tricky and expensive to decide directly [11]; As an alternative, Bagnol and Kuperberg used k -token games as potential characterisations for history-determinism [2]. Roughly, they resemble a (fair) simulation-like game, played on a single automaton, where one player, Eve (in the role of Duplicator), must build transition-by-transition a run on a word dictated letter-by-letter by Adam, who, after each of Eve's choices, also builds k runs transition-by-transition. Eve then wins if her run is accepting whenever Adam's run is accepting. Bagnol and Kuperberg showed that for Büchi automata, the 2-token game indeed characterises history-determinism, which means that deciding history-determinism for Büchi automata is in PTIME [2]. Since then, the 1- or 2-token games have been shown to characterise history-determinism for various automata classes, including coBüchi [4], DSum, LimInf, and LimSup automata [5]. These games contrast with the *letter game*, a game which always characterises history-determinism, but which is often challenging to solve directly [11].

In this work, we make heavy use of token-games, this time to understand the connection between history-determinism and guidability. In particular, we extend token games to be played over two automata (Definition 3), separating between Eve's and Adam's automata, and define, given an automaton \mathcal{A} , that an automaton \mathcal{A}' is a *1-token ghost* of \mathcal{A} if \mathcal{A}' is language-equivalent to \mathcal{A} and Eve wins the 1-token game between \mathcal{A}' and \mathcal{A} . For some classes of automata such a ghost is easy to construct, while for other classes it might not exist due to lacking closure properties.

With these notions, we can now state our criteria.

► **Theorem 1.** *The notions of history-determinism and guidability coincide for a class C of labelled transitions systems (LTSs) if at least one of the following holds:*

1. *Determinisation. C is closed under history-determinism, i.e., for each nondeterministic LTS in C , there is a language-equivalent history-deterministic (or deterministic) LTS in C as well. (Lemma 4)*
2. *1-token ghost. 1-token games characterise history-determinism in C , and C is closed under 1-token ghost. (Lemma 8)*
3. *Strategy ghost. For every $\mathcal{A} \in C$ that is not history-deterministic, there is a deterministic LTS \mathcal{B} over the alphabet of transitions of \mathcal{A} , such that \mathcal{B} recognises the plays of a winning strategy of Adam in the letter game on \mathcal{A} , and \mathcal{B} , projected onto the alphabet of \mathcal{A} , has a 1-token ghost in C . (Lemma 9)*

We prove the criteria of Theorem 1 in Section 4, and use them in Section 5 to show that the notions of history-determinism and guidability coincide for numerous classes of automata, listed in Table 1. We also provide counter-examples of classes for which history-determinism and guidability do not coincide, as listed in Table 1, and elaborated on in Section 6. We restrict our analysis to automata over infinite words, which are better behaved in this context, and discuss how to adapt our techniques for finite words in Section 7.

A practical corollary of our result is that guidability is decidable, with the complexity of deciding history-determinism, for ω -regular automata (EXPTIME for parity automata, PTIME for Büchi and coBüchi), safety and reachability timed automata (EXPTIME) and visibly pushdown automata (EXPTIME). For details on the complexity of these procedures, we refer the reader to a recent survey [6].

12:4 History-Determinism vs Fair Simulation

■ **Table 1** History-determinism vs guidability.

Automata Class	HD = Guidability by
ω -regular	<i>Determinisation or Strategy ghost</i>
Fixed-index parity (e.g., Büchi), Weak	<i>Strategy ghost</i> Corollary 10
Linear	<i>Strategy ghost</i> Theorem 19. (Not ghost-closed Theorem 18)
Safety & Reachability pushdown automata, VASS, timed, one-counter automata, one-counter net, Parikh	<i>1-token ghost</i> Theorem 17 and Corollary 15
VPA with any ω -regular acceptance condition	<i>Strategy ghost</i> Theorem 16
Classes for which HD \neq guidability:	
– Büchi automata with a bounded number of states. Theorem 20	
– Timed automata with a bounded number of clocks. Theorem 21	
Notable classes for which we leave the question HD \neq guidability open:	
– PDA/OCA/OCN/Timed automata with general ω -regular acceptance	

2 Preliminaries

We use \mathbb{N} and \mathbb{N}^+ to denote the set of non-negative and positive integers respectively. We use $[i..j]$ to denote the set $\{i, \dots, j\}$ of integers, and $[i]$ for the set $[1..i]$. An alphabet Σ is a non-empty set of letters. A finite or infinite word is a finite or infinite sequence of letters from Σ respectively. We let ε be the empty word, and Σ_ε the set $\Sigma \cup \{\varepsilon\}$. The set of all finite (resp. infinite) words is denoted by Σ^* (resp. Σ^ω). A *language* is a set of words.

2.1 Labelled transition systems

A *labelled transition system* (LTS) $\mathcal{A} = (\Sigma, Q, \iota, \Delta, \alpha)$ consists of a potentially infinite alphabet Σ , a potentially infinite state-space Q , an initial state $\iota \in Q$, a labelled transition relation $\Delta \subseteq Q \times \Sigma_\varepsilon \times Q$, and a set of accepting runs α , where a run ρ is a (finite or infinite) sequence of transitions starting in ι and following Δ . We may write $q \xrightarrow{\sigma} q'$ instead of $(q, \sigma, q') \in \Delta$ for $\sigma \in \Sigma_\varepsilon$. Given a finite run $\rho = q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_k} q_k$ on the word $v = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdot \dots \cdot \sigma_k \in \Sigma_\varepsilon^{k+1}$, we write $q_0 \xrightarrow{v, \rho} q_k$ to denote that ρ is a transition sequence on the word v that starts at q_0 and ends at q_k . An LTS is *deterministic* if for every state q and letter $\sigma \in \Sigma$, there is at most one transition $q \xrightarrow{\sigma} q'$ from q on the σ , and there are no transitions on ε .

A word $w \in \Sigma^\omega$ is accepted by an LTS \mathcal{A} if there is an accepting run of \mathcal{A} on w . The language $L(\mathcal{A})$ of \mathcal{A} is the set of words that it accepts. An LTS \mathcal{A} is contained in an LTS \mathcal{B} , denoted by $\mathcal{A} \leq \mathcal{B}$, if $L(\mathcal{A}) \subseteq L(\mathcal{B})$, while \mathcal{A} and \mathcal{B} are equivalent, denoted by $\mathcal{A} \equiv \mathcal{B}$ if $L(\mathcal{A}) = L(\mathcal{B})$.

A *transducer* is like an LTS without acceptance condition and ε -transitions but with an output labelling: it is given by a tuple $(\Sigma_I, \Sigma_O, Q, \iota, \Delta, \gamma)$, where Σ_I and Σ_O are the input and output alphabets, respectively, $\Delta \subseteq Q \times \Sigma_I \times Q$ is the transition relation, and $\gamma : \Delta \rightarrow \Sigma_O$ is the output function. A *strategy* in general is a deterministic transducer. It is finite memory if the transducer has a finite state space.

2.2 Automata

We briefly recall the automata types considered here, which we assume to operate on infinite words unless stated otherwise, and leave the formal definitions to the appendix.

LTSs are represented concisely by various automata. An automaton \mathcal{A} induces an LTS \mathcal{B} , whose states are the configurations of \mathcal{A} , and whose runs are the same as \mathcal{A} 's runs. If \mathcal{A} 's states and configurations are the same, as is the case with ω -regular automata which we define below, then \mathcal{B} is identical to \mathcal{A} , but with the acceptance condition given by a set of runs (as opposed to an ω -regular conditions). If the configurations of \mathcal{A} contain additional data, as is the case for example with pushdown automata, then \mathcal{B} and \mathcal{A} have different states. Notice that \mathcal{A} is deterministic iff \mathcal{B} is. The acceptance condition of \mathcal{A} induces the acceptance condition on \mathcal{B} .

In a parity condition, α assigns priorities in \mathbb{N} to either states or transitions, and a run is accepting if the highest priority that occurs infinitely often along it is even. An $[i, j]$ -parity automaton, for $i < j$ two natural numbers is a parity automaton whose priorities are in $[i, j]$. A parity automaton is said to be a *weak automaton* if there is no cycle in the automaton containing both an even and odd priority. In a reachability condition, some states are labelled final; a run accepts if it reaches a final state. In a safety automaton, some states are labelled safe; a run accepts if it remains within the safe region.

In a nondeterministic ω -regular automaton $(\Sigma, Q, \iota, \delta, \alpha)$, Σ and Q are finite, and the acceptance condition α is based on the set of states (or transitions) visited infinitely often along a run. A timed automaton (TWA) $(\Sigma, Q, \iota, C, \delta, \alpha)$ has a set of clocks C and its transitions are guarded by inequalities between the clock values and can reset clocks. It reads timed words, which consist of letters of a finite alphabet Σ paired with delays from \mathbb{R} . A timed automaton recognises a timed language, for example “at some point an event occurs twice exactly one time-unit apart.”

We also handle pushdown automata, one-counter automata, vector addition systems with states, one-counter nets and Parikh automata with reachability and safety acceptance. We define these classes of infinite state systems uniformly in Section 5.2 as classes of finite state automata with transitions that modify an infinite *content space*. A visibly pushdown automaton (VPA) is a pushdown automaton without ε -transitions, in which the input alphabet is partitioned into `pop`, `push` and `noop` letters that induce only transitions that `pop`, `push` and have no effect on the stack respectively.

3 History-determinism, simulation and related games

Different simulation-like games that capture either a relationship between automata or properties of a single automaton are at the heart of our technical developments. In this section we go over the various games – both known and newly defined – that will be played throughout this article, and which allow us to connect guidability and history-determinism. These games are all based on Adam (the spoiler) building a word letter by letter, and potentially a run in an automaton over that word, while his opponent Eve (the duplicator) tries to build a single accepting run transition by transition. The differences between these games are based on whether they are played on one or two automata, whether Adam picks transitions, and if so, whether he does it before Eve. The winning condition is similar in all cases: Eve's run must be accepting whenever Adam's run is accepting, or if Adam does not have a run, then whenever the word built by Adam's moves is in the language of a specified automaton. This results in three styles of games: (i) simulation games, played on two automata, in which Adam plays before Eve, and each builds a run in their respective automaton; (ii) token-games,

which can be played on one or two automata, in which Adam first declares the letter, and then Eve plays her transition before Adam plays his; and (iii) the letter game, played on a single automaton, in which Adam only chooses letters and does not build a run at all.

Fair simulation between two LTSs (or automata) is captured by the simulation game defined below:

► **Definition 2** (Simulation game). Consider LTSs $\mathcal{A} = (\Sigma, Q_A, \iota_A, \Delta_A, \alpha_A)$ and $\mathcal{B} = (\Sigma, Q, \iota_B, \Delta_B, \alpha_B)$. The simulation game $\text{Sim}(\mathcal{B}, \mathcal{A})$ between \mathcal{B} and \mathcal{A} is a two player-game played between Adam and Eve with positions in $Q_A \times Q_B$ which starts at the position $(p_0, q_0) = (\iota_A, \iota_B)$. At round i of the play, for $i \geq 0$, when the position is (p_i, q_i) :

- Adam picks $\sigma_i \in \Sigma$ and a transition (or transition sequence, in the presence of ε -transitions) $p_i \xrightarrow{\sigma_i, \rho_i} p_{i+1}$ in \mathcal{A} ;
- Eve picks a transition (or transition sequence, in the presence of ε -transitions) $q_i \xrightarrow{\sigma_i, \rho'_i} q_{i+1}$ in \mathcal{B} ; they proceed from (p_{i+1}, q_{i+1}) .

An infinite play produces a run ρ_A in \mathcal{A} consisting of transitions chosen by Adam and a run ρ_E in \mathcal{B} of transitions chosen by Eve, both on $\sigma_0\sigma_1\sigma_2\cdots$. We say that Eve wins the play if ρ_E is accepting or ρ_A is rejecting.

If Eve has a winning strategy in this game, we say that \mathcal{B} simulates \mathcal{A} , denoted by $\text{Sim}(\mathcal{B}, \mathcal{A})$. It is easy to observe that if \mathcal{B} simulates \mathcal{A} , then $L(\mathcal{A}) \subseteq L(\mathcal{B})$. An LTS \mathcal{A} is *guidable* with respect to a class C of LTSs if \mathcal{A} simulates every LTS \mathcal{A}' in C that satisfies $L(\mathcal{A}') \subseteq L(\mathcal{A})$.

The following *letter game* based definition of history-determinism, was introduced by Henzinger and Piterman [11], and coincides with Colcombet's notion of translation strategies [8].

Given an LTS \mathcal{A} , the *letter game* on \mathcal{A} , denoted by $\text{HD}(\mathcal{A})$ is similar to the simulation game except that instead of playing transitions in an automaton, Adam just chooses letters and builds a word w , letter by letter, which should be in the language of \mathcal{A} , while Eve tries to build a run of \mathcal{A} over w . More precisely, the letter game starts with Eve's token at the initial state ι , and proceeds in rounds. At round i , where Eve's token is at q_i :

- Adam chooses a letter σ_i in the alphabet Σ of \mathcal{A} ;
- Eve chooses a transition $q_i \xrightarrow{\sigma_i} q_{i+1}$ (or a transition sequence $q_i \xrightarrow{\sigma_i, \rho_i} q_{i+1}$ in the presence of ε -transitions) of \mathcal{A} over σ_i ; Eve's token moves to q_{i+1} .

In the limit, a play consists of the word $w = \sigma_0\sigma_1\cdots$ and the run $\rho = \rho_0 \cdot \rho_1 \cdot \rho_2 \cdots$. Eve wins the play if $w \notin L(\mathcal{A})$ or ρ is accepting. We say that \mathcal{A} is history-deterministic (HD) if Eve has a winning strategy in the letter game over \mathcal{A} .

Token games are known to characterise history-determinism on various classes of automata [2, 5, 6]. We generalise token games to be played on two LTSs below, which makes them more akin to a variation of simulation. This will help us relate simulation and history-determinism in Section 4. We only use the 1-token version here.

► **Definition 3** (1-token games over two LTSs (or automata)). Consider LTSs \mathcal{A}' and \mathcal{A} with initial states p_0 and q_0 respectively. In the 1-token game on \mathcal{A}' and \mathcal{A} denoted by $G_1(\mathcal{A}', \mathcal{A})$, Eve has a token with which she constructs a run in \mathcal{A}' , and Adam has a token with which he constructs a run in \mathcal{A} . The game proceeds in rounds, and at round i of the play with token positions (p_i, q_i) , for each $i \geq 0$:

- Adam chooses a letter σ_i in Σ ;
- Eve chooses a transition (or a transition sequence, in the presence of ε -transitions) $p_i \xrightarrow{\sigma_i, \rho'_i} p_{i+1}$ in \mathcal{A}' ;
- Adam chooses a transition (or transition sequence, in the presence of ε -transitions) $q_i \xrightarrow{\sigma_i, \rho_i} q_{i+1}$; the game proceeds from (p_{i+1}, q_{i+1}) .

An infinite play produces a word $w = \sigma_0 \dots$, a sequence of transitions ρ_E of \mathcal{A}' chosen by Eve, and a sequence of transitions ρ_A in \mathcal{A} chosen by Adam. Eve wins if ρ_E is accepting or if ρ_A is rejecting.

A strategy for Eve here is a function $s : (\Delta^+)^* \times \Sigma \rightarrow (\Delta')^*$, where Σ is the alphabet of \mathcal{A} and \mathcal{A}' , and Δ and Δ' are the sets of transitions of \mathcal{A} and \mathcal{A}' , respectively. When clear from context, $G_1(\mathcal{A}', \mathcal{A})$ also denotes the claim that Eve has a winning strategy in the game $G_1(\mathcal{A}', \mathcal{A})$. As an automaton and its induced LTS have the same runs, $G_1(\mathcal{A}', \mathcal{A})$ holds for automata \mathcal{A} and \mathcal{A}' iff it holds for their induced LTSs. We also write $G_1(\mathcal{A})$ for $G_1(\mathcal{A}, \mathcal{A})$.

Note that the 1-token game and the simulation game differ in one key aspect: in the simulation game, Adam plays first, and Eve can use the information of the transition to inform her choice, while in the 1-token game, Eve must choose her transition based only on the *letter* chosen by Adam, who plays his transition after Eve.

4 Criteria for when History-Determinism = Guidability

We now provide criteria which guarantee that history-determinism and guidability coincide for a class of LTSs. In Section 5, we use these to show the coincidence of the two notions for many standard automata classes.

4.1 Closure under (history-)determinism

A first observation is that if every LTS \mathcal{A} can be determinised within the class C , or even if there exists an equivalent HD LTS \mathcal{A}' within C then \mathcal{A} is HD if and only if it is guidable.

► **Lemma 4.** *History-determinism and guidability coincide for any class C of LTSs in which the languages expressed by history-deterministic (or deterministic) LTSs are the same as languages expressed by nondeterministic LTSs.*

The proof is simple: one direction is trivial (HD always implies guidability) and conversely, if an automaton \mathcal{A} is not HD, then it cannot simulate any equivalent HD automaton, implying that \mathcal{A} is not guidable.

Various examples of such classes are provided in Section 5.1, as summarised in Table 1. In particular, the general class of all labelled-transition systems [13], safety/reachability visibly pushdown automata [1], as well as finite-state automata on finite words (NFAs), and co-Büchi, Parity, Rabin, Streett, and Muller automata on infinite words. Yet, this is not the case for Büchi automata or parity automata with a fixed parity index. History-determinism is also strictly less expressive than nondeterminism for pushdown automata, Parikh automata, timed automata and one-counter nets.

4.2 Via token games

For classes that are not closed under determinisation, we have to find some other type of automaton that is difficult to simulate. To do so, we revisit token games, previously used to help decide history-determinism, to relate history-determinism and guidability. Recall that we extended the definition of 1-token games, so that they are played on two automata, rather than one. In the next definition, we use this extended notion of 1-token game to identify, for each automaton \mathcal{A} , an automaton \mathcal{A}' such that Eve wins the the 1-token game on \mathcal{A} if and only if \mathcal{A} simulates \mathcal{A}' .

► **Definition 5** (1-token ghost). *An LTS (or an automaton) \mathcal{A}' is a 1-token ghost of an LTS \mathcal{A} , denoted by $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$, if $\mathcal{A}' \equiv \mathcal{A}$ and $G_1(\mathcal{A}', \mathcal{A})$.*

To show that the ghost automaton has the property that $\text{Sim}(\mathcal{A}, \mathcal{A}')$ if and only if Eve wins $G_1(\mathcal{A}, \mathcal{A})$, we compose the strategies in $\text{Sim}(\mathcal{A}, \mathcal{A}')$ and $G_1(\mathcal{A}', \mathcal{A})$.

► **Lemma 6.** *Consider LTSs \mathcal{A} and \mathcal{A}' , such that \mathcal{A} simulates \mathcal{A}' and $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$. Then Eve wins $G_1(\mathcal{A})$.*

Proof. Let s_{sim} be a winning strategy of Eve in the simulation game between \mathcal{A} and \mathcal{A}' , and s' her winning strategy in $G_1(\mathcal{A}', \mathcal{A})$. Eve then has a winning strategy s in $G_1(\mathcal{A})$: she plays the strategy s_{sim} in $\text{Sim}(\mathcal{A}, \mathcal{A}')$ against her imaginary friend Berta, who plays the strategy s' in $G_1(\mathcal{A}', \mathcal{A})$ against Adam. In more detail: In each round i of the game $G_1(\mathcal{A})$, Adam chooses a transition sequence ρ_{i-1} in \mathcal{A} on σ_{i-1} (except for the first round) on his token and a letter σ_i , then Berta chooses the transition sequence $\rho_i^B = s'(\rho_0 \dots \rho_{i-1}, \sigma_i)$ over the letter σ_i in \mathcal{A}' on her token in $G_1(\mathcal{A}', \mathcal{A})$, and then Eve chooses the transition sequence $\rho_i = s_{sim}(\rho_0^B \dots \rho_{i-1}^B)$ in \mathcal{A} .

The run built by Eve with the strategy s is accepting if the run built by Berta is, which is in turn accepting if Adam's run is. Hence, s is a winning strategy for Eve in $G_1(\mathcal{A})$. ◀

Then, for classes in which token games characterise history-determinism and which are closed under the ghost relation, guidability and history-determinism coincide.

► **Definition 7.** *A class C of LTSs is closed under 1-token ghost if for every $\mathcal{A} \in C$ there exists $\mathcal{A}' \in C$ such that $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$.*

► **Lemma 8.** *Given a class C of LTSs closed under 1-token ghost for which G_1 characterises history-determinism, history-determinism and guidability coincide for C .*

Proof. Being HD always implies guidability, so one direction is easy. For the other direction, if \mathcal{A} simulates every LTS $\mathcal{A}' \in C$, such that $\mathcal{A}' \leq \mathcal{A}$, then in particular it simulates an LTS $\mathcal{A}' \in C$, such that $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$, as C is closed under the 1-token ghost. By Lemma 6, Eve wins $G_1(\mathcal{A})$, implying that \mathcal{A} is HD, as G_1 characterises history-determinism in C . ◀

A 1-token ghost is often easy to build, by delaying nondeterministic choices by one letter (Definition 12), as shown in Section 5.2 for pushdown automata, one-counter automata, vector addition system with states, one-counter nets and Parikh automata.

For some automata classes, however, showing closure under 1-token ghosts is trickier: for VPA the stack action must occur as the letter is read, and for timed automata configuration updates are sensitive to the current timestamp. We handle these complications in Section 5.3. We can also only use Lemma 8 with respect to automata classes for which 1-token games characterise history-determinism, which is not the case for parity automata or ω -VPA [2].

4.3 Via Adam's strategy in the letter game

As we will see in detail in Section 5.4, some classes, such as linear automata, are neither closed under 1-token ghost nor determinisation, so there is no hope for the above criteria to apply. Our final criterion is an alternative which, instead of requiring all automata to admit a 1-ghost, builds a difficult-to-simulate automaton from Adam's winning strategy in the letter game. The intuition is that Adam's winning strategy in the letter game on an automaton \mathcal{A} captures behaviour that is difficult for \mathcal{A} to simulate, so if we can turn Adam's strategy into an automaton (which will be language-contained in \mathcal{A} since Adam must play a word in the language of \mathcal{A}), then this automaton will not be simulated by \mathcal{A} . To build this automaton, we first project an automaton \mathcal{B} recognising Adam's winning plays from his strategy onto the alphabet of \mathcal{A} , to obtain an automaton \mathcal{B}_Σ that recognises the words played by Adam's strategy. Then, by taking the 1-token ghost of \mathcal{B}_Σ , we obtain an automaton \mathcal{B}' against which the simulation game is essentially the letter game against Adam's strategy. If the resulting automaton is always still in the class C , guidability and history-determinism coincide.

► **Lemma 9.** *History-determinism and guidability coincide for classes C of LTSs in which, for each $\mathcal{A} \in C$ that is not history-deterministic, there is a deterministic LTS \mathcal{B} over the alphabet of transitions of \mathcal{A} that recognises the plays of a winning strategy of Adam in the letter game on \mathcal{A} , and \mathcal{B} , projected onto the alphabet of \mathcal{A} , has a 1-token ghost in C .*

Proof. Consider a winning strategy τ of Adam in the letter game on \mathcal{A} , and let \mathcal{B} be a deterministic LTS that recognises the plays of τ , seen as runs of \mathcal{A} . Let \mathcal{B}_Σ be the projection of \mathcal{B} onto Σ : it is otherwise like \mathcal{B} , except that its alphabet is Σ instead of the transitions $\Delta_{\mathcal{A}}$ of \mathcal{A} and as a result it has additional nondeterminism. Crucially, every transition in \mathcal{B} is still a transition in \mathcal{B}_Σ . Given a sequence of transitions $t_0 t_1 \dots t_i \in \Delta_{\mathcal{A}}^*$, we call $t''_0 t''_1 \dots t''_i$ its run in \mathcal{B} , which is uniquely defined since \mathcal{B} is deterministic. Note that this sequence of transitions is also a run over the word of $t_0 t_1 \dots t_i$ in \mathcal{B}_Σ . This also extends to infinite sequences. Since every run accepted by \mathcal{B} is a play winning for Adam in the letter game over \mathcal{A} , their projection onto Σ must be in $L(\mathcal{A})$, so $L(\mathcal{B}_\Sigma) \subseteq L(\mathcal{A})$.

Now, let \mathcal{B}' be the 1-token ghost of \mathcal{B}_Σ , witnessed by a strategy s_1 of Eve in the game $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$. Assume, towards contradiction, that $\text{Sim}(\mathcal{A}, \mathcal{B}')$ via some strategy s_{sim} . We construct a strategy s of Eve in the letter game on \mathcal{A} that is winning against τ , in which Eve plays s_{sim} against her imaginary friend Berta in $\text{Sim}(\mathcal{A}, \mathcal{B}')$, who in turn is playing s_1 against Adam in $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$.

In more detail, Adam begins by playing σ_0 according to τ in the letter game on \mathcal{A} ; Berta responds with a transition (or sequence of transitions in the presence of ε -transitions) $\rho'_0 = s_1(\sigma_0)$; and then Eve responds with $s(\sigma_0) = \rho_0 = s_{\text{sim}}(\rho'_0)$. On the i^{th} round, when Adam chooses the letter σ_i , after the sequence $\rho_0 \dots \rho_{i-1}$ of Eve's moves in the letter game, and the sequence $\rho''_0, \dots, \rho''_{i-1}$ of transitions in \mathcal{B}_Σ , which is the Σ -projection of the unique run of \mathcal{B} on $\rho_0 \dots \rho_{i-1}$, viewed as a word over $\Delta_{\mathcal{A}}$, Berta makes the move $\rho'_i = s_1(\rho''_0, \dots, \rho''_{i-1}, \sigma_i)$ in $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$, and then Eve the move $s(\sigma_0, \rho_0, \dots, \rho_{i-1}, \sigma_i) = \rho_i = s_{\text{sim}}(\rho'_0, \dots, \rho'_{i-1})$ in $\text{Sim}(\mathcal{A}, \mathcal{B}')$ and in the letter game.

We argue that s is winning against τ . Indeed, the run $\rho''_0 \rho''_1 \dots$ in \mathcal{B}_Σ must be accepting since the sequence of transitions $\rho_0 \rho_1 \dots$ that Eve plays agrees with τ . Then, since Berta is playing a winning strategy in $G_1(\mathcal{B}', \mathcal{B}_\Sigma)$, the sequence $\rho'_0 \rho'_1 \dots$ is also an accepting run over the same word. Since Eve is playing a winning strategy in $\text{Sim}(\mathcal{A}, \mathcal{B}')$, the sequence $\rho_0 \rho_1 \dots$ is also an accepting run over the same word. This contradicts τ being a winning strategy for Adam. We conclude that \mathcal{A} does not simulate \mathcal{B}' and is therefore not guidable. ◀

Lemma 9 can be applied to various automata classes, as summarised in Table 1, including ω -regular automata with an $[i, j]$ -parity acceptance condition (Section 5.1), linear automata (Section 5.4), and visibly pushdown automata (Section 5.3).

This concludes our criteria. Concerning the necessity of each criterion, notice that:

- The first criterion (Theorem 1.1) is not subsumed by the others, as demonstrated with the class of all LTSs – it is closed under determinization [7, Theorem 3.4], but G_1 does not characterise history-determinism, which is required for the second criterion, and the letter game need not always be determined, which is required for the third.
- The second criterion (Theorem 1.2) does not imply the first one, as demonstrated by, for instance, safety pushdown automata [10, Theorem 4.1]. The implication from the second criterion to third criterion is unclear, however, and connects to the case of PDA, where the strategies for the players in letter game are not yet understood [10, Section 6].
- Finally, the third criterion (Theorem 1.3) is not subsumed by the other two, as evident from the case of linear automata (Section 5.4).

5 Automata Classes for which History-Determinism = Guidability

5.1 Straightforward cases

By Theorem 1.1, history-determinism and guidability coincide for all automata classes closed under determinisation, including: regular automata (NFAs); VPAs on finite words; ω -regular automata [15]; co-Büchi [16]; and subclasses of ω -regular automata whose deterministic fragment is ω -regular-complete, such as parity, Rabin, Streett, Muller, and Emerson-Lei. Some subclasses of ω -regular automata are not closed under determinisation, e.g., Büchi automata, but as long as they subsume safety automata, we can build on the fact that Adam’s letter-game strategies are recognised by deterministic safety automata, and apply Theorem 1.3: since safety automata are determinisable they are closed under 1-token ghost.

► **Corollary 10.** *History-determinism and guidability coincide for classes of ω -regular automata with an $[i, j]$ -parity acceptance condition, as well as for the class of weak automata.*

5.2 Uniform infinite state systems

In this section, we show that the notions of history-determinism and guidability coincide on the following classes with safety and reachability acceptance conditions: pushdown automata, one-counter automata, vector addition system with states, one-counter nets and Parikh automata. We take a unified approach by defining all of these classes as cases of “uniform automata classes”, and showing that the two notions coincide for such classes (Theorem 14).

These uniform automata classes are specified by a content space \mathcal{C} (e.g., stack contents) and a set \mathcal{K} of partial functions $f : \mathcal{C} \rightarrow \mathcal{C}$ that contains the identity function f_{id} that maps each element in \mathcal{C} to itself (e.g., stack updates). The class specified by \mathcal{C} and \mathcal{K} contains all the automata $\mathcal{A} = (\Sigma, Q, \iota, c_0, \Delta, F_A, F_C)$ that have a finite alphabet Σ , a finite state space Q , and finitely many transitions $(q, \sigma, f, q') \in \Delta$, labelled by a letter $\sigma \in \Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and a function $f \in \mathcal{K}$. The automaton \mathcal{A} induces an LTS that has states $(q, c) \in Q \times \mathcal{C}$, with transitions $(q, c) \xrightarrow{\sigma} (q', c')$, such that (q, σ, f, q') is a transition in \mathcal{A} and $f(c) = c'$.

The acceptance semantics of an automaton in such a class is specified by a set of accepting states $F_A \subseteq Q$ and a set of accepting contents $F_C \subseteq \mathcal{C}$. We will often desire some structure on F_C , so we impose the restriction that F_C belongs to a set $\mathcal{S} \subseteq \mathcal{P}(\mathcal{C})$ of subsets of \mathcal{C} . We call “ $(\mathcal{C}, \mathcal{K}, \mathcal{S})$ -automata” the class of all automata $\mathcal{A} = (\Sigma, Q, \iota, c_0, \Delta, F_A, F_C)$ as above with $F_C \in \mathcal{S}$. Safety automata require an accepting run to have all states in F_A and all content in F_C . We distinguish between synchronous reachability that requires an accepting run to reach an accepting state and an accepting content at the same time, and asynchronous reachability that requires an accepting run to just reach an accepting state and an accepting content, not necessarily at the same time.

► **Definition 11.** *A class of automata is uniform if it can be specified as $(\mathcal{C}, \mathcal{K}, \mathcal{S})$ -automata with either safety, synchronous reachability, or asynchronous reachability acceptance semantic.*

We show that uniform automata classes are closed under 1-token ghost by explicitly constructing for each automaton \mathcal{A} in the class a 1-token ghost, called $\text{Delay}(\mathcal{A})$, inspired by Prakash and Thejaswini [18, Lemma 11]. For each run in \mathcal{A} , we will have a run in $\text{Delay}(\mathcal{A})$ that lags one transition behind. This one-step lag is implemented by storing the previous letter in the state space of \mathcal{A} , in addition to the state of \mathcal{A} ; transitions are then taken based on the previous letter, while reading the current letter, which is now stored.

► **Definition 12** (Delay). Let $\mathcal{A} = (\Sigma, Q, \iota, c_0, \Delta, F_Q, F_C)$ be an automaton in a uniform class $(\mathcal{C}, \mathcal{K}, \mathcal{S})$. The automaton $\text{Delay}(\mathcal{A}) = (\Sigma, Q', \iota', c_0, \Delta', F'_Q, F_C)$ is the Delay of \mathcal{A} , where

1. The set of states Q' is $(Q \times \Sigma) \cup \{\iota'\}$
2. The set of transitions Δ' is given by the union of:
 - $\{(\iota', \sigma, f_{id}, (\iota, \sigma)) \mid \sigma \in \Sigma\}$
 - $\{((q, \sigma), \sigma', f, (q', \sigma')) \mid \sigma, \sigma' \in \Sigma, \text{ and } (q, \sigma, f, q') \in \Delta\}$
 - $\{(q, \sigma), \varepsilon, f, (q', \sigma) \mid \sigma \in \Sigma, \text{ and } (q, \varepsilon, f, q') \in \Delta\}$
3. The set F'_Q consists of state of the form (q, σ) such that $q \in F_Q$, and ι' if $\iota \in F_Q$.

The automaton $\text{Delay}(\mathcal{A})$ has the same acceptance semantics as \mathcal{A} (safety, synchronous reachability or asynchronous reachability).

► **Lemma 13.** Given an automaton \mathcal{A} in a uniform automata class C , the automaton $\text{Delay}(\mathcal{A})$ is in C and is a 1-token ghost of \mathcal{A} .

We show that G_1 characterises history-determinism on all uniform automata classes in the full version [3][Lemma 22], by reducing to safety and reachability LTSs [7]. With Lemma 13 and Theorem 1.2, we get that history-determinism and guidability coincide on all uniform automata classes.

► **Theorem 14.** History-determinism and guidability coincide for uniform automata classes.

It now suffices to represent various automata classes as uniform ones to show that guidability and history-determinism coincide on them. For pushdown and one-counter automata, vector addition systems and one-counter nets, as well as for Parikh automata, the contents are the counter or stack contents, while the update functions are their increments, decrements, pops and pushes. The update partial functions also implement which parts of the contents can be used to enable transitions: for example, for pushdown automata, the partial functions are either defined for all contents where the stack is empty, or undefined for all such contents; for Parikh automata, the contents do not influence which transitions are enabled, so the functions are fully defined. (Formal definitions can be found in the full version [3][Section C.1].)

► **Corollary 15.** History-determinism and guidability coincide for the classes of pushdown automata, one-counter automata, vector addition systems with states, one-counter nets with safety and reachability acceptance conditions, and for Parikh automata with safety, synchronous reachability and asynchronous reachability acceptance conditions.

Non-uniform classes

The class of visibly pushdown automata is not uniform, as there are additional constraints on transitions, namely the kind of function that changes content depends on the letter seen. Timed automata also do not constitute a uniform class, since the alphabet is infinite as it consists of all timed letters, and the clock valuations are updated according to both the transition (resets) and the delay of the input letter. In Section 5.4, we consider linear automata: these are Büchi automata that have no cycles apart from self-loops. Linear automata also does not form a uniform class, since they restrict the state-space. In what follows, we give alternative constructions of 1-token-ghosts for these classes. The case of linear automata is trickier, as we show that it is not closed under 1-token ghost. We therefore use, in Section 5.4, a more involved argument that allows us to use Theorem 1.3.

5.3 Visibly pushdown and timed automata

Visibly pushdown automata over infinite words (ω -VPAs) are neither (history-) determinisable, nor does G_1 characterise history-determinism on them. Nevertheless, we can use Theorem 1.3 to show that history-determinism and guidability coincide for this class.

► **Theorem 16.** *History-determinism and guidability coincide for the class of visibly pushdown automata with any ω -regular acceptance condition.*

Proof sketch. First we show that the class is closed under 1-token ghost. Like in the previous cases, we build an automaton that executes the same transitions, but one step later. The technical challenge is executing transitions with a delay, as an ω -VPA must respect the stack discipline of the input alphabet. We overcome this by maintaining a “semantic stack” that consists of the actual stack and one additional letter that is embedded in the state space and stores, when necessary, the letter that should have been in the top of the stack.

Then, we describe the letter-game for an ω -VPA as a game on a visibly pushdown arena with a “stair parity” acceptance condition, to show that Adam’s winning strategies can be implemented by ω -VPA transducers. We then turn this transducer into a deterministic ω -VPA recognising the plays that agree with Adam’s strategy, and apply Theorem 1.3. ◀

We turn to safety and reachability timed automata, for which we apply Theorem 1.2, yet with a specially tailored Delay construction.

► **Theorem 17.** *For the class of timed automata with safety or reachability acceptance conditions, the notions of history-determinism and guidability coincide.*

Proof sketch. The goal is to simulate such an automaton \mathcal{A} with a delay, as in Definition 12. Yet, the difficulty is that delaying a clock-reset by a step will affect the value of the clock for future comparisons, and there is no delaying of the passage of time. Hence a naive construction would end up recognising the timed language of words in which timestamps are shifted by one. We overcome the difficulty by duplicating in the 1-token ghost construction each clock of \mathcal{A} , using one copy for comparisons in guards and the other to simulate retroactive resets. In addition, the state-space stores the effect of the previous delay, by remembering the corresponding region, that is, how the timestamp compares to existing clocks and constants. With this construction, and the G_1 -characterisation of history-determinism for safety and reachability automata, we complete the proof. ◀

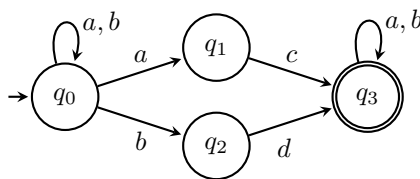
5.4 Linear automata

A *linear* (also called *very weak*) automaton is a Büchi automaton in which all cycles are self loops. (In linear automata, the acceptance condition does not really matter, since over an automaton with only self loops, all the standard ω -regular acceptance conditions coincide.)

First, observe that linear automata are not closed under (history-)determinisation. (The standard Büchi automaton over the alphabet $\Sigma = \{a, b\}$ recognizing the language of finitely many a ’s is linear.) We show that they are also not closed under 1-token ghost, by proving that the linear automaton depicted in Figure 1 admits no 1-token ghost in the class.

► **Theorem 18.** *The class of linear automata is not closed under 1-token ghost.*

Proof. Let \mathcal{A} be the linear automaton depicted in Figure 1, and assume toward contradiction that there is a linear automaton \mathcal{A}' , satisfying $1\text{-TokenGhost}(\mathcal{A}', \mathcal{A})$, witnessed by a winning strategy s of Eve in $G_1(\mathcal{A}', \mathcal{A})$.



■ **Figure 1** A linear automaton which admits no linear automaton that is a 1-token ghost of it.

In a play π_1 of $G_1(\mathcal{A}', \mathcal{A})$ in which Eve plays along s and Adam plays $(ab)^*$ while staying in q_0 , at some points of time $2k-1$ and $2k$, Eve must remain in the same state q' of \mathcal{A}' after Adam chose the letters a and b , respectively, since \mathcal{A}' is linear.

In a play π_2 of $G_1(\mathcal{A}', \mathcal{A})$ in which Eve plays along s and Adam starts with $(ab)^{k-1}a$ while staying in q_0 , Eve reaches, as per the previous claim, the state q' of \mathcal{A}' . Then, if Adam continues with the word ca^ω , while moving from q_0 to q_1 (over the previous a) and then to q_3 (over c), Eve has some accepting continuation run ρ from the state q' over the suffix ca^ω , since s is winning for Eve in $G_1(\mathcal{A}', \mathcal{A})$ and Adam's run is accepting.

Thus, there is an accepting run of \mathcal{A}' on the word $w = (ab)^k ca^\omega$, following in the first $2k$ steps the run of Eve in the play π_1 , reaching the state q' , and then in the next steps following her accepting continuation in π_2 . Yet, \mathcal{A} does not have an accepting run on w , contradicting the equivalence of \mathcal{A} and \mathcal{A}' , and thus the assumption that $\text{1-TokenGhost}(\mathcal{A}', \mathcal{A})$. ◀

Yet, history-determinism and guidability do coincide for the class of linear automata. The underlying reason is that when a linear automaton \mathcal{A} is not history-deterministic, Adam's winning strategy in the letter game can be adapted to a linear automaton that does have a 1-token ghost within the class of linear automata, thus satisfying Theorem 1.3.

▶ **Theorem 19.** *History-determinism and guidability coincide for the class of linear automata.*

Proof sketch. History-determinism implies guidability with respect to all classes. For the other direction, consider a linear automaton $\mathcal{A} = (\Sigma, Q, \iota, \Delta, \alpha)$ that is not HD, and let $\mathcal{M} = (\Delta, \Sigma, M, m_0, \Delta_M : M \times \Delta \rightarrow M, \gamma : M \rightarrow \Sigma)$ be a deterministic finite-state transducer representing a finite-memory winning strategy s_M of Adam in the letter game.

We then build, by taking a product of \mathcal{M} and \mathcal{A} , a deterministic safety automaton \mathcal{P} , that recognises the set of plays that can occur in the letter game on \mathcal{A} if Adam plays according to s_M . From \mathcal{P} , we take its projection \mathcal{N} onto the alphabet Σ of \mathcal{A} . \mathcal{N} need not be linear, but we adapt it into a linear \mathcal{N}' that will still correspond to a winning strategy of Adam in the letter game. \mathcal{N}' will thus constitute a projection of a deterministic automaton \mathcal{P}' onto the alphabet Σ , where \mathcal{P}' is over the alphabet of transitions of \mathcal{A} and recognises the plays of a winning strategy of Adam in the letter game. Once achieving that, we can apply the Delay construction on \mathcal{N}' – it will not introduce, in this case, non-self cycles, since the states of \mathcal{N}' (as the projection of the states of \mathcal{P}'), have outgoing transitions only on a single letter. Hence, we satisfy Theorem 1.3, proving the stated claim. ◀

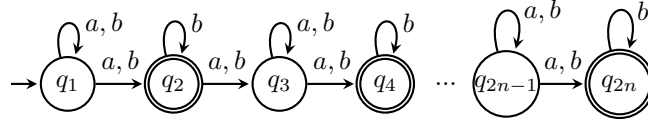
6 Automata Classes for which History-Determinism \neq Guidability

In this section we study classes which admit guidable automata that are not history-deterministic. They offer insight into how, in practice, the criteria can fail to hold, and witness that even on arguably natural automata classes, guidability and history determinism do not necessarily coincide. The main reason for the equivalence between the notions to fail for these classes is a bound on the allowed resources – the number of states in the first class and the number of clocks in the second.

12:14 History-Determinism vs Fair Simulation

Our first example of when history-determinism and guidability differ are Büchi automata with a bounded number of states, witnessed by the automaton in Figure 2.

► **Theorem 20.** *For every $n \in \mathbb{N}^+$, history-determinism and guidability are distinct notions for the class of Büchi automata with up to $2n$ states.*



■ **Figure 2** A Büchi automaton that accepts words with a finite number of a 's. To simulate any equivalent small enough Büchi automaton \mathcal{B} , Eve moves to the next accepting state once the other automaton is in a maximally strongly connected component with an accepting state. The size constraint on \mathcal{B} , and the observation that a such a component can not both have a transition on a and an accepting state guarantees that this strategy wins in the simulation game. However, \mathcal{B} is not history-deterministic.

This counter-example is simple, but quite artificial. We proceed with a class which is, arguably, more natural: timed automata with a bounded number of clocks.

► **Theorem 21.** *History-determinism and guidability are distinct notions for the class \mathbb{T}_k of timed-automata over finite words with at most k clocks, for each $k \in \mathbb{N}$.*

Proof sketch. We consider the language of infinite words in which there are k event pairs that occur exactly one time-unit apart both before and after the first occurrence of a $\$$ letter. Then, the guidable automaton for this language can freely reset its clocks until the $\$$ -separator, which allows it to ensure it tracks all delays tracked by a smaller automaton with up to k clocks. Crucially, any automaton that only accepts words in this language must keep track of k clock values when the separator occurs, as otherwise, it will also accept some word in which the second of matching pair of event is shifted a little. ◀

7 Conclusions

We have presented sufficient conditions for a class of automata to guarantee the coincidence of history-determinism and guidability, and used them to show that this is the case for many standard automata classes on infinite words. As a result, we get algorithms to decide guidability for many of these classes. Guidable automata allow for simple model-checking procedures, and once guidability check is simple, one can take advantage of it whenever applicable. For example, consider a specification modelled by a Büchi or coBüchi automaton \mathcal{A} . Model-checking whether a system \mathcal{S} satisfies \mathcal{A} is *PSPACE*-hard. Using our results, one can check first in *PTime* whether \mathcal{A} is guidable, and in the fortunate cases that it is, conclude the model checking in *PTime*, by checking whether \mathcal{A} simulates \mathcal{S} . We have also demonstrated automata classes for which guidability and history-determinism do not coincide.

We believe that our positive results extend to additional automata classes, such as register automata [14], which behave quite similarly to timed automata. Furthermore, we believe them to extend to additional families of automata classes:

- *Finite words.* We have focused on automata over infinite words, which in this context, are better behaved. Ends of words bring additional complications to our constructions, but overall we believe our approach to be amenable to the analysis of finite word automata.

- *Quantitative automata.* In quantitative automata, transitions carry additional information in the form of weights. As a result, there is an additional difference between the letter game and simulation game, which makes extending our analysis to the quantitative setting particularly relevant. We believe that many of our techniques adapt to that setting.

One could argue that for model-checking, the more interesting property is whether a (not necessarily safety) automaton is guidable by just safety automata, since we typically represent specifications by safety automata. Interestingly, this property often coincides with guidability w.r.t. the full class of automata, as demonstrated in by our third criterion (Theorem 1.3): if Adam’s strategies in the letter game can be translated into automata, these automata are safety ones, and therefore guidability w.r.t. safety automata is just as hard as guidability w.r.t. the full class of automata with the more complex acceptance conditions.

References

- 1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.
- 2 Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, page 16, 2018.
- 3 Udi Boker, Thomas A. Henzinger, Karoliina Lehtinen, and Aditya Prakash. History-determinism vs fair simulation, 2024. arXiv:2407.08620.
- 4 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *arXiv preprint*, 2020. arXiv:2002.07278.
- 5 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative automata. In *FoSSaCS*, pages 120–139, 2022. A submitted journal version is available at arXiv:2110.14308.
- 6 Udi Boker and Karoliina Lehtinen. When a little nondeterminism goes a long way: An introduction to history-determinism. *ACM SIGLOG News*, 10(1):24–51, 2023. doi:10.1145/3584676.3584682.
- 7 Sougata Bose, Thomas A. Henzinger, Karoliina Lehtinen, Sven Schewe, and Patrick Totzke. History-deterministic timed automata, 2023. arXiv:2304.03183.
- 8 Thomas Colcombet. Fonctions régulières de coût. *Habilitation à diriger les recherches, École Doctorale de Sciences Mathématiques de Paris Centre*, 2013.
- 9 Thomas Colcombet and Christof Löding. The non-deterministic mostowski hierarchy and distance-parity automata. In *Proc. of ICALP*, volume 5126, pages 398–409, 2008. doi:10.1007/978-3-540-70583-3_33.
- 10 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. *Log. Methods Comput. Sci.*, 20(1), 2024. doi:10.46298/LMCS-20(1:3)2024.
- 11 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proceedings of CSL*, pages 395–410, 2006.
- 12 Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. *Inf. Comput.*, 173(1):64–81, 2002. doi:10.1006/inco.2001.3085.
- 13 Thomas A. Henzinger, Karoliina Lehtinen, and Patrick Totzke. History-deterministic timed automata. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPICs*, pages 14:1–14:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

12:16 History-Determinism vs Fair Simulation

- 14 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 15 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.
- 16 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω -words. *Theoretical Computer Science*, 32:321–330, 1984.
- 17 Damian Niwiński and Michał Skrzypczak. On Guidable Index of Tree Automata. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 81:1–81:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.81.
- 18 Aditya Prakash and K. S. Thejaswini. On history-deterministic one-counter nets. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 218–239. Springer, 2023. doi:10.1007/978-3-031-30829-1_11.

The Power of Counting Steps in Quantitative Games

Sougata Bose ✉ 

University of Liverpool, UK

Rasmus Ibsen-Jensen ✉ 

University of Liverpool, UK

David Purser ✉ 

University of Liverpool, UK

Patrick Totzke ✉ 

University of Liverpool, UK

Pierre Vandenhover ✉ 

LaBRI, Université de Bordeaux, France

Abstract

We study deterministic games of infinite duration played on graphs and focus on the strategy complexity of quantitative objectives. Such games are known to admit optimal memoryless strategies over finite graphs, but require infinite-memory strategies in general over infinite graphs.

We provide new lower and upper bounds for the strategy complexity of *mean-payoff* and *total-payoff* objectives over infinite graphs, focusing on whether *step-counter strategies* (sometimes called *Markov strategies*) suffice to implement winning strategies. In particular, we show that over finitely branching arenas, three variants of lim sup mean-payoff and total-payoff objectives admit winning strategies that are based either on a step counter or on a step counter and an additional bit of memory. Conversely, we show that for certain lim inf total-payoff objectives, strategies resorting to a step counter and finite memory are not sufficient. For step-counter strategies, this settles the case of all classical quantitative objectives up to the second level of the Borel hierarchy.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Games on graphs, Markov strategies, quantitative objectives, infinite-state systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.13

Related Version *Full Version*: <https://arxiv.org/abs/2406.17482> [3]

Funding *P. Vandenhover* is funded by ANR project *G4S* (ANR-21-CE48-0010-01). *S. Bose* and *P. Totzke* acknowledge support from the EPSRC (EP/V025848/1, EP/X042596/1). Collaboration initiated via Royal Society International Exchanges grant (IES\R1\201211).

1 Introduction

Two-player (zero-sum, turn-based, perfect-information) games on graphs are an established formalism in formal verification, especially for *reactive synthesis* [1, 13]. They are used to model the interaction between a system, trying to satisfy a given *specification*, against an uncontrollable environment, assumed to act antagonistically as a worst case. We can model the system and its environment as two opposing players, called *Player 1* and *Player 2* respectively, who move a token through the graph of possible system configurations (called the *arena*). The specification is modelled as a winning condition (called *objective* henceforth), which is a set of all those interactions that the system player deems acceptable. The main algorithmic task when using this approach for formal verification is *solving* such games:



© Sougata Bose, Rasmus Ibsen-Jensen, David Purser, Patrick Totzke, and Pierre Vandenhover; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

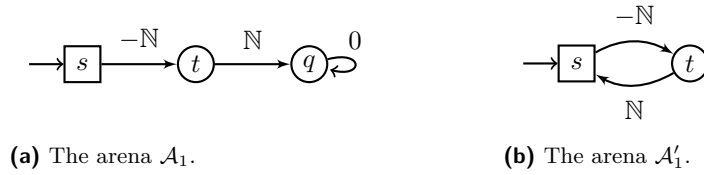
Editors: Rupak Majumdar and Alexandra Silva; Article No. 13; pp. 13:1–13:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

13:2 The Power of Counting Steps in Quantitative Games



■ **Figure 1** Arenas implementing the “match the number” game. Circles designate vertices controlled by Player 1 and squares designate Player 2. The edge labels indicate that for every $i \in \mathbb{N}$ there is a distinct edge with weight $-i$ from s to t , and $+i$ from t to q or from t to s . For \mathcal{A}_1 , consider the objective “sum of weights exceeds 0”. Player 1 can always match and thus win, but needs unbounded memory. The arena \mathcal{A}'_1 shows a repeated version for the lim sup *mean*-payoff objective.

given an arena, an objective, and an initial vertex, decide whether the system player has a *winning strategy*, which corresponds to a controller for the system that guarantees that the specification holds no matter the behaviour of the environment. Additionally, reactive synthesis aims to *synthesise* (compute a representation of) a winning strategy if one exists.

Strategy complexity. To synthesise winning strategies, it is useful to know what kind of resources “suffice”, i.e., are needed to implement a winning strategy, should one exist. This naturally depends on the model used for the interaction (the size and topology of the arena) and on the specification (the type of objective and whether probabilistic or absolute guarantees are required). We assume that strategies make decisions based on some internal memory, that stores and updates an abstraction of the past play.

The simplest strategies are those that are *memoryless*, meaning they base their decisions solely on the current arena vertex. Games on finite arenas where memoryless strategies are sufficient to win can usually be solved in $\text{NP} \cap \text{coNP}$ [28] and winning strategies effectively synthesised. This is true for *parity*, *discounted-payoff* [31], *mean-payoff* [11], and *total-payoff* [8, 16] objectives. Even beyond finite graphs, memoryless strategies may suffice in more general contexts, such as for parity objectives over arenas of arbitrary cardinality [12, 33], or *discounted-payoff objectives* over finitely branching arenas [26, Corollary 2.1].¹ For concurrent (stochastic) *reachability* games on finite arenas, memoryless strategies also suffice [2, 21].

Generally more powerful than memoryless strategies are *finite-memory* strategies, which refer to strategies that can be implemented with a finite-state (Mealy) machine. A canonical class of languages over infinite words, and standard for defining objectives in games, are the ω -regular languages [30, 17]. One of the celebrated related results about reactive synthesis is the *finite-memory determinacy* of ω -regular games [6, 30, 18], which means that if there is a winning strategy in a game on a finite arena and with an ω -regular objective, there is one that can be implemented with a simple finite-state machine (whose size can be bounded). This implies that games with ω -regular objectives can be solved and that strategies can be synthesised, since it bounds the search space for winning strategies. Remarkably, the existence of winning finite-memory strategies for ω -regular games even holds over arbitrary infinite arenas [33]. When finite-memory strategies are sufficient, one of the main questions is usually to *minimise* their size, i.e., to find winning strategies with as few memory states as possible [10, 7, 9, 5, 4].

¹ Thus we consider the strategy complexity in discounted-payoff games as settled for the setting we consider. On infinitely branching arenas, step-counter strategies are insufficient (see Figure 1a).

Already very simple games require infinite memory to win. This especially holds for quantitative objectives, which ask that the aggregate of individual edge weights along a play exceeds some threshold. For instance, consider a game where the environment picks a number and then the controller has to pick a larger one (see Figure 1a). In order to win, Player 1 has to remember the (per se unbounded) initial challenge and no finite memory structure would be sufficient to do so. This objective is not ω -regular since it is built upon an infinite alphabet. We seek to understand for different classes of games, what kind of infinite-memory structures are sufficient for winning strategies.

A natural, arguably the simplest, type of infinite memory structure is a *step counter*: it only remembers how many steps have elapsed since the start of the game. The availability of such a counter is a reasonable assumption for practical applications, as most embedded devices have access to the current time, which suffices when each step takes a fixed amount of time. A *step-counter strategy* is one that, in addition to the current arena vertex, has access to the number of steps elapsed. Notice that in the game in Figure 1a, a step counter does not provide any relevant information (every path to vertex t has length one). Therefore, step-counter strategies do not suffice for Player 1. An important ingredient for these counterexamples is that the underlying arena is infinitely branching (and uses arbitrary weights). For many classes of games on *finitely* branching arenas, strategies based on a step counter and additional finite memory are close to being the simplest kinds of strategies sufficient to win. Examples are especially prevalent in stochastic games. For instance, in the “Big Match” (a concurrent mean-payoff game on a finite arena), neither a step counter nor finite memory is sufficient to play ε -optimally, yet a step counter *together with* one bit is [19]. The same is true for the “Bad Match”, which can be presented as a Büchi (repeated reachability) game [23, 32, 22]. This upper bound holds generally for concurrent Büchi games on finite arenas [22].

Quantitative objectives. Objectives based on numerical weights are commonly called *quantitative objectives*. These are defined using *quantitative payoff functions*, which combine any finite sequence of weights into an aggregate number. The three more common ones are the discounted-payoff [31], mean-payoff [15, 11], and total-payoff functions [14, 8]. Every payoff function induces four variants of objectives, depending on whether we consider the lim sup or lim inf, and on whether we ask that the limit is larger or strictly larger than a threshold. For total payoff, it is also relevant to distinguish the use of real values or ∞ as a threshold. We give an example to describe informally how we denote such objectives: $\overline{\text{MP}}_{\geq 0}$ refers to the set of infinite sequences of rational numbers that achieve a value ≥ 0 for the lim sup variant (the line is above MP) of the mean-payoff function (specified by letters MP). Over infinite arenas, the four variants are not equivalent and infinite-memory strategies are needed for at least one of the players (see [29, Example 8.10.2] and [27]).

To study the strategy complexity for different quantitative objectives, we classify them according to which level of the *Borel hierarchy* they belong to (which also ensures that the games we consider are determined [24]). In the first level of the hierarchy lie the *open* and *closed* objectives (i.e., the sets respectively in Σ_1^0 and Π_1^0), for which there exist recent characterisations of the sufficient memory structures over finite or infinite arenas [9, 5]. We build on this to establish upper bounds for more complex objectives. All variants of mean-payoff and total-payoff objectives are on the second or third level of the Borel hierarchy. Ohlmann and Skrzypczak [27] study objectives through their topological properties and provide a characterisation of the *prefix-independent* Σ_2^0 objectives for which memoryless strategies suffice for Player 1 over arbitrary arenas. They show in particular that memoryless strategies suffice for Player 1 for the quantitative objectives $\overline{\text{MP}}_{>0}$ and $\overline{\text{TP}}_{>-\infty}$, even over

infinitely branching arenas. Over stochastic games, quantitative (in particular lim inf mean-payoff) objectives on infinite arenas generally do not have (ε -)optimal strategies based on a step counter, even for finitely branching Markov decision processes [25].

Our contributions. We settle the strategy complexity over infinite, deterministic games for the mean-payoff and total-payoff objectives up to the second level of the Borel hierarchy. In particular, we show for which of these, step-counter strategies are sufficient for Player 1. Our upper bounds all allow for arenas with arbitrary weights, while our strongest lower bounds only use weights -1 , 0 , and 1 . Our results are as follows and summarised in Table 1.

- For $\overline{\text{TP}}_{>0}$ and $\overline{\text{TP}}_{\geq 0}$, strategies using a step counter and an arbitrary amount of finite memory do not suffice, even over acyclic finitely branching arenas (Theorem 10, Section 3). The proof rules out finite-memory structures using an application of the *infinite Ramsey theorem* to allow Player 2 to stay winning in a particular infinite arena regardless of the finite-memory structure of Player 1.
- In Section 5, we provide a sufficient condition for when step-counter strategies suffice over finitely branching arenas for prefix-independent objectives in $\mathbf{\Pi}_2^0$, i.e. countable intersections of open sub-objectives (Theorem 16). This implies in particular that step-counter strategies do suffice for $\overline{\text{MP}}_{\geq 0}$ and $\overline{\text{TP}}_{=+\infty}$ (Corollary 17), which is tight in the sense that finite-memory strategies do not suffice for these objectives, even over acyclic finitely branching arenas (Lemma 4). The proof uses carefully constructed expanding “bubbles”, so that within each consecutive bubble, Player 1 can satisfy the next open sub-objective. The step counter is used to determine the current bubble.
- In Section 6, we show that for $\overline{\text{TP}}_{\geq 0}$, which is not prefix-independent, strategies using a step-counter and one additional bit of memory suffice (Theorem 20). This is tight in that neither finite-memory strategies nor step-counter strategies suffice, even over acyclic finitely branching arenas (Lemmas 4 and 5). The proof similarly employs bubbles, but an additional bit is needed to keep track of whether a “sub-objective” has been achieved in the current bubble and then switches to stay in the winning region.

Structure. We define the various notions used throughout the paper in Section 2. Section 3 is dedicated to all lower bounds on the strategy complexity of the various objectives, culminating in a lower bound for $\overline{\text{TP}}_{>0}$. Section 4 is devoted to recalling useful results on open and closed objectives, upon which the following sections build. Section 5 proves a sufficient condition for the sufficiency of step-counter strategies for prefix-independent $\mathbf{\Pi}_2^0$ objectives. Section 6 proves an upper bound on the strategy complexity of $\overline{\text{TP}}_{\geq 0}$.

Due to space constraints, some proofs are omitted from this conference version. Complete details for all proofs can be found in the extended version [3].

2 Preliminaries

Given a set X , we write X^* for the set of finite words on X , X^+ for the set of non-empty finite words on X , and X^ω for the set of infinite words on X . For $w \in X^*$, we write $|w|$ for the length of w . For $w \in X^\omega$ and $j \in \mathbb{N}$, we write $w_{\leq j}$ for the finite prefix of length j of w .

Games. We study two-player zero-sum *games*, each given by an *arena* and an *objective*, as defined below. We refer to the two opposing players as Player 1 and Player 2.

■ **Table 1** Results for quantitative objectives up to the second level of the Borel hierarchy for finitely branching arenas. *SC* refers to *step counter*, and *FM* refers to *finite memory*.

Obj.	Description	Class	Strategy complexity
$\underline{\text{MP}}_{>0}$	$\bigcup_{m \geq 1} \bigcup_{i \geq 1} \bigcap_{j \geq i} \{w \mid \text{MP}(w_{\leq j}) \geq \frac{1}{m}\}$	Σ_2^0	Memoryless (even over infinitely branching arenas) [27]
$\underline{\text{TP}}_{>-\infty}$	$\bigcup_{m \geq 1} \bigcup_{i \geq 1} \bigcap_{j \geq i} \{w \mid \text{TP}(w_{\leq j}) \geq -m\}$	Σ_2^0	
$\underline{\text{TP}}_{>0}$	$\bigcup_{m \geq 1} \bigcup_{i \geq 1} \bigcap_{j \geq i} \{w \mid \text{TP}(w_{\leq j}) \geq \frac{1}{m}\}$	Σ_2^0	SC + FM insufficient (Theorem 10)
$\overline{\text{MP}}_{\geq 0}$	$\bigcap_{m \geq 1} \bigcap_{i \geq 1} \bigcup_{j \geq i} \{w \mid \text{MP}(w_{\leq j}) \geq \frac{-1}{m}\}$	Π_2^0	SC sufficient (Corollary 17)
$\overline{\text{TP}}_{=+\infty}$	$\bigcap_{m \geq 1} \bigcap_{i \geq 1} \bigcup_{j \geq i} \{w \mid \text{TP}(w_{\leq j}) \geq m\}$	Π_2^0	FM insufficient (Lemma 4)
$\overline{\text{TP}}_{\geq 0}$	$\bigcap_{m \geq 1} \bigcap_{i \geq 1} \bigcup_{j \geq i} \{w \mid \text{TP}(w_{\leq j}) \geq \frac{-1}{m}\}$	Π_2^0	SC + 1-bit sufficient (Theorem 20) FM insufficient (Lemma 4) SC insufficient (Lemma 5)

An *arena* is a directed graph with two kinds of vertices where edges are labelled by an element of C , a non-empty set of *colours*. Formally, an arena is a tuple $\mathcal{A} = (V, V_1, V_2, E)$ where $V = V_1 \cup V_2$ is a non-empty set of *vertices*, V_1 and V_2 are disjoint, and $E \subseteq V \times C \times V$ is a set of labelled *edges*. Vertices in V_1 and V_2 are respectively controlled by Player 1 and Player 2, which will appear clearly when we define strategies below. We require that for every vertex $v \in V$, there is an edge $(v, c, v') \in E$ (arenas are “non-blocking”). For $e = (v, c, v')$, we write $\text{from}(e)$ for v , $\text{col}(e)$ for c , and $\text{to}(e)$ for v' . An arena is *finite* if V is finite, and *finitely branching* if for every $v \in V$, the set $\{e \in E \mid \text{from}(e) = v\}$ is finite.

A *history* is a finite sequence $h = e_1 \dots e_n \in E^*$ of edges such that for $i \in \{1, \dots, n-1\}$, $\text{to}(e_i) = \text{from}(e_{i+1})$. We write $\text{from}(h)$ for $\text{from}(e_1)$, $\text{to}(h)$ for $\text{to}(e_n)$, and $\text{col}(h)$ for the sequence $\text{col}(e_1) \dots \text{col}(e_n) \in C^*$. For convenience, we assume that for every vertex v , there is a distinct *empty history* λ_v such that $\text{from}(\lambda_v) = \text{to}(\lambda_v) = v$. The set of histories of \mathcal{A} is denoted as $\text{hists}(\mathcal{A})$. For $p \in \{1, 2\}$, we write $\text{hists}_p(\mathcal{A})$ for the set of histories h such that $\text{to}(h) \in V_p$. A *play* is an infinite sequence of edges $\rho = e_1 e_2 \dots \in E^\omega$ such that for $i \geq 1$, $\text{to}(e_i) = \text{from}(e_{i+1})$. We write $\text{from}(\rho)$ for $\text{from}(e_1)$ and $\text{col}(\rho)$ for $\text{col}(e_1) \text{col}(e_2) \dots \in C^\omega$. A history h (resp. a play ρ) is said to be *from* v if $v = \text{from}(h)$ (resp. $v = \text{from}(\rho)$).

An *objective* (sometimes called a *winning condition* in the literature) is a set $O \subseteq C^\omega$. An objective O is *prefix-independent* if for all $w \in C^*$, $w' \in C^\omega$, $ww' \in O$ if and only if $w' \in O$.

Strategies. A *strategy of Player p on \mathcal{A}* is a function $\sigma: \text{hists}_p(\mathcal{A}) \rightarrow E$ such that for all $h \in \text{hists}_p(\mathcal{A})$, $\text{from}(\sigma(h)) = \text{to}(h)$. A play $\rho = e_1 e_2 \dots$ is *consistent with a strategy σ of Player p* if for all finite prefixes h of ρ such that $\text{to}(h) \in V_p$, $\sigma(h) = e_{|h|+1}$. A strategy σ of Player 1 is *winning for objective O from a vertex v* if all plays from v consistent with σ induce a sequence of colours in O . For a fixed objective, the set of vertices of an arena \mathcal{A} from which a winning strategy for Player 1 exists is called the *winning region of Player 1 on \mathcal{A}* and is denoted $W_{\mathcal{A},1}$. A strategy σ of Player 1 is *uniformly winning for objective O in \mathcal{A}* if σ is winning from every vertex of the winning region of \mathcal{A} .

A *memory structure for an arena $\mathcal{A} = (V, V_1, V_2, E)$* is a tuple $\mathcal{M} = (M, m_0, \delta)$ where M is a set of *memory states*, $m_0 \in M$ is an *initial state*, and $\delta: M \times E \rightarrow M$ is a *memory update function*. We extend δ to a function $\delta^*: M \times E^* \rightarrow M$ in a natural way. A memory structure \mathcal{M} is *finite* if M is finite. A strategy σ of Player p on \mathcal{A} is *based on \mathcal{M}* if there exists a function $f: V_p \times M \rightarrow E$ such that, for all $h \in \text{hists}_p(\mathcal{A})$, $\sigma(h) = f(\text{to}(h), \delta^*(m_0, h))$. We will abusively assume that a strategy based on a memory structure is this function f .

13:6 The Power of Counting Steps in Quantitative Games

A *memoryless strategy* is a strategy based on a memory structure with a single memory state. A *1-bit strategy* is a strategy based on a memory structure with two memory states. A *step counter* is a memory structure $\mathcal{S} = (\mathbb{N}, 0, (s, e) \mapsto s + 1)$ that simply counts the number of steps already elapsed in a game. A strategy σ of Player p on \mathcal{A} is a *step-counter strategy* if σ is based on a step counter; in other words, if there is a function $f: V_p \times \mathbb{N} \rightarrow E$ such that $\sigma(h) = f(\text{to}(h), |h|)$. This means that σ only considers the current vertex and the number of steps elapsed to make its decisions. Step-counter strategies are sometimes called “Markov strategies” [32, 20].

A *step-counter and finite-memory structure* is a memory structure with state space $M = \mathbb{N} \times \{0, \dots, K - 1\}$, initial state $(0, 0)$, and a transition function δ such that $\delta((s, m), e) = (s + 1, \delta'((s, m), e))$ for some function $\delta': M \times E \rightarrow \{0, \dots, K - 1\}$. Notice that a step counter corresponds to the special case of a step-counter and finite-memory structure with $K = 1$. A *step-counter + 1-bit strategy* is a strategy based on a step-counter and finite-memory structure with $K = 2$.

We say that a kind of strategies *suffices for objective O over a class of arenas* if, for all arenas in this class, from all vertices of her winning region, Player 1 has a winning strategy of this kind. We say that a kind of strategies *suffices uniformly for objective O over a class of arenas* if, for all arenas in this class, Player 1 has a uniformly winning strategy of this kind.

For an arena $\mathcal{A} = (V, V_1, V_2, E)$ and a memory structure $\mathcal{M} = (M, m_0, \delta)$, we write $\mathcal{A} \otimes \mathcal{M}$ for the *product between \mathcal{A} and \mathcal{M}* . It is the arena (V', V'_1, V'_2, E') such that $V' = V \times M$, $V'_1 = V_1 \times M$, $V'_2 = V_2 \times M$, and $E' = \{((v, m), c, (v', \delta(m, e))) \mid e = (v, c, v') \in E, m \in M\}$. Observe that Player 1 has a winning strategy based on \mathcal{M} from a vertex v in an arena \mathcal{A} if and only if Player 1 has a winning memoryless strategy from vertex (v, m_0) in $\mathcal{A} \otimes \mathcal{M}$.

To simplify reasonings over specific arenas, we show that step counters do not have any use when the arena already *encodes the step count*.

► **Lemma 1.** *Let $\mathcal{A} = (V, V_1, V_2, E)$ be an arena, and $v_0 \in V$ be an initial vertex. Assume that for each pair of histories h_1, h_2 from v_0 to some $v \in V$, we have $|h_1| = |h_2|$ (i.e., the arena already “encodes the step count from v_0 ”). Then, a step-counter and finite-memory strategy with K states of finite memory can be simulated from v_0 by a strategy with only K states of finite memory.*

Proof. By hypothesis on \mathcal{A} , there exists $n_v \in \mathbb{N}$ the length of any history from v_0 to v . Let $\sigma': V_1 \times \mathbb{N} \times M \rightarrow E$ be a step-counter and finite-memory strategy with $M = \{0, \dots, K - 1\}$, with finite-memory update function $\delta': M \times E \rightarrow \{0, \dots, K - 1\}$. Let $\mathcal{M} = (M, 0, \delta)$ be the memory structure with $\delta(m, e) = \delta'((n_{\text{from}(e)}, m), e)$. By construction, the strategy $\sigma: V_1 \times M \rightarrow E$ such that $\sigma(v, m) = \sigma'(v, n_v, m)$ behaves exactly like σ' from v_0 . ◀

Quantitative objectives. We consider classical quantitative objectives: mean-payoff and total-payoff objectives, as defined below. Let $C \subseteq \mathbb{Q}$ (when colours are rational numbers, we often refer to them as *weights*). For a finite word $w = c_1 \dots c_{|w|} \in C^*$, define $\text{TP}(w) = \sum_{i=1}^{|w|} c_i$ for the *total payoff* of the word, i.e., the sum of the weights it contains. Further, when $|w| \geq 1$, let $\text{MP}(w) = \text{TP}(w)/|w|$ denote the *mean payoff* of the word w , i.e., the mean of the weights it contains. We extend any such aggregate function $X: C^* \rightarrow \mathbb{R}$ to infinite words by taking limits: for $w \in C^\omega$, we define $\overline{X}(w) = \limsup_j X(w_{\leq j})$ and $\underline{X}(w) = \liminf_j X(w_{\leq j})$. Fixing a binary relation $\triangleright \subseteq \mathbb{R}^2$ and threshold $r \in \mathbb{Q} \cup \{-\infty, \infty\}$, this naturally defines objectives $\overline{X}_{\triangleright r} = \{w \in C^\omega \mid \overline{X}(w) \triangleright r\}$ and $\underline{X}_{\triangleright r} = \{w \in C^\omega \mid \underline{X}(w) \triangleright r\}$.

In particular, we are interested in the limit infimum/supremum objectives for total and mean payoff.² We consider the mean-payoff variants with threshold $r \in \mathbb{Q}$, and the total-payoff variants with $r \in \mathbb{Q} \cup \{-\infty, +\infty\}$. Note that all four mean-payoff objectives and all four total-payoff objectives with ∞ threshold are prefix-independent, but the four total-payoff objectives with threshold in \mathbb{Q} are not prefix-independent.

► **Remark 2.** Our results are generally stated for threshold $r = 0$. This is without loss of generality since the results deal with large classes of arenas, and little modifications to the arenas allow to reduce from an arbitrary rational threshold to threshold 0. ◻

Topology of objectives. For $w \in C^*$, we write $wC^\omega = \{ww' \mid w' \in C^\omega\}$ for the objective containing all infinite words that start with w (it is sometimes called the *cylinder* or *cone* of w). An objective O is *open* if there is a set $A \subseteq C^*$ such that $O = \bigcup_{w \in A} wC^\omega$. For an open objective O , we say that a finite word $w \in C^*$ *already satisfies* O if $wC^\omega \subseteq O$. If an objective is open, then by definition, any infinite word it contains has a finite prefix that already satisfies it. An objective is *closed* if it is the complement of an open set.

Open and closed objectives are at the first level of the *Borel hierarchy*; the set of open (resp. closed) objectives is denoted Σ_1^0 (resp. Π_1^0). For $i > 1$, we can define Σ_i^0 as all the countable unions of sets in Π_{i-1}^0 , and Π_i^0 as all the countable intersections of sets in Σ_{i-1}^0 . All the objectives considered in this paper lie in the first three levels of this hierarchy, and we focus on those on the second level.

3 Lower bounds

We provide lower bounds on the size/structure of the memory to build winning strategies, focusing on objectives $\overline{\text{MP}}_{\geq 0}$, $\overline{\text{TP}}_{=+\infty}$, $\overline{\text{TP}}_{\geq 0}$, and $\underline{\text{TP}}_{>0}$, which are the four objectives on the second level of Borel hierarchy for which we want to establish whether step-counters strategies suffice. We mention where our constructions directly work for further objectives.

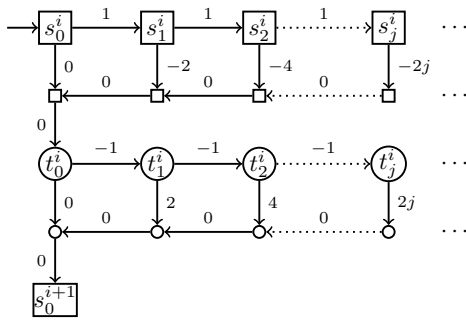
All lower bounds are based on the simple idea that one player chooses some number and the other must match it. We first observe that on infinitely branching arenas with arbitrary weights, neither finite memory nor a step counter, nor both together, is sufficient. The proof uses the arenas from Figure 1, discussed informally in Section 1 (the missing proofs in this section are available in [3, Appendix A]).

► **Lemma 3.** *Over infinitely branching arenas with arbitrary weights, step-counter and finite-memory strategies are not sufficient for Player 1 for objectives $\overline{\text{MP}}_{>0}$, $\overline{\text{MP}}_{\geq 0}$, $\overline{\text{TP}}_{=+\infty}$, $\underline{\text{TP}}_{>0}$, $\underline{\text{TP}}_{\geq 0}$, $\overline{\text{TP}}_{>0}$ and $\overline{\text{TP}}_{\geq 0}$.*

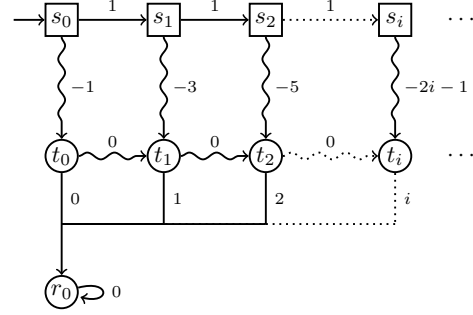
We now establish lower bounds over finitely branching arenas. Firstly, the example \mathcal{A}'_1 can be made finitely branching and acyclic, as depicted in Figure 2. The resulting arena, \mathcal{A}_2 , simply unfolds \mathcal{A}'_1 so that any edge $(s, -j, t)$ is replaced by a finite path $s_0^i \rightarrow \dots \rightarrow s_j^i \rightarrow t_0^i$, and similarly for the responses. This construction works as long as one can discourage (i.e., make losing) the choice to stay on the infinite intermediate chain of vertices and not moving on to a vertex controlled by the opponent. Here, this is achieved by using weights 1 on the chains of Player 2 and weights -1 on the chains of Player 1, which are then compensated by weights twice as large. In practice, edges with weights $i \in \mathbb{N}$ (resp. $-i \in -\mathbb{N}$) can be

² We only consider objectives where the threshold is a *lower* bound ($\triangleright \in \{>, \geq\}$); each variant with *upper* bound behaves like a variant with lower bound when we replace each weight c in arenas with its additive inverse $-c$ and switch the sup/inf (for instance, $\overline{\text{MP}}_{<r}$ behaves like $\underline{\text{MP}}_{>r}$ when we invert the weights).

13:8 The Power of Counting Steps in Quantitative Games



■ **Figure 2** The arena \mathcal{A}_2 is acyclic and every vertex has finite in- and out-degree. We recall that circles are controlled by Player 1 and squares by Player 2.



■ **Figure 3** The arena \mathcal{A}_3 . Arrows $s_i \xrightarrow{-2i-1} t_i$ are shorthand for paths of length $2i+1$ with edge weights -1 , and $t_i \xrightarrow{0} t_{i+1}$ are shorthand for paths of length 3 with edge weights 0.

replaced by chains of i weights 1 (resp. i weights -1). This allows to obtain lower bounds on the lim sup objectives. The fact that finite-memory strategies are insufficient for variants of the mean-payoff objectives over finitely branching arenas was already discussed in [29, Example 8.10.2] and [27]; we rephrase it here for completeness.

► **Lemma 4.** *Over finitely branching arenas, finite-memory strategies are not sufficient for Player 1 for objectives $\overline{\text{MP}}_{>0}$, $\overline{\text{MP}}_{\geq 0}$, $\overline{\text{TP}}_{=+\infty}$, $\overline{\text{TP}}_{>0}$, and $\overline{\text{TP}}_{\geq 0}$.*

Notice that although finite memory is insufficient for Player 1 in \mathcal{A}_2 , a step counter allows her to deduce an upper bound on the previous choice of Player 2 and is therefore sufficient. Indeed, since \mathcal{A}_2 is finitely branching and every round starts in a unique initial vertex for that round, Player 1 can (over) estimate that all steps of the history so far were spent by her opponent's choice (steps between s_0^i up to some s_j^i and then leading directly to t_0^{i+1}).

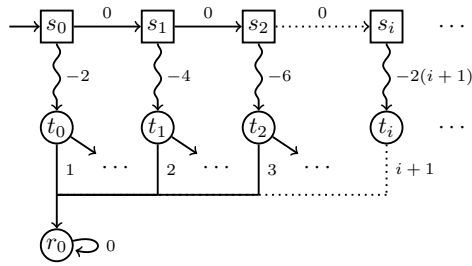
In order to construct an arena in which no step-counter strategy is sufficient, we obfuscate possible histories leading to Player 1's choices by making them the same length (see Figure 3).

► **Lemma 5.** *Consider the arena \mathcal{A}_3 depicted in Figure 3. Player 1 has a winning strategy, but no winning step-counter strategy for objectives $\underline{\text{TP}}_{>0}$, $\underline{\text{TP}}_{\geq 0}$, $\overline{\text{TP}}_{>0}$, and $\overline{\text{TP}}_{\geq 0}$. Hence, over finitely branching arenas, step-counter strategies are not sufficient for Player 1 for objectives $\underline{\text{TP}}_{>0}$, $\underline{\text{TP}}_{\geq 0}$, $\overline{\text{TP}}_{>0}$, and $\overline{\text{TP}}_{\geq 0}$.*

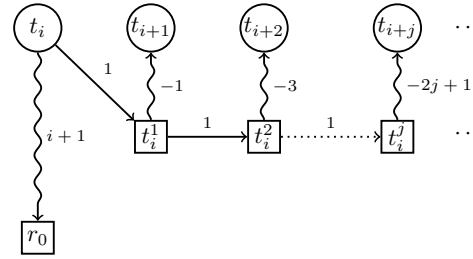
Proof. Player 1 only makes relevant choices at vertices t_i , and the choice is whether to *delay* (move to t_{i+1}) or *exit* (move to r_0). A winning (finite-memory) strategy for all mentioned objectives is to delay twice and then exit. Indeed, any history leading to t_i has total payoff of at least $-i-1$. By delaying twice and then exiting, Player 1 guarantees that the sink vertex r_0 is reached and the total payoff collected on the way is at least 1.

Conversely, any strategy σ of Player 1 that is based solely on a step counter cannot distinguish histories leading to the same vertex t_i . Let us assume that σ does not choose to avoid r_0 indefinitely, as doing so would result in a negative total payoff, which is losing for her. Then there is at least one vertex t_i from which the strategy exits. Player 2 can exploit this by going there via s_i . The resulting play has a negative total payoff. ◀

We now extend the previous examples to show that even access to both a step counter and finite memory is not sufficient for Player 1. The construction below is stated for the total-payoff objective $\underline{\text{TP}}_{\geq 0}$, and also works for $\underline{\text{TP}}_{>0}$. The main idea is to require Player 1 to delay going to r_0 more than a constant number of times, as dictated by Player 2's initial move.



■ **Figure 4** The arena \mathcal{A}_4 . Arrows $s_i \xrightarrow{-2(i+1)} t_i$ are shorthand for paths of length $2i+3$ with total payoff $-2(i+1)$. From a vertex t_i , Player 1 either exits to r_0 or moves to the gadget in Figure 5.



■ **Figure 5** The delay gadget from vertex t_i in arena \mathcal{A}_4 . The arrows from t_i^j to t_{i+j} are shorthand for paths of length $2j$ and payoff $-2j+1$.

► **Definition 6.** Let \mathcal{A}_4 be the arena from Figure 4. It has a similar high-level structure to \mathcal{A}_3 with different weights, and with more complex gadgets (Figure 5) between vertices t_i . At each vertex t_i , Player 1 decides between two actions:

1. to exit to r_0 and gain payoff $i+1$ by doing so, or
2. to delay to some vertex t_{i+j} where $j > 0$ is chosen by Player 2, and gain payoff $-j+1$.

Notice that, after Player 2 moved down from vertex s_k , Player 1 can (only) win by delaying at least $k+1$ times (which we show in Lemma 8). We will show that the gadgets allow Player 2 to confuse any strategy of Player 1 that is only based on a step counter and finite memory. Without them, the current vertex t_i together with finite extra memory would allow Player 1 to approximate how many delays she has chosen so far and therefore allow her to win with a finite-memory strategy.³

A simple counting argument shows that all paths from s_0 to a vertex t_k have the same length (proof in [3, Appendix A]). By Lemma 1, it implies that a step counter is useless in \mathcal{A}_4 .

► **Lemma 7.** For every t_k in arena \mathcal{A}_4 , all paths from s_0 to t_k have the same length.

The following lemma will be used to argue that Player 1 wins, albeit with infinite memory.

► **Lemma 8.** From a vertex t_i , if Player 2 does not stay forever in a gadget, the strategy σ_k of Player 1 that enters the delay gadget exactly $k \in \mathbb{N}$ times achieves a total payoff of exactly $i+k+1$ in r_0 .

Proof. Assume that Player 2 never stays forever in a gadget (which would be winning for Player 1 for all quantitative objectives considered). The total payoff on the path from t_i to the next vertex t_{i+j} is $-j+1$. Suppose Player 1 delays k times and let $j(1), j(2), \dots, j(k)$ be the lengths of the intermediate paths through gadgets, as chosen by Player 2. That is, the play ends up in vertex t_{i+l} for $l = \sum_{c=1}^k j(c)$ and has gained payoff $\sum_{c=1}^k ((-j(c)+1)) = -l+k$. After k delays, exiting to r_0 from vertex t_{i+l} gives an immediate payoff of $i+l+1$. The total payoff from t_i to r_0 is thus $(-l+k) + (i+l+1) = i+k+1$. ◀

³ The idea would be to partition t_i 's into (growing) intervals, so that each interval is picked so large that it is safe to exit from any vertex after the interval if the play entered a vertex before or at the start of that interval. A winning strategy is then to keep on delaying to t_i 's until vertices in three different intervals have been seen, and then exit. This requires 3 memory states to remember the interval changes.

13:10 The Power of Counting Steps in Quantitative Games

► **Lemma 9.** *Consider the game played on \mathcal{A}_4 . Then, from vertex s_0 ,*

1. *Player 1 wins for objective $\underline{\text{TP}}_{>0}$;*
2. *every step-counter and finite-memory strategy of Player 1 is losing for $\underline{\text{TP}}_{\geq 0}$.*

Proof. For point (1), let σ be the Player 1 strategy that, upon observing history $s_0 \xrightarrow{*} s_k \rightarrow t_k$, switches to the finite-memory strategy σ_{k+1} from the previous lemma (delay $k+1$ times and then exit). Consider any play consistent with this strategy σ . Either Player 2 never moves to a vertex t_k , and then the total payoff is 0, which is winning for Player 1 for $\underline{\text{TP}}_{\geq 0}$. Otherwise, a vertex t_k is reached (and accordingly, the payoff until reaching it is $-2(k+1)$). Using σ_{k+1} , Player 1 guarantees a lim inf total payoff of at least 0 on any continuation: either Player 2 never leaves some gadget and the total payoff is $+\infty$, or Player 1 exits to r_0 after $k+1$ delays, which adds $k + (k+1) + 1 = 2(k+1)$ to the total payoff by Lemma 8. In this second case, the total payoff is therefore $-2(k+1) + 2(k+1) = 0$.

For point (2), by Lemmas 1 and 7, it suffices to show that every finite-memory strategy of Player 1 is losing. Consider now any such strategy σ_1 of Player 1 with memory of size $K \in \mathbb{N}$ and memory update function δ . We will show that there exists a strategy σ_2 for Player 2 that is winning against σ_1 . Player 2's strategy is determined by 1) the initial choice of t_j it visits and 2) which vertex t_{i+j} to select in the gadgets (Figure 5) when Player 1 delays from vertex t_i . We show the existence of suitable choices by employing an argument based on the infinite Ramsey theorem, as follows.

First, δ defines naturally, for any history $h \in E^*$, a function $\delta_h: M \rightarrow M$ that specifies how the memory is updated when observing this history (formally, $\delta_h(m) = \delta^*(m, h)$). Further, for every $i \geq 0$ there is a function $f_i: M \rightarrow \{0, 1\}$ that describes for which memory states the strategy σ_1 chooses to delay or exit from t_i (formally, $f_i(m)$ equals 1 if $\sigma_1(t_i, m) = (t_i, i+1, r_0)$, and 0 otherwise). Since $|M| = K \in \mathbb{N}$, there are only finitely many distinct such functions f_i and δ_h . Consider now the edge-labelled graph G consisting of all vertices $t_i, i \geq 0$, and where for any two $i, j \in \mathbb{N}$, the edge between t_i and t_{i+j} is labelled by the pair (f_i, δ_h) where $h = t_i \rightarrow t_i^1 \rightarrow \dots \rightarrow t_i^j \rightarrow t_{i+j}$ is the history through the delay gadget in \mathcal{A}_4 .

Recall the infinite Ramsey theorem: If one labels all edges of the complete (undirected and countably infinite) graph with finitely many colours, then there exists an infinite monochromatic subgraph. Applying this to our graph G yields an infinite subgraph, say with vertices $t_{\ell(i)}$ identified by $\ell: \mathbb{N} \rightarrow \mathbb{N}$, where all edges have the same label. W.l.o.g., assume that $\ell(0) \geq K$ and $\ell(i+1) > \ell(i) + 1$ for all $i \geq 0$. Based on this, the strategy σ_2 of Player 2 will 1) initially move to $t_{\ell(0)}$ and 2) whenever Player 1 chooses to delay from $t_{\ell(i)}$ then Player 2 moves to vertex $t_{\ell(i+1)}$. Now consider the play ρ consistent with both strategies σ_1 and σ_2 . There are two cases. Either along this play Player 1 chooses to exit from some vertex $t_{\ell(j)}, j < K$, or not. If she exits too early (after delaying only $j < K$ times), then the total payoff after exiting is exactly $-2(\ell(0) + 1) + (\ell(0) + j + 1) = -\ell(0) + j - 1$ by Lemma 8, which is < 0 as $\ell(0) \geq K > j$. Hence, the play is won by Player 2. Alternatively, if along the play, Player 1 delays at least K times then, by the pigeonhole principle, there is at least one memory mode that she revisits. More precisely, the play visits vertices $t_{\ell(i)}$ and $t_{\ell(j)}, i < j \leq K$ in the same memory mode. Recall that the functions $f_{\ell(i)}$ are all identical for $i \geq 0$. It follows that the play will continue visiting vertices $t_{\ell(k)}, k \geq 0$ only and never exit to r_0 . Finally, observe that in any delay gadget from a vertex $t_{\ell(i)}$, the path to vertex $t_{\ell(i+1)}$ has total payoff of $1 - (\ell(i+1) - \ell(i))$. Consequently, the infinite play ρ that visits all $t_{\ell(i)}$ will be such that $\underline{\text{TP}}(\rho) = -\infty$ and is losing for Player 1 for $\underline{\text{TP}}_{\geq 0}$. ◀

► **Theorem 10.** *Strategies based on a step counter and finite memory are not sufficient for Player 1 in games with finitely branching arenas and objectives $\underline{\text{TP}}_{\geq 0}$ or $\underline{\text{TP}}_{>0}$.*

Proof. For $\underline{\text{TP}}_{\geq 0}$ this follows directly from Lemma 9. For $\underline{\text{TP}}_{> 0}$, just extend the arena by a new initial vertex s_{-1} with sole outgoing edge $s_{-1} \xrightarrow{1} s_0$ to ensure that the play in which Player 2 never moves to a vertex t_i is won by Player 1. ◀

4 Open objectives

The quantitative objectives defined in Section 2 all belong to the second or third level of the Borel hierarchy, and the strategy complexity of such objectives is not yet well understood. However, they use as building blocks objectives from the first level of the Borel hierarchy (i.e., open and closed objectives), for which there already exist characterisations of memory requirements. We recall some of these results for the memory structures that we study.

Step-monotonicity. Let $O \subseteq C^\omega$ be an objective. For two finite words $w_1, w_2 \in C^*$, we write $w_1 \preceq_O w_2$ if for all $w \in C^\omega$, $w_1 w \in O$ implies $w_2 w \in O$ (meaning that the winning continuations of w_1 are included in those of w_2). The relation \preceq_O is a preorder and satisfies that for $w_1, w_2 \in C^*$ and $c \in C$, $w_1 \preceq_O w_2$ implies $w_1 c \preceq_O w_2 c$ (i.e., it is a “congruence”). We write $w_1 \prec_O w_2$ if $w_1 \preceq_O w_2$ but $w_2 \not\preceq_O w_1$. We say that two finite words $w_1, w_2 \in C^*$ are *comparable for \preceq_O* if $w_1 \preceq_O w_2$ or $w_2 \preceq_O w_1$. We extend preorder \preceq_O to histories: we write $h_1 \preceq_O h_2$ if $\text{col}(h_1) \preceq_O \text{col}(h_2)$.

We say that an objective O is *step-monotonic* if for any two finite words $w_1, w_2 \in C^*$ such that $|w_1| = |w_2|$, w_1 and w_2 are comparable for \preceq_O . In other words, for any two finite words that are read up to the same state of a step counter, one of the words must include at least the winning continuations of the other word. This is a specialisation of the *\mathcal{M} -strong-monotony* property [5] for the step-counter memory structure $\mathcal{M} = \mathcal{S}$.

► **Example 11.** Let $C = \{a, b\}$. The open objective $O = aaC^\omega \cup bbC^\omega$ is *not* step-monotonic, since for $w_1 = a$ and $w_2 = b$, we have that $|w_1| = |w_2|$, but w_1 and w_2 are not comparable for \preceq_O . Indeed, a^ω (resp. b^ω) is a winning continuation of w_1 but not w_2 (resp. w_2 but not w_1).

Now, let $C = \mathbb{Q}$ and $s \in \mathbb{N}$. The open objective $O_s = \{w \in C^\omega \mid \exists j \geq s, \text{TP}(w_{\leq j}) \geq 0\}$ (containing all infinite words whose total payoff goes over 0 at some point after s steps) is step-monotonic. Indeed, consider two finite words $w_1, w_2 \in C^*$ such that $|w_1| = |w_2|$. If w_2 already satisfies O_s (i.e., $w_2 C^\omega \subseteq O_s$), then necessarily, $w_1 \preceq_{O_s} w_2$. Similarly, if w_1 already satisfies O_s , then $w_2 \preceq_{O_s} w_1$. When neither w_1 nor w_2 already satisfies O_s , they can be compared by their current total payoff: if $\text{TP}(w_1) \leq \text{TP}(w_2)$, then $w_1 \preceq_{O_s} w_2$. ◻

► **Remark 12.** Variations of objective O_s are used as building blocks to define quantitative objectives (as can be seen in the descriptions in Table 1), and will be considered again later. An important remark is that \preceq_{O_s} is not completely determined by the current total payoff of words. For instance, if $w_1 = -1, 0$ and $w_2 = 0, -100$, we have $w_1 \prec_{O_1} w_2$ even though $\text{TP}(w_1) > \text{TP}(w_2)$. The reason is that w_2 *already satisfies O_1* after 1 step, and any continuation is therefore winning, despite the current total payoff being lower. ◻

Step-counter strategies for open objectives. In general, the step-monotonicity property is necessary for the uniform sufficiency of step-counter strategies over finitely branching arenas (this is a specialisation of [5, Lemma 5.2] to the step-counter memory structure \mathcal{S}). However, the results of [5] do not yield a characterisation for open objectives in full generality. For the special case of the step-counter memory structure, we can actually show a converse: for open objectives, step-monotonicity implies that step-counter strategies suffice over finitely branching arenas. This is what we show over the next three lemmas (the missing proofs in this section are available in [3, Appendix B]).

13:12 The Power of Counting Steps in Quantitative Games

First, a handy result about open objectives is that in a *finitely branching* arena, any winning strategy already satisfies the objective within a bounded number of steps.

► **Lemma 13.** *Let $O \subseteq C^\omega$ be an open objective, \mathcal{A} be a finitely branching arena, and v_0 be an initial vertex in \mathcal{A} . If a strategy σ is winning from v_0 for O , then there is $s \in \mathbb{N}$ such that all histories h of length $\geq s$ consistent with σ already satisfy O , i.e., $\text{col}(h)C^\omega \subseteq O$.*

Second, the following lemma shows that for step-monotonic objectives, step-counter strategies can be “locally not worse” than arbitrary strategies.

► **Lemma 14.** *Let $O \subseteq C^\omega$ be a step-monotonic objective. Let $\mathcal{A} = (V, V_1, V_2, E)$ be a finitely branching arena, $v_0 \in V$ be an initial vertex, and σ' be any strategy of Player 1 on \mathcal{A} . There is a step-counter strategy σ such that, for every history h from v_0 consistent with σ , there is a history h' from v_0 consistent with σ' such that $|h'| = |h|$, $\text{to}(h') = \text{to}(h)$, and $h' \preceq_O h$.*

The previous two lemmas imply that step-counter strategies suffice to win for open, step-monotonic objectives.

► **Corollary 15.** *Let $O \subseteq C^\omega$ be an open, step-monotonic objective. Step-counter strategies suffice for O over finitely branching arenas.*

Proof. Let \mathcal{A} be a finitely branching arena. Let v_0 be a vertex from the winning region and σ' be an arbitrary winning strategy from v_0 . By Lemma 13, using that O is open and \mathcal{A} is finitely branching, for all histories h of length $\geq s$ consistent with σ' , we have $\text{col}(h)C^\omega \subseteq O$.

As O is step-monotonic, let σ be the step-counter strategy provided by Lemma 14. Every history h of length s from v_0 consistent with σ is at least as good (for \preceq_O) as a history h' of length s from v_0 consistent with σ' . Since h' only has winning continuations, so does h . Therefore, strategy σ is winning from v_0 . ◀

5 Prefix-independent Π_2^0 objectives

In this section, we show that step-counter strategies suffice for Player 1 for objectives $\overline{\text{MP}}_{\geq 0}$ and $\overline{\text{TP}}_{=+\infty}$. In fact, we give a sufficient condition for when step-counter strategies suffice for Player 1 in finitely branching games where the objectives are prefix-independent and in Π_2^0 .

Recall that an objective is in Π_2^0 if it can be written as $\bigcap_{m \in \mathbb{N}} O_m$ for some open objectives O_m .

► **Theorem 16.** *Let $O = \bigcap_{m \in \mathbb{N}} O_m \subseteq C^\omega$ be a prefix-independent Π_2^0 objective such that the objectives O_m are open and step-monotonic. Then, step-counter strategies suffice uniformly for O over finitely branching arenas.*

Proof. Let $\mathcal{A} = (V, V_1, V_2, E)$ be a finitely branching arena, and let $v_0 \in V$ be an initial vertex. Let $W_{\mathcal{A},1} \subseteq V$ be the winning region of \mathcal{A} for O . We assume that v_0 is in the winning region $W_{\mathcal{A},1}$, and build a winning *step-counter* strategy from v_0 .

We build a winning step-counter strategy $\sigma: V_1 \times \mathbb{N} \rightarrow E$ from v_0 by induction on parameter m used in the definition of $O = \bigcap_{m \in \mathbb{N}} O_m$. We consider the product arena $\mathcal{A} \otimes \mathcal{S}$, and fix a strategy for increasingly high step values. The inductive scheme is as follows: for every $m \in \mathbb{N}$, we fix σ on $V_1 \times \{0, \dots, k_m - 1\}$ for some step bound $k_m \in \mathbb{N}$. We ensure that

- along all histories from v_0 consistent with σ of length at most k_m , the history does not leave $W_{\mathcal{A},1}$ (i.e., for all reachable (v, s) , we have $v \in W_{\mathcal{A},1}$), and
- the open objectives $O_{m'}$ for $m' \leq m$ are already satisfied within k_m steps (i.e., any history of length k_m consistent with σ only has winning continuations for $O_{m'}$).

For the base case, we may assume that we start the induction at $m = -1$ with $k_{-1} = 0$ and $O_{-1} = C^\omega$. We indeed have that from $(v_0, 0)$, the winning region is not left within $k_{-1} = 0$ step and that the open objective O_{-1} is already satisfied.

Now, assume that for some $m \geq 0$, the above properties hold, so we have already fixed the moves of σ in $\mathcal{A} \otimes \mathcal{S}$ on $V_1 \times \{0, \dots, k_m - 1\}$, yielding arena $(\mathcal{A} \otimes \mathcal{S})_m$. We first show that in arena $(\mathcal{A} \otimes \mathcal{S})_m$ the vertex $(v_0, 0)$ still belongs to the winning region. We have assumed by induction that the winning region $W_{\mathcal{A},1}$ is not left within k_m steps. This means that for all (v, k_m) reachable from $(v_0, 0)$ in $(\mathcal{A} \otimes \mathcal{S})_m$, v is in $W_{\mathcal{A},1}$. As O is prefix-independent, no matter the history from $(v_0, 0)$ to (v, k_m) , there is still a winning strategy from (v, k_m) (recall that no choice for Player 1 has been fixed beyond step k_m). Hence, no matter how Player 2 plays in the first k_m steps, there is still a way to win for O from $(v_0, 0)$.

We therefore take an (arbitrary) winning strategy σ'_{m+1} of Player 1 from $(v_0, 0)$ in $(\mathcal{A} \otimes \mathcal{S})_m$. As σ'_{m+1} is winning for $O = \bigcap_{m \in \mathbb{N}} O_m$, σ'_{m+1} wins in particular for the open O_{m+1} . Since the arena is finitely branching and O_{m+1} is open, applying Lemma 13, there is $k'_{m+1} \in \mathbb{N}$ such that for all histories h' of length $\geq k'_{m+1}$ consistent with σ'_{m+1} , h' already satisfies O_{m+1} (i.e., $\text{col}(h')C^\omega \subseteq O_{m+1}$). As O_{m+1} is step-monotonic, by Lemma 14, there is a step-counter strategy σ_{m+1} such that for every history h from $(v_0, 0)$ consistent with σ_{m+1} , there is a history h' from $(v_0, 0)$ consistent with σ'_{m+1} such that $|h'| = |h|$, $\text{to}(h') = \text{to}(h)$, and $h' \preceq_{O_{m+1}} h$.

To ensure that we fix at least one extra step of the strategy in the inductive step, let $k_{m+1} = \max\{k'_{m+1}, k_m + 1\}$. We extend the definition of σ to play the same moves as σ_{m+1} on $V_1 \times \{k_m, \dots, k_{m+1} - 1\}$, which also defines $(\mathcal{A} \otimes \mathcal{S})_{m+1}$. We prove the two items of the inductive scheme.

First, σ still does not leave $W_{\mathcal{A},1}$ up to step k_{m+1} : indeed, for every history consistent with σ_{m+1} , there is a history consistent with σ'_{m+1} reaching the same vertex. Since σ'_{m+1} is winning and O is prefix-independent, no such vertex can be outside of the winning region.

Second, strategy σ then guarantees O_{m+1} within k_{m+1} steps: after k_{m+1} steps, every history consistent with σ_{m+1} is at least as good for $\preceq_{O_{m+1}}$ as a history of length k_{m+1} of σ'_{m+1} . But every history h' of length k_{m+1} consistent with σ' is such that $\text{col}(h')C^\omega \subseteq O_{m+1}$, and therefore has only winning continuations.

This concludes the induction argument and shows the existence of a winning step-counter strategy from v_0 as we iterate this process for $m \rightarrow \infty$.

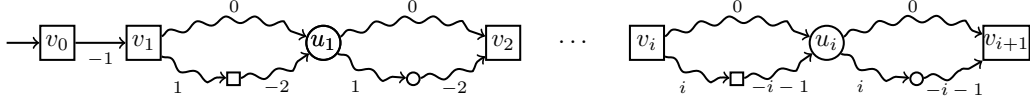
We now know that for any vertex from the winning region, there is a winning step-counter strategy. The existence of a *uniformly* winning step-counter strategy can be shown using prefix-independence of O ; this part of the proof is standard and is detailed in [3, Appendix C]. ◀

This theorem applies to $\overline{\text{MP}}_{\geq 0}$ and $\overline{\text{TP}}_{=+\infty}$ (see [3, Appendix D] for a full proof).

► **Corollary 17.** *Step-counter strategies suffice uniformly for $\overline{\text{MP}}_{\geq 0}$ and $\overline{\text{TP}}_{=+\infty}$.*

To illustrate Theorem 16 further, we apply it to a non-quantitative objective.

► **Example 18.** Let C be at most countable and $O \subseteq C^\omega$ be the objective requiring that all colours are seen infinitely often (it is an intersection of *Büchi conditions*). Formally, $O = \bigcap_{c \in C} \bigcap_{i \geq 1} \bigcup_{j \geq i} \{w = c_1 c_2 \dots \in C^\omega \mid c_j = c\}$. This objective is prefix-independent and in Π_2^0 : it is the countable intersection of the open, step-monotonic objectives $O_{c,i} = \bigcup_{j \geq i} \{w = c_1 c_2 \dots \in C^\omega \mid c_j = c\}$. By Theorem 16, step-counter strategies suffice over finitely branching arenas for O . This result is relatively tight: finite-memory strategies do not suffice over finitely branching arenas when C is infinite, and step-counter strategies do not suffice over infinitely branching arenas when $|C| = 2$ (see [3, Appendix D] for details). ◻



■ **Figure 6** Arena \mathcal{A} used in Example 19. Player 1 has a winning 1-bit strategy for $\overline{\text{TP}}_{\geq 0}$, but no winning step-counter strategy.

6 A non-prefix-independent Π_2^0 objective

In this section, we consider objective $\overline{\text{TP}}_{\geq 0} = \bigcap_{m \geq 1} \bigcap_{i \geq 1} \bigcup_{j \geq i} \{w \in C^\omega \mid \text{TP}(w_{\leq j}) \geq \frac{-1}{m}\}$ (in Π_2^0). Its definition is very close to the one of $\overline{\text{MP}}_{\geq 0}$ from the previous section, but one important difference is that it is not prefix-independent (for instance, $0^\omega \in \overline{\text{TP}}_{\geq 0}$, but $-1, 0^\omega \notin \overline{\text{TP}}_{\geq 0}$). Hence, Theorem 16 does not apply.

As argued in Lemma 5, it turns out that step-counter strategies do not suffice for $\overline{\text{TP}}_{\geq 0}$, even over finitely branching arenas. We show a second example, only suited for this particular objective, illustrating more clearly the trade-off to consider to build simple winning strategies.

► **Example 19.** Consider the arena \mathcal{A} in Figure 6. We assume that a play starts in v_0 , hence reaching sum of weights -1 in v_1 . We assume that a play is decomposed into rounds, where round i corresponds to the choice of Player 2 and Player 1 in v_i and u_i respectively. At each round i , Player 2 and then Player 1 choose either 0, or i followed by $-i-1$. As previously, we can assume that this arena only uses weights in $C = \{-1, 0, 1\}$, and that all histories from v_0 reaching the same vertex have the same length.

Player 1 has a winning strategy, consisting of playing “the opposite” of what Player 2 just played: if Player 2 played the sequence of 0 (resp. $i, -i-1$), then Player 1 replies with $i, -i-1$ (resp. the sequence of 0). This ensures that (i) the current sum of weights in v_i is exactly $-i$ (it starts at -1 in v_1 and decreases by 1 at each round), and (ii) the current sum of weights reaches exactly 0 once during each round, after i is played. This shows that this strategy is winning for $\overline{\text{TP}}_{\geq 0}$. Such a strategy can be implemented with two memory states that simply remember the choice of Player 2 at each round.

As all histories leading to vertices u_i have the same length, a step-counter strategy cannot distinguish the choices of Player 2 (Lemma 1). Any step-counter strategy is losing:

- either Player 1 only plays 0, in which case Player 2 wins by only playing 0, thereby ensuring that the current sum of weights is -1 from v_1 onwards;
- or Player 1 plays $i, -i-1$ at some u_i . In this case, Player 2 wins by only playing $i, -i-1$. This means that the sum of weights decreases by at least 1 at every round, but decreases by 2 in round i . Hence, for $j \geq i$, the sum of weights at round j is at most $-j-1$. Such a sum can never go above 0 again when a player plays $j, -j-1$. \lrcorner

This example shows that in general, there is a trade-off between “obtaining a high value for a short time, to go above 0 temporarily” and “playing safe in order not to decrease the value too much”. Two memory states sufficed: if the opponent just saw a high sum of weights (≥ 0), then we can play it safe temporarily; if the opponent played it safe, we may need to aim for a high value, even if the overall sum decreases. This reasoning generalises to all finitely branching arenas: in general, step-counter + 1-bit strategies suffice for $\overline{\text{TP}}_{\geq 0}$.

► **Theorem 20.** *Step-counter + 1-bit strategies suffice for $\overline{\text{TP}}_{\geq 0}$ over finitely branching arenas.*

We provide a proof sketch here (full proof in [3, Appendix E]). It follows the same scheme as the proof of Theorem 16, where we inductively fix choices for ever longer histories. However, we need to be more careful not to leave the winning region. As the objective is not prefix-independent, the winning region $W'_{\mathcal{A},1}$ is described not just by a set of vertices, but by pairs of a vertex and current total payoff (i.e., the current sum of weights), i.e., $W'_{\mathcal{A},1} \subseteq V \times \mathbb{Q}$.

We start with a lemma about the sufficiency of memoryless strategies to *stay* in this winning region. Staying in $W'_{\mathcal{A},1}$ is necessary but not sufficient to win for $\overline{\text{TP}}_{\geq 0}$.

► **Lemma 21.** *Let $\mathcal{A} = (V, V_1, V_2, E)$ be a finitely branching arena. There exists a memoryless strategy σ_{safe} of Player 1 in \mathcal{A} such that, for every $(v_0, r) \in W'_{\mathcal{A},1}$, σ_{safe} never leaves $W'_{\mathcal{A},1}$ from v_0 with initial weight value r .*

The following lemma is an analogue of Lemma 14, but ensures a stronger property with a more complex memory structure (using an extra bit). It says that locally, with a step-counter + 1-bit strategy, we can guarantee a high value temporarily while staying in the winning region $W'_{\mathcal{A},1}$, generalising the phenomenon of Example 19. The bit is used to aim for a high value (bit value 0) or stay in the winning region (bit value 1) by playing σ_{safe} from Lemma 21.

We use a rewriting of $\overline{\text{TP}}_{\geq 0}$: observe that

$$\overline{\text{TP}}_{\geq 0} = \bigcap_{m \geq 1} \bigcup_{j \geq m} \left\{ w \in C^\omega \mid \text{TP}(w_{\leq j}) \geq \frac{-1}{m} \right\}, \quad (1)$$

where the variable m is used both for the $-\frac{1}{m}$ lower bound and for the m lower bound on the step count. Indeed, this also enforces that, for arbitrarily long prefixes, the current total payoff goes above values arbitrarily close to 0. For $m \geq 1$, let $O_m = \bigcup_{j \geq m} \{w \mid \text{TP}(w_{\leq j}) \geq \frac{-1}{m}\}$ be the open set used in the definition of $\overline{\text{TP}}_{\geq 0}$ in (1).

► **Lemma 22.** *Let $\mathcal{A} = (V, V_1, V_2, E)$ be an arena and $v_0 \in V$ be an initial vertex in the winning region of Player 1 for $\overline{\text{TP}}_{\geq 0}$. For all $m \geq 1$, there exists a step-counter + 1-bit strategy σ_m such that σ_m is winning for O_m from v_0 and never leaves $W'_{\mathcal{A},1}$ (i.e., for all histories h from v_0 consistent with σ_m , $(\text{to}(h), \text{TP}(h)) \in W'_{\mathcal{A},1}$).*

The inductive scheme used in the proof of Theorem 20 is similar to that of Theorem 16, building a step-counter + 1-bit strategy $\sigma: V_1 \times \mathbb{N} \times \{0, 1\} \rightarrow E$.

For \mathcal{M} a step-counter and 1-bit memory structure, consider the product arena $\mathcal{A}' = \mathcal{A} \otimes \mathcal{M}$ (in which the bit updates are not fixed yet, and will be fixed inductively). We have that $(v_0, (0, 0))$ is in the winning region of \mathcal{A}' . The inductive scheme is as follows: for infinitely many $m \in \mathbb{N}$, for some step bound $k_m \in \mathbb{N}$, we fix σ on $V_1 \times \{0, \dots, k_m - 1\} \times \{0, 1\}$, yielding arena \mathcal{A}'_m . Using Lemma 22, we ensure that

- along all histories h from v_0 consistent with σ of length at most k_m , $W'_{\mathcal{A},1}$ is not left, and
- the open objective O_m is already satisfied within k_m steps (i.e., any history of length k_m consistent with σ only has winning continuations for O_m).

Iterating this procedure defines a step-counter + 1-bit strategy σ that satisfies O_m for infinitely many $m \geq 1$. Since the sequence $(O_m)_{m \geq 1}$ is decreasing ($O_1 \supseteq O_2 \supseteq \dots$), we have that σ is winning for O_m for all $m \geq 1$. Hence, σ is winning for $\overline{\text{TP}}_{\geq 0}$.

► **Remark 23.** Unlike for Theorem 16, the upper bound in this section does not apply *uniformly* in general (an arena illustrating this is in [3, Appendix E]). ◻

► **Remark 24.** Over integer weights ($C \subseteq \mathbb{Z}$), $\overline{\text{TP}}_{> 0} = \overline{\text{TP}}_{\geq 1} \in \Pi_2^0$. As $\overline{\text{TP}}_{\geq 1}$ behaves like $\overline{\text{TP}}_{\geq 0}$ (Remark 2), the results from this section apply to $\overline{\text{TP}}_{> 0}$ over integer weights. Up to some scaling factor, this also applies to rational weights with bounded denominators. However, for general rational weights, $\overline{\text{TP}}_{> 0}$ can only be shown to be in Σ_3^0 , so the above does not apply. ◻

7 Conclusion

We established whether step-counter strategies (possibly with finite memory) suffice for the objectives $\overline{MP}_{\geq 0}$, $\overline{TP}_{> 0}$, $\overline{TP}_{\geq 0}$, $\overline{TP}_{=+\infty}$, and $\overline{TP}_{\geq 0}$. We used the structure of these objectives as sets in the Borel hierarchy, and pinpointed the strategy complexity for all classical quantitative objectives on the second level of Borel hierarchy. This leaves open the cases of $\overline{MP}_{> 0}$, $\overline{MP}_{\geq 0}$, $\overline{TP}_{> 0}$ (over \mathbb{Q}), and $\overline{TP}_{=+\infty}$, all on the third level. The sufficiency of other less common infinite memory structures, such as *reward counters* [25], could also be investigated.

References

- 1 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 921–962. Springer, 2018. doi:10.1007/978-3-319-10575-8_27.
- 2 Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. Optimal strategies in concurrent reachability games. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14–19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 7:1–7:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.CSL.2022.7.
- 3 Sougata Bose, Rasmus Ibsen-Jensen, David Purser, Patrick Totzke, and Pierre Vandenhover. The power of counting steps in quantitative games. *CoRR*, abs/2406.17482, 2024. doi:10.48550/arXiv.2406.17482.
- 4 Sougata Bose, Rasmus Ibsen-Jensen, and Patrick Totzke. Bounded-memory strategies in partial-information games. In *ACM/IEEE Symposium on Logic in Computer Science, LICS '24*, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3661814.3662096.
- 5 Patricia Bouyer, Nathanaël Fijalkow, Mickael Randour, and Pierre Vandenhover. How to play optimally for regular objectives? In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10–14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 118:1–118:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.118.
- 6 J. Richard Büchi and Lawrence H. Landweber. Definability in the monadic second-order theory of successor. *Journal of Symbolic Logic*, 34(2):166–170, 1969. doi:10.2307/2271090.
- 7 Antonio Casares. On the minimisation of transition-based Rabin automata and the chromatic memory requirements of Muller conditions. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14–19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CSL.2022.12.
- 8 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *Proceedings of the 3rd International Conference on Embedded Software, EMSOFT 2003, Philadelphia, PA, USA, October 13–15, 2003*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003. doi:10.1007/978-3-540-45212-6_9.
- 9 Thomas Colcombet, Nathanaël Fijalkow, and Florian Horn. Playing safe, ten years later. *Logical Methods in Computer Science*, 20(1), 2024. doi:10.46298/LMCS-20(1:10)2024.
- 10 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, LICS 1997, Warsaw, Poland, June 29 – July 2, 1997*, pages 99–110, 1997. doi:10.1109/.1997.614939.
- 11 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979. doi:10.1007/BF01768705.

- 12 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, FOCS 1991, San Juan, Puerto Rico, October, 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.
- 13 Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, and Mateusz Skomra. Games on graphs, 2023. arXiv:2305.10546.
- 14 Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer New York, 1996. URL: <https://books.google.fr/books?id=211cbnzDNwsC>.
- 15 Dean Gillette. *Stochastic Games with Zero Stop Probabilities*, pages 179–188. Princeton University Press, Princeton, 1957. doi:10.1515/9781400882151-011.
- 16 Hugo Gimbert and Wiesław Zielonka. When can you play positionally? In Jiří Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science, MFCS 2004, Prague, Czech Republic, August 22–27, 2004*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–697. Springer, 2004. doi:10.1007/978-3-540-28629-5_53.
- 17 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.
- 18 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC 1982, San Francisco, CA, USA, May 5–7, 1982*, pages 60–65. ACM, 1982. doi:10.1145/800070.802177.
- 19 Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, and Abraham Neyman. The big match with a clock and a bit of memory. *Mathematics of Operations Research*, 48(1):419–432, 2023. doi:10.1287/moor.2022.1267.
- 20 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Strategy complexity of parity objectives in countable MDPs. In Igor Konnov and Laura Kovács, editors, *Proceedings of the 31st International Conference on Concurrency Theory, CONCUR 2020, September 1–4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPICs*, pages 39:1–39:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CONCUR.2020.39.
- 21 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Memoryless strategies in stochastic reachability games, 2024. To appear in *Lecture Notes in Computer Science*.
- 22 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Strategy complexity of Büchi objectives in concurrent stochastic games, 2024. arXiv:2404.15483.
- 23 Ashok P. Maitra and William D. Sudderth. *Discrete Gambling and Stochastic Games*. Springer-Verlag, 1996.
- 24 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. URL: <http://www.jstor.org/stable/1971035>.
- 25 Richard Mayr and Eric Munday. Strategy complexity of point payoff, mean payoff and total payoff objectives in countable MDPs. *Logical Methods in Computer Science*, 19(1), 2023. doi:10.46298/LMCS-19(1:16)2023.
- 26 Pierre Ohlmann. *Monotonic graphs for parity and mean-payoff games*. PhD thesis, IRIF – Research Institute on the Foundations of Computer Science, 2021.
- 27 Pierre Ohlmann and Michał Skrzypczak. Positionality in Σ_2^0 and a completeness result. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *Proceedings of the 41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12–14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPICs*, pages 54:1–54:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.STACS.2024.54.




13:18 The Power of Counting Steps in Quantitative Games

- 28 Anuj Puri. *Theory of Hybrid Systems and Discrete Event Systems*. PhD thesis, EECS Department, University of California, Berkeley, December 1995. URL: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/1995/2950.html>.
- 29 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. doi:10.1002/9780470316887.
- 30 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969. doi:10.2307/1995086.
- 31 Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.
- 32 Frank Thuijsman. *Optimality and Equilibria in Stochastic Games*. Number no. 82 in CWI Tract – Centrum voor Wiskunde en Informatica. Centrum voor Wiskunde en Informatica, 1992. URL: <https://books.google.co.uk/books?id=sfzuAAAAMAAJ>.
- 33 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

As Soon as Possible but Rationally




Véronique Bruyère   

Université de Mons (UMONS), Belgium

Christophe Grandmont   

Université de Mons (UMONS), Belgium

Université Libre de Bruxelles (ULB), Belgium

Jean-François Raskin   

Université Libre de Bruxelles (ULB), Belgium

Abstract

This paper addresses complexity problems in rational verification and synthesis for multi-player games played on weighted graphs, where the objective of each player is to minimize the cost of reaching a specific set of target vertices. In these games, one player, referred to as the system, declares his strategy upfront. The other players, composing the environment, then rationally make their moves according to their objectives. The rational behavior of these responding players is captured through two models: they opt for strategies that either represent a Nash equilibrium or lead to a play with a Pareto-optimal cost tuple.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Solution concepts in game theory; Theory of computation → Logic and verification

Keywords and phrases Games played on graphs, rational verification, rational synthesis, Nash equilibrium, Pareto-optimality, quantitative reachability objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.14

Related Version *Full Version:* <https://arxiv.org/abs/2403.00399> [17]

Funding This work has been supported by the Fonds de la Recherche Scientifique – FNRS under Grant n° T.0023.22 (PDR Rational).

Jean-François Raskin: Supported by Fondation ULB (<https://www.fondationulb.be/en/>).

1 Introduction

Nowadays, formal methods play a crucial role in ensuring the reliability of critical computer systems. Still, the application of formal methods on a large scale remains elusive in certain areas, notably in multi-agent systems. Those systems pose a significant challenge for formal verification and automatic synthesis because of their heterogeneous nature, encompassing everything from conventional reactive code segments to fully autonomous robots and even human operators. Crafting formal models that accurately represent the varied components within these systems is often a too complex task.

Although constructing detailed operational models for humans or sophisticated autonomous robots might be problematic, it is often more feasible to identify the *overarching goals* that those agents pursue. Incorporating these goals is crucial in the design and validation process of systems that interact with such entities. Typically, a system is not expected to function flawlessly under all conditions but rather in scenarios where the agents it interacts with act in alignment with their objectives, i.e., they *behave rationally*. *Rational synthesis* focuses on creating a system that meets its specifications against any behavior of environmental agents that is guided by their goals (and not against any of their behaviors). *Rational verification* studies the problem of ensuring that a system enforces certain correctness properties, not in all conceivable scenarios, but specifically in scenarios where environmental agents behave in accordance with their objectives.



© Véronique Bruyère, Christophe Grandmont, and Jean-François Raskin;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 14; pp. 14:1–14:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 1** Summary of complexity results.

	Non-coop. verif.	Universal non-coop. verif.	Coop. synthesis	Non-coop. synthesis
PO, weights	Π_2^P -complete	PSPACE-complete	PSPACE-complete	NEXPTIME-complete [11]
PO, qualitative	Π_2^P -complete	PSPACE-complete	PSPACE-complete	NEXPTIME-complete [18]
NE, weights	coNP-complete	coNP-complete	NP-complete	Unknown, EXPTIME-hard ¹⁾
NE, qualitative	coNP-complete [27]	coNP-complete [27]	NP-complete [21]	PSPACE-complete [21]

1) For the important special case of one-player environments, we provide an algorithm that runs in EXPTIME and we can prove PSPACE-hardness. The EXPTIME-hardness of the general case already holds for two-player environments.

Rationality can be modeled in various ways. In this paper, we focus on two general approaches. The first approach comes from game theory where rationality is modeled by the concept of equilibrium, such as *Nash equilibria* (NE) [35] or *subgame perfect equilibria* (SPE), a refinement of NEs [36]. The second approach treats the environment as a single agent but with multiple, sometimes conflicting, goals, aiming for behaviors that achieve a *Pareto-optimal* balance among these objectives. The concept of Pareto-optimality (PO) and its application in multi-objective analysis have been explored primarily in the field of optimization [37], but also in formal methods [2, 4]. These two notions of rationality are different in nature: in the first setting, each component of the environment playing an equilibrium is considered to be an independent selfish individual, excluding cooperation scenarios, while in the second setting, several components of the environment can collaborate and agree on trade-offs. The challenge lies in adapting these concepts to *reactive systems* characterized by ongoing, non-terminating interactions with their environment. This necessitates the transition from two-player zero-sum games on graphs, the classical approach used to model a fully adversarial environment (see e.g. [38]), to the more complex setting of *multi-player non zero-sum games on graphs* used to model environments composed of various rational agents.

Rational synthesis and rational verification have attracted large attention recently, see e.g. [7, 19, 21, 26, 28, 29, 33, 34]. But the results obtained so far, with a few exceptions that we detail below, are limited to the *qualitative* setting formalized as Boolean outcomes associated with ω -regular objectives. Those objectives are either specified using linear temporal specifications or automata over infinite words (like parity automata). The complexity of those problems is now well understood (with only a few complexity gaps remaining, see e.g. [21, 34]). The methods to solve those problems and get completeness results for worst-case complexity are either based on automata theory (using mainly automata over infinite trees) or by reduction to zero-sum games. *Quantitative* objectives are less studied and revealed to be much more challenging. For instance, it is only very recently that the rational verification problem was studied, for SPEs in non zero-sum games with mean-payoff, energy, and discounted-sum objectives in [7], for an LTL specification in multi-agent systems that behave according to an NE with mean-payoff objectives in [29] or with quantitative probabilistic LTL objectives in [30]. In [1], the rational synthesis problem was studied for the quantitative extension $LTL[\mathcal{F}]$ of LTL where the Boolean operators are replaced with arbitrary functions mapping binary tuples into the interval $[0, 1]$.

In this paper, we consider *quantitative reachability* objectives. Our choice for studying these objectives was guided by their fundamental nature and also by their relative simplicity. Nevertheless, as we will see, they are challenging for both rational synthesis and rational verification. Indeed, to obtain worst-case optimal algorithms and establish completeness results, we had to resort to the use of *innovative* theoretical tools, more advanced than those necessary for the qualitative framework. In our endeavor, we have established the exact complexity of most studied decision problems in rational synthesis and rational verification.

Technical Contributions. In this work, we explore both verification and synthesis problems through the lens of rationality, defined by Pareto-optimality and Nash equilibria, for quantitative reachability objectives. For the synthesis problem, we also consider the *cooperative* variant where the environment cooperates with the system: we want to decide whether the system has a strategy and the environment a rational response to this strategy such that the objective of the system is enforced. Our results are presented in Table 1, noting that all results lacking explicit references are, to our knowledge, novel contributions. For completeness, the table includes (new and known) results for the qualitative scenario.

The results for *PO rationality* are as follows. (1) For the verification problems, we assume that the behavior of the system is formalized by a *nondeterministic Mealy machine*, used to represent a (usually infinite) set of its possible implementations. For each of those implementations, we verify that the quantitative reachability objective of the system is met against any rational behavior of the environment. We establish that this problem is PSPACE-complete. To obtain the upper bound, we rely on a *genuine combination of techniques* based on Parikh automata and a recursive PSPACE algorithm (for positive Boolean combinations of bounded safety objectives, a problem of independent interest). Parikh automata are used to guess a compact representation of certificates which are paths of possibly exponential length in the size of the problem input. When the Mealy machine is deterministic, we show that the complexity goes down to Π_2^P -completeness, as the previous PSPACE algorithm is replaced by a coNP oracle. (2) For the synthesis problems, we only consider the cooperative version which we prove to be PSPACE-complete, as the non-cooperative version was established to be NEXPTIME-complete in [11].

The results for *NE rationality* are as follows. (1) We establish that, surprisingly, the verification problems are coNP-complete both for the general case of a nondeterministic Mealy machine and for the special case where it is deterministic. The upper bounds for those problems are again based on Parikh automata certificates but here there is no need to use a coNP oracle. (2) For the synthesis problems, the landscape is more challenging. For the cooperative case, we were able to establish that the problem is NP-complete. For the non-cooperative case, we have partially solved the problem and established the following results. When the environment is composed of a single rational player, the problem is in EXPTIME and PSPACE-hard. For an environment with at least two players, we show that the problem is EXPTIME-hard but we leave its decidability open. The lower bounds are obtained using an elegant reduction from countdown games [31]. We give indications in the paper why the problem is difficult to solve and why classical automata-theoretic methods *may not be sufficient* (if the problem is decidable).

In this paper, we focus on nonnegative weights as we show that considering arbitrary weights leads to undecidability of the synthesis problems. We also focus on NEs instead of SPEs, even if the latter are a better concept to model rational behavior in games played on graphs. Indeed, it is well-known that SPEs pose greater challenges than NEs. So, starting with NEs offers a better initial step for the algorithmic treatment of rational verification and synthesis in quantitative scenarios, an area that remains largely unexplored.

Related Work. The survey [15] presents several results about different game models and different kinds of objectives related to reachability. Quantitative objectives in *two-player zero-sum games* were largely studied, see e.g. [13, 20, 22], even if exact complexity results are often elusive due to the intricate nature of the problems (e.g. the exact complexity of solving mean-payoff games is still an open problem). In multi-player non zero-sum games, the (*constrained*) *existence* of equilibria is also well studied. The existence of simple NEs

was established in [12] for mean-payoff and discounted-sum objectives. No decision problem is considered in that paper. The constrained existence of SPEs in quantitative reachability games was proved PSPACE-complete in [8]. We prove here that the complexity is lower when we use NEs to model rationality, as we obtain NP-completeness for the related cooperative synthesis problem. Deciding the constrained existence of SPEs was recently solved for quantitative reachability games in [9] and for mean-payoff games in [5, 6]. The cooperative and non-cooperative rational *synthesis problems* were studied in [25] for games with mean-payoff and discounted-sum objectives when the environment is composed of a single player. The mean-payoff case was proved to be NP-complete and the discounted-sum case was linked to the open target discounted sum problem, which explains the difficulty of solving the problem in this case.

Structure of the Paper. The background is given in Section 2. The formal definitions of the studied problems and our main complexity results are stated in Section 3. The proofs of our results are given for PO rationality in Section 4, and for NE rationality in Section 5. We give a conclusion and future work in Section 6.

2 Background

Arenas and Plays. A (finite) *arena* \mathcal{A} is a tuple $(V, E, \mathcal{P}, (V_i)_{i \in \mathcal{P}})$ where V is a finite set of *vertices*, $E \subseteq V \times V$ is a set of *edges*, \mathcal{P} is a finite set of *players*, and $(V_i)_{i \in \mathcal{P}}$ is a partition of V , where V_i is the set of vertices *owned* by player i . We assume that $v \in V$ has at least one *successor*, i.e., the set $\text{succ}(v) = \{v' \in V \mid (v, v') \in E\}$ is nonempty.

We define a *play* $\pi \in V^\omega$ (resp. a *history* $h \in V^*$) as an infinite (resp. finite) sequence of vertices $\pi_0 \pi_1 \dots$ such that $(\pi_i, \pi_{i+1}) \in E$ for any two consecutive vertices π_i, π_{i+1} . The *length* $|h|$ of a history h is the number of its vertices. The empty history is denoted ε . Given a play π and two indexes $k < k'$, we write $\pi_{\leq k}$ the prefix $\pi_0 \dots \pi_k$ of π , $\pi_{\geq k}$ the suffix $\pi_k \pi_{k+1} \dots$ of π , and $\pi_{[k, k']}$ for $\pi_k \dots \pi_{k'-1}$. We denote the first vertex of π by $\text{first}(\pi)$. These notations are naturally adapted to histories. We also write $\text{last}(h)$ for the last vertex of a history $h \neq \varepsilon$. The set of all plays (resp. histories) of an arena \mathcal{A} is denoted $\text{Plays}_{\mathcal{A}} \subseteq V^\omega$ (resp. $\text{Hist}_{\mathcal{A}} \subseteq V^*$), and we write Plays (resp. Hist) when the context is clear. For $i \in \mathcal{P}$, the set $\text{Hist}_i \subseteq V^* V_i$ represents all histories ending in a vertex $v \in V_i$. That is, $\text{Hist}_i = \{h \in \text{Hist} \mid h \neq \varepsilon \text{ and } \text{last}(h) \in V_i\}$.

We can *concatenate* two nonempty histories h_1 and h_2 into a single one, denoted $h_1 \cdot h_2$ or $h_1 h_2$ if $(\text{last}(h_1), \text{first}(h_2)) \in E$. When a history can be concatenated to itself, we call it a *cycle*. Furthermore, a play $\pi = \mu \nu \nu \dots = \mu(\nu)^\omega$ where $\mu \nu \in \text{Hist}$ with ν a cycle, is called a *lasso*. The *length* of π is then the length of $\mu \nu$.² Given a play π , a *cycle along* π refers to a sequence $\pi_{[m, n]}$ with $\pi_m = \pi_n$. We denote $\text{Occ}(\pi) = \{v \in V \mid \exists n \in \mathbb{N}, v = \pi_n\}$ the set of all vertices occurring along π , and we say that π *visits* or *reaches* a vertex $v \in \text{Occ}(\pi)$ or a set T such that $T \cap \text{Occ}(\pi) \neq \emptyset$. The previous notions extend to histories.

Given an arena \mathcal{A} , if we fix an *initial vertex* $v_0 \in V$, we say that \mathcal{A} is *initialized* and we denote by $\text{Plays}(v_0)$ (resp. $\text{Hist}(v_0)$) all its plays (resp. nonempty histories) starting with v_0 . An arena is called *weighted* if it is augmented with a non-negative *weight function* $w_i : E \rightarrow \mathbb{N}$ for each player i . We denote by W the greatest weight, i.e., $W = \max\{w_i(e) \mid e \in E, i \in \mathcal{P}\}$. We extend w_i to any history $h = \pi_0 \dots \pi_n$ such that $w_i(h) = \sum_{j=1}^n w_i((\pi_{j-1}, \pi_j))$.

² To have a well-defined length for a lasso π , we assume that $\pi = \mu(\nu)^\omega$ with $\mu \nu$ of minimal length.

Reachability Games. A *reachability game* is a tuple $\mathcal{G} = (\mathcal{A}, (T_i)_{i \in \mathcal{P}})$ where \mathcal{A} is a weighted arena and $T_i \subseteq V$ is a *target set* for each $i \in \mathcal{P}$. We define a *cost function* $\text{cost}_i : \text{Plays} \rightarrow \mathbb{N} \cup \{+\infty\}$ for each player i , such that for all plays $\pi = \pi_0\pi_1 \dots \in \text{Plays}$, $\text{cost}_i(\pi) = w_i(\pi_0 \dots \pi_n)$ with n the smallest index such that $\pi_n \in T_i$, if it exists and $\text{cost}_i(\pi) = +\infty$ otherwise.

The *reachability objective* of player i is to *minimize* this cost as much as possible, i.e., given two plays π, π' such that $\text{cost}_i(\pi) < \text{cost}_i(\pi')$, player i prefers π to π' . We extend $<$ to tuples of costs as follows: $(\text{cost}_i(\pi))_{i \in \mathcal{P}} < (\text{cost}_i(\pi'))_{i \in \mathcal{P}}$ if $\text{cost}_i(\pi) \leq \text{cost}_i(\pi')$ for all $i \in \mathcal{P}$ and there exists some $j \in \mathcal{P}$ such that $\text{cost}_j(\pi) < \text{cost}_j(\pi')$. Given a play π , we denote by $\text{Visit}(\pi)$ the set of players i such that π visits T_i , i.e., $\text{Visit}(\pi) = \{i \in \mathcal{P} \mid \text{cost}_i(\pi) < +\infty\}$. When for all $i \in \mathcal{P}$ and $e \in E$, $w_i(e) = 0$, we speak of *qualitative reachability games*, since $\text{cost}_i(\pi) = 0$ if $\text{Occ}(\pi) \cap T_i \neq \emptyset$ and $+\infty$ otherwise.

Strategies and Mealy Machines. Let $\mathcal{A} = (V, E, \mathcal{P}, (V_i)_{i \in \mathcal{P}})$ be an arena. A *strategy* $\sigma_i : \text{Hist}_i \rightarrow V$ for player i maps any history $h \in \text{Hist}_i$ to a vertex $v \in \text{succ}(\text{last}(h))$, which is the next vertex that player i chooses to move to after reaching the last vertex in h . The set of all strategies of player i is denoted Σ_i . A play $\pi = \pi_0\pi_1 \dots$ is *consistent* with σ_i if $\pi_{k+1} = \sigma_i(\pi_0 \dots \pi_k)$ for all $k \in \mathbb{N}$ such that $\pi_k \in V_i$. Consistency is naturally extended to histories. A tuple of strategies $\sigma = (\sigma_i)_{i \in \mathcal{P}}$ with $\sigma_i \in \Sigma_i$, is called a *strategy profile*. In an arena initialized at v_0 , we limit the domain of each strategy σ_i to $\text{Hist}_i(v_0)$; the play π starting from v_0 and consistent with each σ_i is denoted $\langle \sigma \rangle_{v_0}$ and called *outcome*.

Given an initialized arena \mathcal{A} , we can encode a strategy or a set of strategies by a (finite) *nondeterministic Mealy machine* [7, 19] $\mathcal{M} = (M, m_0, \delta, \tau)$ on \mathcal{A} , where M is a finite set of *memory states*, $m_0 \in M$ is the *initial state*, $\delta : M \times V \rightarrow 2^M$ is the *update function*, and $\tau : M \times V_i \rightarrow 2^V$ is the *next-move function*. Such a machine embeds a (possibly infinite) set of strategies σ_i for player i , called *compatible strategies*. Formally, σ_i is compatible with \mathcal{M} if there exists a mapping $h \mapsto m_h$ such that $m_{hv} \in \delta(m_h, v)$ for every $hv \in \text{Hist}(v_0)$ (with $m_h = m_0$ when h is empty), and when $v \in V_i$, $\sigma_i(hv) \in \tau(m_h, v)$. An example of such a machine \mathcal{M} is given in Appendix A. We denote by $\llbracket \mathcal{M} \rrbracket$ the set of all strategies compatible with \mathcal{M} . The *memory size* of \mathcal{M} is equal to $|M|$. We say that \mathcal{M} is *deterministic* when the image of both functions δ and τ is a singleton. Thus when \mathcal{M} is deterministic, $\llbracket \mathcal{M} \rrbracket = \{\sigma_i\}$ and σ_i is called *finite-memory*, and when additionally $|M| = 1$, σ_i is called *memoryless*.

3 Studied Problems

In this section, within the context of rational synthesis and verification, we consider a reachability game $\mathcal{G} = (\mathcal{A}, (T_i)_{i \in \mathcal{P}})$ with \mathcal{A} an initialized weighted arena and $\mathcal{P} = \{0, 1, \dots, t\}$ such that player 0 is a specific player, often called *system* or *leader*, and the other players $1, \dots, t$ compose the *environment* and are called *followers*. Player 0 announces his strategy σ_0 at the beginning of the game and is not allowed to change it according to the behavior of the other players. The response of those players to σ_0 is supposed to be *rational*, where the rationality can be described as the outcome of a *Nash equilibrium* [35] or as a *Pareto-optimal* play [18].

Nash Equilibria. A strategy profile for the environment is a Nash equilibrium if no player has an incentive to unilaterally deviate from this profile. In other words, no player can improve his cost by switching to a different strategy, assuming that the other players stick to their current strategies. Formally, given the initial vertex v_0 and a strategy σ_0 announced by player 0, a strategy profile $\sigma = (\sigma_i)_{i \in \mathcal{P}}$ is called a *0-fixed Nash equilibrium* (0-fixed NE) if for every player $i \in \mathcal{P} \setminus \{0\}$ and every strategy $\tau_i \in \Sigma_i$, it holds that $\text{cost}_i(\langle \sigma \rangle_{v_0}) \leq \text{cost}_i(\langle \tau_i, \sigma_{-i} \rangle_{v_0})$, where σ_{-i} denotes $(\sigma_j)_{j \in \mathcal{P} \setminus \{i\}}$, i.e., τ_i is not a profitable deviation. We also say that σ is a *σ_0 -fixed NE* to emphasize the strategy σ_0 of player 0.

Pareto-Optimality. When all players collaborate to obtain a best cost for everyone, we need another concept of rationality. In that case, we suppose that the players in $\mathcal{P} \setminus \{0\}$ form a *single* player, player 1, that has a *tuple* of targets sets $(T_i)_{i \in \{1, \dots, t\}}$. For each play $\pi \in \text{Plays}(v_0)$, player 1 gets a cost tuple $\text{cost}_{\text{env}}(\pi) = (\text{cost}_i(\pi))_{i \in \{1, \dots, t\}}$, and prefers π to π' if $\text{cost}_{\text{env}}(\pi) < \text{cost}_{\text{env}}(\pi')$ for the componentwise partial order $<$ over $(\mathbb{N} \cup \{+\infty\})^t$. Given such a modified game and a strategy σ_0 announced by player 0, we consider the set C_{σ_0} of cost tuples of plays consistent with σ_0 that are *Pareto-optimal* for player 1, i.e., minimal with respect to $<$. Hence, $C_{\sigma_0} = \min\{\text{cost}_{\text{env}}(\pi) \mid \pi \in \text{Plays}(v_0) \text{ consistent with } \sigma_0\}$. Notice that C_{σ_0} is an antichain. A cost tuple p (called cost in the sequel) is said to be σ_0 -fixed Pareto-optimal (σ_0 -fixed PO or simply 0-fixed PO) if $p \in C_{\sigma_0}$. Similarly, a play is said to be σ_0 -fixed PO if its cost is σ_0 -fixed PO.

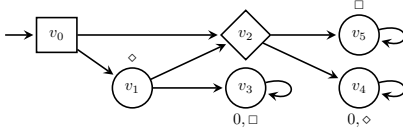
In some problems studied in this paper, we will have to consider games such that all vertices owned by player 0 have only one successor, which means that player 0 has no choice but to choose this successor. In this case, we say that *player 1 is the only one to play*.

Rational Verification. We now present the studied decision problems related to the concept of *rational verification*. Given some threshold $c \in \mathbb{N}$, the goal is to verify that a strategy σ_0 announced by player 0 guarantees him a cost $\text{cost}_0(\pi) \leq c$ whatever the rational response π of the environment. By rational response, we mean either a σ_0 -fixed NE outcome π , or a σ_0 -fixed PO play π . The strategy σ_0 is usually given as a deterministic Mealy machine. We can go further: with a nondeterministic Mealy machine, we want to verify whether all strategies $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$ are solutions. In the latter case, we speak about *universal* verification.

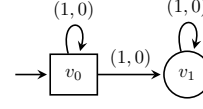
- **Problem 1.** *Given a reachability game \mathcal{G} with an initialized arena, a nondeterministic Mealy machine \mathcal{M}_0 for player 0, and a threshold $c \in \mathbb{N}$,*
- *If $\llbracket \mathcal{M}_0 \rrbracket = \{\sigma_0\}$, the Non-Cooperative Nash Verification problem (NCNV) asks whether for all σ_0 -fixed NEs σ , it holds that $\text{cost}_0(\langle \sigma \rangle_{v_0}) \leq c$.*
 - *The Universal Non-Cooperative Nash Verification problem (UNCNV) asks whether for all $\sigma_0 \in \llbracket \mathcal{M}_0 \rrbracket$ and all σ_0 -fixed NEs σ , it holds that $\text{cost}_0(\langle \sigma \rangle_{v_0}) \leq c$.*
 - *If $\llbracket \mathcal{M}_0 \rrbracket = \{\sigma_0\}$, the Non-Cooperative Pareto Verification problem (NCPV) asks whether for all σ_0 -fixed PO plays π , it holds that $\text{cost}_0(\pi) \leq c$.*
 - *The Universal Non-Cooperative Pareto Verification problem (UNCPV) asks whether for all $\sigma_0 \in \llbracket \mathcal{M}_0 \rrbracket$ and all σ_0 -fixed PO plays π , it holds that $\text{cost}_0(\pi) \leq c$.*

Rational Synthesis. We consider the more challenging problem of *rational synthesis*. Given a threshold $c \in \mathbb{N}$, the goal is to synthesize a strategy σ_0 for player 0 (instead of verifying some σ_0) that guarantees him a cost $\text{cost}_0(\pi) \leq c$ whatever the rational response π of the environment. We also consider the simpler problem where the environment *cooperates* with the leader by proposing *some* rational response π that guarantees him a cost $\text{cost}_0(\pi) \leq c$.

- **Problem 2.** *Given a reachability game \mathcal{G} with an initialized arena and a threshold $c \in \mathbb{N}$,*
- *The Cooperative Nash Synthesis (CNS) problem asks whether there exists $\sigma_0 \in \Sigma_0$ and a σ_0 -fixed NE σ such that $\text{cost}_0(\langle \sigma \rangle_{v_0}) \leq c$.*
 - *The Non-Cooperative Nash Synthesis (NCNS) problem asks whether there exists $\sigma_0 \in \Sigma_0$ such that for all σ_0 -fixed NEs σ , it holds that $\text{cost}_0(\langle \sigma \rangle_{v_0}) \leq c$.*
 - *The Cooperative Pareto Synthesis (CPS) problem asks whether there exists $\sigma_0 \in \Sigma_0$ and a σ_0 -fixed PO play π such that $\text{cost}_0(\pi) \leq c$.*
 - *The Non-Cooperative Pareto Synthesis (NCPS) problem asks whether there exists $\sigma_0 \in \Sigma_0$ such that for all σ_0 -fixed PO plays π , it holds that $\text{cost}_0(\pi) \leq c$.*



■ **Figure 1** An example illustrating the two concepts of rational response.



■ **Figure 2** An example showing that PO lasso plays in the coNCPV problem may have an exponential length.

► **Example 3.** To illustrate these problems, let us study a simple example depicted in Figure 1 with three players: the system, player 0, and two players in the environment, players \square and \diamond . Player 0 owns the circle vertices, player \square owns the square initial vertex v_0 , and player \diamond owns the diamond vertex v_2 . Each player i has a target set, $T_0 = \{v_3, v_4\}$, $T_\square = \{v_3, v_5\}$ and $T_\diamond = \{v_1, v_4\}$, and a constant weight $w_i(e) = 1$ for all $e \in E$. When a vertex v is in T_i , we depict the symbol of player i nearby v . As the graph is acyclic, the possible player strategies are all memoryless. In the sequel, we thus only indicate the successor chosen by the player.

Let us show that σ_0 defined by $\sigma_0(v_1) = v_2$ is a solution to the NCNS problem with the threshold $c = 3$. Given σ_0 , there exist four distinct strategy profiles $\sigma = (\sigma_0, \sigma_\square, \sigma_\diamond)$. When, for example, $\sigma_\square(v_0) = v_2$ and $\sigma_\diamond(v_2) = v_5$, we abusively denote σ as $\{v_0 \rightarrow v_2, v_2 \rightarrow v_5\}$:

- $\{v_0 \rightarrow v_2, v_2 \rightarrow v_5\}$ is not a σ_0 -fixed NE because its outcome $\pi_1 = v_0 v_2 (v_5)^\omega$ has a infinite cost for player \diamond who will deviate from v_2 to v_4 to get a cost of 2;
- similarly, $\{v_0 \rightarrow v_1, v_2 \rightarrow v_5\}$ with outcome $\pi_2 = v_0 v_1 v_2 (v_5)^\omega$ is not a σ_0 -fixed NE;
- the profile $\{v_0 \rightarrow v_1, v_2 \rightarrow v_4\}$ is a σ_0 -fixed NE, its outcome is $\pi_3 = v_0 v_1 v_2 (v_4)^\omega$ with $\text{cost}_\square(\pi_3) = +\infty$, $\text{cost}_\diamond(\pi_3) = 1$ and $\text{cost}_0(\pi_3) = 3 \leq c$, so if player \square deviates from v_1 to v_2 , his cost is still $+\infty$, and player \diamond has no incentive to deviate since $\text{cost}_\diamond(\pi_3)$ is already the smallest available;
- the profile $\{v_0 \rightarrow v_2, v_2 \rightarrow v_4\}$ with the outcome $\pi_4 = v_0 v_2 (v_4)^\omega$ is also a σ_0 -fixed NE and $\text{cost}_0(\pi_4) = 2 \leq c$.

So, σ_0 is a solution to the NCNS problem with $c = 3$, but not with $c = 2$. It is also a solution for the CNS problem. One can verify that σ'_0 such that $\sigma'_0(v_1) = v_3$ is a solution to the NCNS problem with $c = 2$, since the only σ'_0 -fixed NE outcome is $\pi_5 = v_0 v_1 (v_3)^\omega$.

We now show that σ_0 is not a solution to the CPS problem with $c = 2$. Let us consider the same four outcomes as before. Their cost for the environment are: $\text{cost}_{\text{env}}(\pi_1) = (2, +\infty)$, $\text{cost}_{\text{env}}(\pi_2) = (3, 1)$, $\text{cost}_{\text{env}}(\pi_3) = (+\infty, 1)$, and $\text{cost}_{\text{env}}(\pi_4) = (+\infty, 2)$, meaning that $C_{\sigma_0} = \{(2, +\infty), (3, 1)\}$. Consequently, the only σ_0 -fixed PO plays are π_1 and π_2 , both giving a cost of $+\infty$ to player 0. However, the strategy σ'_0 is a solution, as there is only one σ'_0 -fixed PO play, $\pi_5 = v_0 v_1 (v_3)^\omega$, with $\text{cost}_{\text{env}}(\pi_5) = (2, 1)$ and $\text{cost}_0(\pi_5) = 2$.

Main Results. Our main results for Problems 1-2 are the following ones when the rational responses of the environment are 0-fixed PO plays. One problem was already solved in [11].

► **Theorem 4.**

- (a) *The Non-Cooperative Pareto Verification problem is Π_2^P -complete.*
- (b) *The Universal Non-Cooperative Pareto Verification problem is PSPACE-complete.*
- (c) *The Cooperative Pareto Synthesis problem is PSPACE-complete.*
- (d) *The Non-Cooperative Pareto Synthesis problem is NEXPTIME-complete [11].*

For 0-fixed NE responses of the environment, we obtain the next main results.

► **Theorem 5.**

- (a) *The Non-Cooperative Nash Verification problem is coNP-complete.*
- (b) *The Universal Non-Cooperative Nash Verification problem is coNP-complete.*
- (c) *The Cooperative Nash Synthesis problem is NP-complete.*
- (d) *The Non-Cooperative Nash Synthesis problem is EXPTIME-hard, already with a two-player environment. With a one-player environment, it is in EXPTIME and PSPACE-hard.*

These complexity results depend on the size $|V|$ of the arena, the number t of players i (resp. target sets T_i) in case of 0-fixed NE responses (resp. 0-fixed PO responses), the maximal weight W encoded in binary appearing in the functions w_i , the threshold c encoded in binary, and the size $|M|$ of the Mealy machine \mathcal{M}_0 (for the verification problems). Note that for all problems except the NCNS problem, the complexity classes are the same for both qualitative and quantitative frameworks (see Table 1). Hence, in the case of a *unary* encoding of the weights and the threshold c , we get the same complexity classes. Due to space constraints, only the most challenging proofs are provided in the paper, while the other proofs or results derived from the literature are deferred in the long version of this paper [17].

In this paper, we focus on zero or positive weights, because with negative weights, there are simple examples of one-player games with no NE or no PO plays (thus with no rational responses). Furthermore, considering any weights leads to the undecidability of the NCNS and NCPS problems. Those results are obtained by reduction from the undecidability of zero-sum multidimensional shortest path games [40, 41]. See details in the long version of this paper [17].

► **Theorem 6.** *With integer weight functions, the Non-Cooperative Nash Synthesis problem and the Non-Cooperative Pareto Synthesis problem are undecidable.*

4 Pareto-Optimality

In this section, we provide the proofs of the upper bounds in Theorem 4. Recall that the environment is here composed of the sole player 1 having t target sets T_i , and his rational responses to a strategy σ_0 announced by player 0 are σ_0 -fixed PO plays. The lower bounds are proved in the long version [17] with reductions from QBF or some of its variants [42]. All those reductions already hold for qualitative reachability games. We thus obtain the same complexity classes as in Theorem 4 for this class of games, as indicated in Table 1.

To solve the two verification problems (NCPV and UNCPV), we first construct the product game³ $\mathcal{G} \times \mathcal{M}_0$ of size polynomial in \mathcal{G} and \mathcal{M}_0 , and we *assume* to directly work with this game, *again denoted* \mathcal{G} . Note that in the product game, when \mathcal{M}_0 is nondeterministic, player 0 is able to play any strategy σ_0 compatible with \mathcal{M}_0 , and when \mathcal{M}_0 is deterministic, the verification problems are simplified as there is a single compatible strategy σ_0 . The complement of the (U)NCPV problem has many similarities with the CPS problem:

► **Problem 7.** *The complement of the (U)NCPV problem (co(U)NCPV) asks whether there exists $\sigma_0 \in \Sigma_0$ and a σ_0 -fixed PO play π such that $\text{cost}_0(\pi) > c$.*

Indeed, the statement is the same except that the inequality $\text{cost}_0(\pi) \leq c$ in the CPS problem is here replaced by $\text{cost}_0(\pi) > c$. To prove the upper bounds of Theorem 4, we thus have to solve the decision problem “do there exist $\sigma_0 \in \Sigma_0$ and a σ_0 -fixed PO play π such that $\text{cost}_0(\pi) \sim c$?” with $\sim \in \{\leq, >\}$. In short, the algorithm to solve the CPS problem and the complement of the (U)NCPV problem proceeds through the following steps:

³ The product of a game with a Mealy machine is recalled in Appendix A.

1. Guess a play π in the form $\pi = \mu(\nu)^\omega$ in polynomial time. The length of the lasso is polynomial or exponential, depending on the studied problem. In the latter case, we will guess a succinct representation of the lasso by using Parikh automata [23, 32].
2. Compute in polynomial time $\text{cost}_{\text{env}}(\pi)$ and verify in polynomial time that $\text{cost}_0(\pi) \sim c$.
3. Verify that player 0 has a strategy σ_0 , with π consistent with σ_0 , that guarantees that $\text{cost}_{\text{env}}(\pi)$ is σ_0 -fixed PO. This last step will be done in coNP or in PSPACE, depending on the studied problem.

Therefore, if a strategy σ_0 exists as in Step 3, the σ_0 -fixed PO play π such that $\text{cost}_0(\pi) \sim c$ is the lasso of Step 1. Let us now provide detailed proofs for these three steps.

4.1 Existence of Lassos

The goal in this section is to prove the next lemma stating that one can always suppose that π is a lasso. For that purpose, we use a classical approach consisting of removing cycles [10, 14, 21].

► **Lemma 8.** *Let $\sigma_0 \in \Sigma_0$ and π be a σ_0 -fixed PO play π such that $\text{cost}_0(\pi) \sim c$. Then there exist $\sigma'_0 \in \Sigma_0$ and a σ'_0 -fixed PO play $\pi' = \mu(\nu)^\omega$ such that $\text{cost}_0(\pi') \sim c$. Moreover, $\text{Visit}(\mu) = \text{Visit}(\mu\nu)$ and*

- *if $\text{cost}_0(\pi) \leq c$, then $|\mu| \leq (t+1)|V|$, $|\nu| \leq |V|$, $\text{cost}_{\text{env}}(\pi') \in \{0, 1, \dots, B, +\infty\}^t$, with $B = (t+2)|V|W$,*
- *if $\text{cost}_0(\pi) > c$, then $|\mu| \leq c + (t+1)|V|$, $|\nu| \leq |V|$, $\text{cost}_{\text{env}}(\pi') \in \{0, 1, \dots, B, +\infty\}^t$, with $B = (c + (t+2)|V|)W$.*

Proof. Let $\pi = \pi_0\pi_1\dots$ be a σ_0 -fixed PO play such that $\text{cost}_0(\pi) \sim c$.

Suppose that $\text{cost}_0(\pi) \leq c$. Consider, along π , any two consecutive first visits to two target sets, say T_i and T_j . If there exists $m < n$ such that $\pi_n = \pi_m$ between these two visits, we remove the cycle $\pi_{[m,n[}$ from π . We repeat this process until there are less than $|V|$ vertices between the two visits, for any such pair T_i, T_j , but also between π_0 and the first visit to a target set. Let us denote π' the resulting play. Consider now along π' the last first visit to a target set, say at index k . We then seek for the first repeated vertex $\pi'_{\ell_1} = \pi'_{\ell_2}$ with $k \leq \ell_1 < \ell_2$ after k . In this way, we obtain $\nu = \pi'_{[\ell_1, \ell_2[}$ with $|\nu| \leq |V|$ and $\mu = \pi'_{[0, \ell_1[}$ with $|\mu| \leq (t+1)|V|$. So, we get the required lasso $\mu(\nu)^\omega$ such that $\text{Visit}(\mu) = \text{Visit}(\mu\nu)$, $\text{cost}_0(\mu(\nu)^\omega) \leq \text{cost}_0(\pi) \leq c$, and $\text{cost}_{\text{env}}(\mu(\nu)^\omega) \in \{0, 1, \dots, B, +\infty\}^t$, with $B = (t+2)|V|W$.

The case $\text{cost}_0(\pi) > c$ is treated similarly, except that we cannot remove cycles along the longest prefix h of π such that $\text{cost}_0(h) \leq c$, as this operation might decrease the cost of player 0. We thus get $|\mu| \leq c + (t+1)|V|$, $\text{cost}_0(\mu(\nu)^\omega) > c$, and $\text{cost}_{\text{env}}(\mu(\nu)^\omega) \in \{0, 1, \dots, B, +\infty\}^t$, with $B = (c + (t+2)|V|)W$.

It remains to explain how to construct a strategy σ'_0 from σ_0 such that $\pi' = \mu(\nu)^\omega$ is σ'_0 -fixed PO. First, σ'_0 is built in a way to produce π' . Second, we have to define σ'_0 outside π' , i.e., from any $h'v$, with $v \in V$, such that h' is prefix of π' but not $h'v$. Let h be such that the elimination of cycles done in π , restricted to h , leads to h' . We then define $\sigma'_0(h'g) = \sigma_0(hg)$ for all histories $g \in \text{Hist}(v)$. Notice that σ'_0 is the required strategy as the elimination of cycles in a history or a play decreases the costs. ◀

► **Example 9.** When $\text{cost}_0(\pi) > c$, Lemma 8 provides a bound on $|\mu\nu|$ that is exponential in the binary encoding of c . In Figure 2, we present a small example of a reachability game showing that this is unavoidable. The initial vertex v_0 is owned by player 1, v_1 is owned by player 0, and there are two weight functions w_0 and w_1 (thus $t = 1$). Both players have the same target set: $T_0 = T_1 = \{v_1\}$. Notice that player 1 is the only one to play, and a play

$\pi \in \text{Plays}(v_0)$ is PO if and only if visits T_1 (and has $\text{cost}_{\text{env}}(\pi) = 0$). Hence, given a threshold c , any PO play π with $\text{cost}_0(\pi) > c$ is equal to $v_0^k(v_1)^\omega$ with $k > c$. The length $|v_0^k v_1|$ is thus greater than c . Therefore, Step 1 of our decision algorithm for the co(U)NCPV cannot guess an explicit representation $\mu(\nu)^\omega$ if we want to stick to a polynomial time algorithm.

4.2 Particular Zero-sum Games

Now that we know we can limit our study to lassos π , Step 3 requires to verify that player 0 has a strategy σ_0 ensuring that $\text{cost}_{\text{env}}(\pi)$ is σ_0 -fixed PO. Before going deeper into this step, we need to study some particular two-player zero-sum games.⁴ Let $\mathcal{A} = (V, E, \mathcal{P}, (V_i)_{i \in \mathcal{P}}, (w_i)_{i \in \{1, \dots, t\}})$ be an arena with $\mathcal{P} = \{\text{Eve}, \text{Adam}\}$ and equipped with t weight functions $w_i : E \rightarrow \mathbb{N}$. We suppose that \mathcal{A} is initialized with $v_0 \in V$. We fix t target sets $T_i \subseteq V$ and t constants $d_i \in \mathbb{N}^{>0} \cup \{+\infty\}$. We denote by $\mathcal{G} = (\mathcal{A}, \Omega)$ a zero-sum game whose *objective* Ω is a Boolean combination of the following objectives:

- $\text{Reach}_{<d_i}(T_i) = \{\pi \in \text{Plays}(v_0) \mid \text{cost}_i(\pi) < d_i\}$ called *bounded reachability objective*, and
- $\text{Safe}_{\geq d_i}(T_i) = \text{Plays}(v_0) \setminus \text{Reach}_{<d_i}(T_i)$ called *bounded safety objective*.

Solving such a game \mathcal{G} means to decide whether *Eve* has a strategy σ such that all plays $\pi \in \text{Plays}(v_0)$ consistent with σ belong to the objective Ω . If such a strategy σ exists, we say that σ is *winning for* Ω and that the initial vertex v_0 is *winning for Eve for* Ω .

For the PO-check required for Step 3, will see in Section 4.3 that we need to solve the zero-sum games stated in the next two propositions, where the constants d_i are encoded in binary. The second proposition will be used in the general case of nondeterministic Mealy machines \mathcal{M}_0 while the first one will be used in the deterministic case. Proposition 10 is a quantitative extension of a result in [24] about (qualitative) generalized reachability games.

► **Proposition 10.** *Let $\mathcal{G} = (\mathcal{A}, \Omega)$ be a zero-sum game with $\Omega = \bigcap_{1 \leq i \leq t} \text{Reach}_{<d_i}(T_i)$ and *Eve* is the only one to play. Deciding whether v_0 is winning for *Eve* is an NP-complete problem.*

Proof. We first notice that if *Eve* has a winning strategy from v_0 , i.e., there exists a play $\pi \in \Omega$, then we can eliminate cycles as in the proof of Lemma 8. Therefore, there exists a lasso $\pi' = \mu(\nu)^\omega \in \Omega$ where $|\mu\nu| \leq (t+2)|V|$. Thus, to get an algorithm in NP, we guess such a lasso π' and verify that $\text{cost}_i(\pi') < d_i$ for each $i \in \{1, \dots, t\}$. This is possible in polynomial time with the costs encoded in binary. It is proved in [24] that solving (qualitative) generalized reachability games with $V_{\text{Adam}} = \emptyset$ is NP-complete. Our problem is thus NP-hard by a reduction from the previous problem with the same arena, the weight functions assigning a null weight to all edges, and by setting $(d_1, \dots, d_t) = (+\infty, \dots, +\infty)$. ◀

The next proposition, of potential independent interest, is easily extended to any positive Boolean combinations of bounded safety objectives.

► **Proposition 11.** *Let $\mathcal{G} = (\mathcal{A}, \Omega)$ be a zero-sum game where $\Omega = \Omega^{(1)} \cup \Omega^{(2)}$, with $\Omega^{(1)} = \left(\bigcap_{1 \leq i \leq t} \text{Safe}_{\geq d_i}(T_i)\right)$ and $\Omega^{(2)} = \left(\bigcup_{1 \leq i \leq t} \text{Safe}_{\geq d_i+1}(T_i)\right)$, and such that $+\infty + 1 = +\infty$. Then, deciding whether v_0 is winning for *Eve* is in PSPACE.*

Proof. We solve the game (\mathcal{A}, Ω) by using a recursive algorithm. To know whether v_0 is winning for *Eve*, we run a depth-first search over a finite tree rooted at v_0 that is the (truncated) unraveling of \mathcal{A} , and we keep track of the accumulated weights along the explored

⁴ We suppose that the reader is familiar with this concept.

branch as a tuple $(c_i)_{1 \leq i \leq t}$, where each c_i is encoded in binary. Each explored branch h will have its leaf decorated by a boolean $f(h) = \perp$ (*Eve* is losing) or $f(h) = \top$ (*Eve* is winning) according to some rules that we describe below. Then the depth-first search algorithm backwardly assigns a boolean to the internal nodes of the tree according to the following rule: for any $hv \in V^*V_{Eve}$, we have $f(hv) = \top$ if there exists $v' \in \text{succ}(v)$ such that $f(hvv') = \top$, otherwise $f(hv) = \perp$, while for any $hv \in V^*V_{Adam}$, we have $f(hv) = \top$ if for all $v' \in \text{succ}(v)$, $f(hvv') = \top$, otherwise $f(hv) = \perp$. To have an algorithm executing in polynomial space, the depth of the tree must be polynomial.

Along a branch, the rules are the following to stop the exploration (the objective Ω may be modified during the exploration):

- If for some i , the current weight c_i is such that $c_i \geq d_i + 1$ and T_i was not visited, then we can stop the branch h and set $f(h) = \top$. Indeed, $\Omega^{(2)}$ is satisfied, and thus also Ω .
- If for some i , we have $c_i < d_i$ while visiting T_i , then $\Omega^{(1)}$ is not satisfiable anymore, and we continue the exploration with the sole objective $\Omega^{(2)}$ where the i -th objective $\text{Safe}_{\geq d_i+1}(T_i)$ being ignored (as it is not satisfied).
- If for some i , we have $c_i = d_i$ while visiting T_i , then we continue the exploration with Ω such that $\text{Safe}_{\geq d_i}(T_i)$ is removed from $\Omega^{(1)}$ (as it is satisfied) and $\text{Safe}_{\geq d_i+1}(T_i)$ is removed from $\Omega^{(2)}$ (as it is not satisfied).
- If $\Omega^{(1)}$ becomes an empty intersection, then we stop the branch h and set $f(h) = \top$.
- If $\Omega^{(1)}$ has been removed from Ω (because it was not satisfiable anymore) and $\Omega^{(2)}$ becomes an empty union, then we stop the branch h and set $f(h) = \perp$.
- There is one more case to stop the branch h : when some vertex v is visited twice, i.e., $h = gv'g'v$ for some $g, g' \in V^*$. Then we stop the branch and set $f(h) = \top$. Indeed, we stand in a better situation in $gv'g'v$ than in gv concerning the accumulated weights, as we consider bounded safety objectives.

The last case happens as soon as the explored branch has length $|V| + 1$ and the other cases do not occur. Therefore, as there are t bounded safety objectives in both $\Omega^{(1)}$ and $\Omega^{(2)}$, any branch has a length polynomially bounded by $t|V|$. Moreover, the accumulated weights c_i are all bounded by $t|V|W$, thus stored in a polynomial space when encoded in binary. We can thus decide in polynomial space whether v_0 is winning for *Eve* for Ω . ◀

4.3 Pareto-Optimality Check

Let us come back to our reachability games. We can now solve Step 3 where given a lasso π with $\text{cost}_{\text{env}}(\pi) \in \{0, 1, \dots, B, +\infty\}^t$ (by Lemma 8), we want to verify whether player 0 has a strategy σ_0 guaranteeing that $\text{cost}_{\text{env}}(\pi)$ is σ_0 -fixed PO. If player 1 is the only one to play in the game, it reduces to verify that $\text{cost}_{\text{env}}(\pi)$ is PO. The latter problem is in coNP as stated in the next lemma, while the former is in PSPACE as stated in Lemma 13.

► **Lemma 12.** *Suppose that player 1 is the only one to play. Deciding whether a given cost $p \in \{0, 1, \dots, B, +\infty\}^t$ is PO is in coNP.*

Proof. The cost p is *not* PO if there exists a play $\pi' \in \text{Plays}(v_0)$ such that $\text{cost}_i(\pi') \leq p_i$ for all $i \in \{1, \dots, t\}$ and $\text{cost}_j(\pi') < p_j$ for some $j \in \{1, \dots, t\}$. That is, if for some j , there exists a play $\pi' \in \Omega^{(j)} = \bigcap_{i \neq j} \text{Reach}_{< p_i+1}(T_i) \cap \text{Reach}_{< p_j}(T_j)$. Solving the zero-sum game (\mathcal{A}, Ω) is in NP by Proposition 10. This concludes the proof. ◀

► **Lemma 13.** *Given $p = \text{cost}_{\text{env}}(\pi) \in \{0, 1, \dots, B, +\infty\}^t$ being the cost of a play π , deciding whether player 0 has a strategy σ_0 ensuring that p is σ_0 -fixed PO is in PSPACE.*

Proof. To prove the lemma, we first fix a prefix h of π , with $v \in V$, such that hv is not a prefix of π (hv is called a *deviation*), and we study the zero-sum game $(\mathcal{A}, \Omega^{(hv)})$ with the objective $\Omega^{(hv)}$ equal to $\{\pi' \in \text{Plays}(v) \mid \neg(\text{cost}_{\text{env}}(h\pi') < p)\}$. Let us show that deciding whether v is winning for player 0 for $\Omega^{(hv)}$ is in PSPACE. Notice that for each $i \in \{1, \dots, t\}$ such that h does not visit T_i , we have, with $q_i = w_i(hv)$ and $+\infty - q_i = +\infty$: $\text{cost}_i(h\pi') < p_i$ if and only if $\text{cost}_i(\pi') < p_i - q_i$. Let us rewrite the condition $\neg(p' < p)$ with $p, p' \in \mathbb{N}^t$ as follows: $(\forall i \in \{1, \dots, t\} p'_i \geq p_i) \vee (\exists i \in \{1, \dots, t\} p'_i > p_i)$. Hence, the objective $\Omega^{(hv)}$ can be rewritten as $\left(\bigcap_{\text{Occ}(h) \cap T_i = \emptyset} \text{Safe}_{\geq p_i - q_i}(T_i) \right) \cup \left(\bigcup_{\text{Occ}(h) \cap T_i = \emptyset} \text{Safe}_{\geq p_i - q_i + 1}(T_i) \right)$.

By Proposition 11, given the constants p_i and q_i , we can check whether v is winning for player 0 in polynomial space. Notice that each q_i can be computed in polynomial space by accumulating the weights, with respect to w_i , as long as T_i is not visited (as $q_i \leq p_i$).

Second, given two deviations $hv, h'v$ ending with the same vertex v and such that h is prefix of h' , if $\text{Visit}(h') = \text{Visit}(h)$ and v is winning for $\Omega^{(hv)}$, then v is also winning for $\Omega^{(h'v)}$ (with the same strategy). Indeed, the constants q'_i for $h'v$ are greater than the constants q_i for hv . We are thus in a “better situation” than in $\Omega^{(hv)}$. So, it suffices to consider polynomially many deviations hv , as π can visit at most t target sets and there are at most $|V|$ vertices v .

Finally, deciding whether player 0 has a strategy σ_0 ensuring that p is σ_0 -fixed PO amounts to solving the zero-sum games $(\mathcal{A}, \Omega^{(hv)})$ for polynomially many deviations hv . If player 0 has a winning strategy σ_{hv} for all those games, the required strategy σ_0 is defined as $\sigma_0(g) = \sigma_{hv}(vg')$ for all histories g such that $g = hv g'$ with the longest prefix h of π . ◀

4.4 Upper Bounds

We are now ready to prove the upper bounds in Theorem 4 by providing the announced algorithms for Steps 1-3. The proof is divided according to the considered problem. We need to recall [23] that a Parikh automaton is a nondeterministic finite automaton (NFA) over an alphabet Σ and whose transitions are weighted by tuples in \mathbb{N}^k , together with a semilinear set $\mathbf{C} \subseteq \mathbb{N}^k$. It accepts a word $w \in \Sigma^*$ if there exists a run on w ending on an accepting state such that the sum of all encountered weight tuples belongs to \mathbf{C} . The non-emptiness problem for Parikh automata is NP-complete for numbers encoded in binary [23].

Proof of the upper bounds in Theorem 4. We begin with the CPS problem (Theorem 4.c). Let us give an algorithm in PSPACE that decides whether there exist $\sigma_0 \in \Sigma_0$ and a σ_0 -fixed PO play π such that $\text{cost}_0(\pi) \leq c$. By Lemma 8, we guess a lasso $\pi = \mu(\nu)^\omega$ with $|\mu\nu| \leq (t+2)|V|$, in time polynomial in $|V|$ and t . Then, we compute $p = \text{cost}_{\text{env}}(\pi)$ and $\text{cost}_0(\pi)$ and check whether $\text{cost}_0(\pi) \leq c$. This can be done in time polynomial in $t, |V|$, and the binary encoding of W and c by Lemma 8. Finally, by Lemma 13, we verify in polynomial space whether player 0 has a strategy σ_0 ensuring that p is σ_0 -fixed PO.

For the NCPV problem (Theorem 4.a), recall that we consider its complementary coNCPV problem (see Problem 7), and that player 1 is the only one to play. We begin by giving an algorithm in NP for Step 1 and 2. Lemma 8 does not provide a polynomial bound on the length of the lasso $\pi = \mu(\nu)^\omega$ due to the threshold c given in binary. However, we will guess a succinct representation of π by using Parikh automata.

The idea is the following one. Along the prefix μ of the lasso π , some target sets T_{k_1}, \dots, T_{k_n} are visited, with $n \leq t$, such that the first visits are in vertices $\pi_{\ell_1}, \dots, \pi_{\ell_n}$ with $\ell_1 < \dots < \ell_n$. And after μ , no more target sets are visited along $\mu\nu$ (see Lemma 8). We start by guessing a sequence $v_0, v_1, \dots, v_n, v_{n+1}$ of vertices, called *markers*, with the aim that v_0 is the initial vertex, $v_i = \pi_{\ell_i}$, $1 \leq i \leq n$, and $v_{n+1} = \text{first}(\nu)$. By Lemma 8, we

know that $\text{cost}_{\text{env}}(\pi) \in \{0, 1, \dots, B, +\infty\}^t$, where $B = (c + (t + 2)|V|)W$. We thus guess a tuple $(p_0, p_1, \dots, p_t) \in \{0, 1, \dots, B, +\infty\}^t$ with the aim that $(p_1, \dots, p_t) = \text{cost}_{\text{env}}(\mu)$ and $p_0 = w_0(\mu)$. We also guess for each *portion* $\pi_{[\ell_i, \ell_{i+1}]}$, $i \leq n$,

- a weight $q_0^{(i)} \in \{0, 1, \dots, B\}$ for player 0 with the aim that $q_0^{(i)} = w_0(\pi_{[\ell_i, \ell_{i+1}]})$ and $w_0(\mu) = p_0 = \sum_i q_0^{(i)}$,
- a “useful” environment weight tuple, i.e., for all $j \in \{1, \dots, t\}$, a weight $q_j^{(i)} \in \{0, 1, \dots, B\}$ such that $\pi_{[0, \ell_i]}$ does not visit T_j , with the aim that $q_j^{(i)} = w_j(\pi_{[\ell_i, \ell_{i+1}]})$ and $\text{cost}_j(\mu) = p_j = \sum_i q_j^{(i)}$.⁵

We can guess in polynomial time the sequence $v_0, v_1, \dots, v_n, v_{n+1}$ and the constants $p_j, q_j^{(i)}$ encoded in binary, as $n \leq t$ and $B = (c + (t + 2)|V|)W$. We then check in polynomial time that v_0 is the initial state, that each v_i belongs to a distinct target set T_{k_i} , $1 \leq i \leq n$, that $p_j = \sum_i q_j^{(i)}$ for each j , and that $p_0 > c$ for the given threshold c .⁶

It remains to check the existence of polynomially many paths:

- For each $i \leq n$, the existence of a path $\rho^{(i)}$ from v_i to v_{i+1} on a subgraph of \mathcal{A} restricted to some sets $V^{(i)}$ and $E^{(i)}$ of vertices and edges respectively, and to some weight functions, such that $w_j(\rho^{(i)}) = q_j^{(i)}$ for all j .
- The existence of a path from v_{n+1} to itself (the cycle ν) that visits no new target set with respect to T_{k_i} , $1 \leq i \leq n$.

The first check can be done thanks to Parikh automata : one can decide in NP the existence of a path in a subgraph of \mathcal{A} between two given vertices and with a given weight tuple \bar{q} (the subgraph is seen as a Parikh automaton with $\Sigma = \{\#\}$ and $\mathbf{C} = \{\bar{q}\}$).⁷ The set $V^{(i)}$ is defined as $V \setminus \left(\bigcup_{j>i+1} T_{k_j} \cup \bigcup_{p_j=+\infty} T_j \right)$, and the set $E^{(i)}$ as $(E \cap V^{(i)} \times V^{(i)}) \setminus \{(v, v') \mid v \in T_{k_{i+1}}\}$. Indeed, for the portion $\pi_{[\ell_i, \ell_{i+1}]}$, we do not allow to prematurely visit a target set T_{k_j} , $j \geq i + 1$, except $v_{i+1} \in T_{k_{i+1}}$, and there are target sets that we do not want to visit at all. We also remove the weight function w_{k_j} with $j \in \{1, \dots, i\}$. The second check can be done thanks to classical automata, by restricting the set of vertices to $V \setminus \left(\bigcup_{p_j=+\infty} T_j \right)$. To show that the coNCPV problem is in Σ_2^P , in the previous algorithm in NP that guesses a lasso π with $\text{cost}_{\text{env}}(\pi) = p$, we add an oracle in coNP to check whether p is a PO cost thanks to Lemma 12. As $\text{NP}^{\text{coNP}} = \Sigma_2^P$, we get that the NCPV problem is in Π_2^P .

It remains to show that the coUNCPV problem is in PSPACE to get the upper bound of Theorem 4.b). The approach is to guess a cost $p \in \{0, \dots, B, +\infty\}^t$ and a length ℓ for the exponential lasso π of Lemma 8, whose both encodings in binary use a polynomial space. We guess π vertex by vertex, by only storing the current edge (u, u') , the current accumulated weight (c_0, c_1, \dots, c_t) on each dimension, and which target sets T_i have already been visited. At any time, the stored information uses a polynomial space. At each guess, we apply the reasoning of Lemma 13 to check in polynomial space whether player 0 can ensure that p is a PO cost from each vertex $v \neq u'$ successor of u (i.e., from any deviation of π). We also check that for each first visit to a target set T_i , we have $c_i = p_i$ if $i \in \{1, \dots, t\}$, and $c_i > c$ if $i = 0$. At each guess, a counter is incremented until reaching the length ℓ , where we stop guessing π and finally check whether $p_i = +\infty$ for each T_i that has not been visited.

This completes the proof as Theorem 4.d is established in [11]. ◀

⁵ If $\pi_{[0, \ell_i]}$ visits T_j , then $\text{cost}_j(\pi)$ is already known as $\text{cost}_j(\pi) = \text{cost}_j(\pi_{[0, \ell_i]})$.

⁶ To keep the proof readable, we assume that each v_i belongs to one target set T_{k_i} . In general, it could belong to several target sets. The proof is easily adapted by considering the union of target sets.

⁷ We do not need to use an oracle here. It suffices to plug the NP algorithm for Parikh automata in ours as if the required path exists, our algorithm will find it in polynomial time.

5 Nash Equilibria

We now discuss the proofs of Theorem 5. The environment is here composed of t players whose rational responses to a strategy σ_0 of player 0 are σ_0 -fixed NE outcomes.

The upper bounds for (U)NCNV and CNS problems given in Theorem 5.a-c are proved with the same approach as for Pareto optimality, limited to Steps 1-2. There is no need for Step 3, thanks to a well-known characterization of NE outcomes (based on the values of some two-player zero-sum games, see e.g. [10, 16] or the long version of this paper [17]) that is directly checked on the lasso guessed in Step 1. We need again Parikh automata to guess a succinct representation of the lasso. The lower bounds for those problems were already known for qualitative reachability games [27]. See the long version [17].

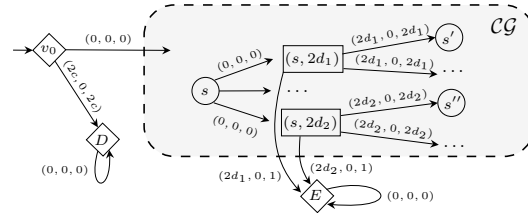
We thus focus on the NCNS problem (Theorem 5.d). We prove below that this problem is EXPTIME-hard, already for two-player environments. The decidability is left open. This decision problem is a real challenge that cannot be solved by known approaches. Indeed, the technique of tree automata, as used in [21] to show the decidability of several ω -regular objectives, is not applicable in the context of quantitative reachability. This is because, while in the scenario of qualitative reachability, the costs are Boolean and can be encoded within the finite state space of a tree automaton, for quantitative reachability, these costs are now integers that are not bounded and vary according to the strategy σ_0 that is being synthesized. Consequently, it is not feasible to directly encode constraints within the states of the automaton in this latter case. Additionally, there is a necessity to enforce constraints related to subtrees, such as comparing (unbounded) costs between two subtrees. Generally, incorporating the capability to enforce subtree constraints in tree automata results in undecidability, with only certain subclasses having a decidable emptiness problem, see e.g. [3]. Therefore, addressing the general case would necessitate either advancements in the field of automata theory or an entirely new methodological approach.

However, we are able to solve the practically relevant case of one-player environments for which we prove that the NCNS problem is PSPACE-hard and in EXPTIME in the long version of this paper [17]. The PSPACE-hardness is given by a classical reduction from the subset-sum game problem [43]. The intuition for the EXPTIME-membership is the following: it consists in finding a play π where $\text{cost}_0(\pi) \leq c$ such that when the only component of the environment deviates from π , either the system inflicts to the deviating play π' a cost for the environment such that $\text{cost}_1(\pi') > \text{cost}_1(\pi)$ meaning that deviating is not profitable, or it ensures a cost for himself such that $\text{cost}_0(\pi') \leq c$. Note that this approach only works for one-player environments.

We are also able to solve the NCNS problem for any number of players in the environment, for the variant where the rational NE responses of the environment aim to ensure costs bounded by a given threshold rather than minimizing these costs (this is also arguably an interesting model of rationality in practice). This is a perspective studied in [39] in the case of NEs for discounted-sum objectives. We show in the long version [17] that this variant is EXPTIME-complete.

► **Theorem 14.** *The Non-Cooperative Nash Synthesis problem where the objective of each player $i \in \{1, \dots, t\}$ is a bounded reachability objective $\text{Reach}_{<d_i}(T_i)$ is EXPTIME-complete, and hardness holds even with a one-player environment.*

Reduction for Two-Player Environments. We finally prove that the NCNS problem is EXPTIME-hard, already for a two-player environment (lower bound of Theorem 5.d). The reduction is given from the *countdown game problem*, known to be EXPTIME-complete [31].



■ **Figure 3** Reduction from the countdown game problem to the NCNS problem (two-player env.).

Given a threshold $c \in \mathbb{N}$, a countdown game \mathcal{CG} is a two-player zero-sum game played on a directed graph (V, E) where $E \subseteq V \times \mathbb{N}^{>0} \times V$. A configuration is a pair $(s, k) \in V \times \mathbb{N}$, initially $(s_0, 0)$ with s_0 an initial vertex, from where player 0 chooses $d \in \mathbb{N}^{>0}$ such that there exists $(s, d, s') \in E$ (we assume that such a d always exists). Player 1 then chooses such an $s' \in V$ to reach the configuration $(s', k + d)$. When reaching a configuration (s, k) with $k \geq c$, the game stops and player 0 wins if and only if $k = c$.⁸ Player 0 wins the game \mathcal{CG} if he has a strategy σ_0 from s_0 that allows him to reach some configuration (s, c) , whatever the strategy of player 1.

► **Theorem 15.** *The Non-Cooperative Nash Synthesis problem with a two-player environment is EXPTIME-hard.*

Proof. Given a countdown game \mathcal{CG} and a threshold c , we build a reachability game \mathcal{G} as depicted in Figure 3 with three players, player 0 (owning the circle vertices of \mathcal{CG}), player 1 (owning the square vertices of \mathcal{CG}), and player 2 (owning the initial vertex v_0 and vertices D, E). The three weight functions are indicated on the edges, with a null weight on all edges for player 1. The initial vertex v_0 has two outgoing edges, one towards vertex D and the other one to the initial vertex s_0 of \mathcal{CG} . Inside \mathcal{CG} , players 0 and 1 are simulating the countdown game. The target sets are $T_0 = T_2 = \{D, E\}$ and $T_1 = V$. Thus, for any play, player 1 gets a cost of 0 and will never have the incentive to deviate from his strategy. The \mathcal{CG} part of the figure contains a slight modification of the given countdown: players 0 and 1 act as in \mathcal{CG} , player 1 can exit it by taking the edge to vertex E , the weights d are multiplied by 2. More precisely, player 0 first selects a transition from a vertex s to some vertex $(s, 2d)$, with $d \in \mathbb{N}^{>0}$, then player 1 responds with a successor s' such that (s, d, s') is an edge in the initial countdown game. At any point $(s, 2d)$, player 1 can exit the \mathcal{CG} by going to E , adding $2d$ to the cost of player 0 and 1 to the cost of player 2, i.e., it gives the cost tuple $(2k + 2d, 0, 2k + 1)$ where $2k$ is the accumulated weight inside \mathcal{CG} before exiting it.

Let us show that a strategy $\sigma_0 \in \Sigma_0$ is a solution to the NCNS problem with the threshold $2c$ if and only if it is winning in the given countdown game and threshold c . We first suppose that σ_0 is a winning strategy for player 0 in the countdown game. We consider this strategy in \mathcal{G} and enumerate all possible plays consistent with σ_0 :

- The play $v_0(D)^\omega$ gives the cost $2c$ to player 0, thus satisfying the threshold $2c$,
- No play staying infinitely often in \mathcal{CG} is the outcome of a σ_0 -fixed NE, as it gives an infinite cost to player 2 while player 2 could deviate in v_0 to get a cost of $2c < +\infty$,
- Any play π ultimately reaching E has $\text{cost}_0(\pi) = 2k + 2d$ and $\text{cost}_2(\pi) = 2k + 1$, for some $k \in \mathbb{N}$. If $2k + 2d \leq 2c$, then $\text{cost}_0(\pi)$ satisfies the threshold constraint. Otherwise, $2k + 2d > 2c$, but as σ_0 is winning in the initial countdown game, this means that there was a previous configuration where the costs of both players 0 and 2 were exactly $2c$. This means that $\text{cost}_2(\pi) = 2k + 1 \geq 2c + 1$, i.e., π is again not a σ_0 -fixed NE outcome.

⁸ Classically, the initial configuration is (s_0, c) and the accumulated weight k decreases until being ≤ 0 .

Assume now that σ_0 is not winning in the countdown game. Hence, there exists a losing play consistent with σ_0 in this game, that leads to a play π in the grey part of Figure 3 such that in none of its vertices, the accumulated weight is exactly $2c$, i.e., there are two consecutive steps where the accumulated weight is $2k < 2c$ and then $2k + 2d > 2c$. So, player 1 can exit between these two steps to reach E . The resulting play π' has $\text{cost}_0(\pi') = 2k + 2d > 2c$ and $\text{cost}_2(\pi') = 2k + 1 < 2c + 1$, thus $\text{cost}_2(\pi') < 2c$. Consequently, π' is a σ_0 -fixed NE outcome but $\text{cost}_0(\pi') > 2c$. It follows that σ_0 is not a solution to the NCNS problem. ◀

6 Conclusion

In this paper, we have determined the exact complexity class for several rational verification and synthesis problems in quantitative reachability games, considering both NE and PO rational behaviors of the environment. However, for the NCNS problem, while we have solved the important one-player environment case, we have left open the multi-player environment case. We believe this latter case poses a significant challenge that may require new advances in automata techniques to be solved.

There are several interesting future works to investigate. (1) We intend to study the FPT complexity of the studied problems. Notice that some of our lower bounds results already hold for one-player environments (see the CNS and UNCNV problems in Section 5). (2) Instead of one reachability objective, player 0 could have several ones and a threshold on these objectives that he wants to see satisfied. (3) The concept of NE could be replaced by SPE or by strong NE (that allows collaborations between the players during deviations). Still, it is important to note that strategies σ_0 that are solutions to the non-cooperative synthesis problems under NE rationality are also solutions under SPE (resp. strong NE) rationality, as SPEs (resp. strong NEs) constitute a subset of NEs.

References

- 1 Shaull Almagor, Orna Kupferman, and Giuseppe Perelli. Synthesis of controllable nash equilibria in quantitative objective game. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 35–41. ijcai.org, 2018. doi:10.24963/IJCAI.2018/5.
- 2 Rajeev Alur, Aldric Degorre, Oded Maler, and Gera Weiss. On Omega-Languages Defined by Mean-Payoff Conditions. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2009. doi:10.1007/978-3-642-00596-1_24.
- 3 Luis Barguñó, Carles Creus, Guillem Godoy, Florent Jacquemard, and Camille Vacher. The Emptiness Problem for Tree Automata with Global Constraints. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 263–272. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.28.
- 4 Romain Brenguier and Jean-François Raskin. Pareto Curves of Multidimensional Mean-Payoff Games. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification – 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, volume 9207 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2015. doi:10.1007/978-3-319-21668-3_15.
- 5 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Subgame-Perfect Equilibria in Mean-Payoff Games. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 8:1–8:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.8.

- 6 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. The Complexity of SPEs in Mean-Payoff Games. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 116:1–116:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.116.
- 7 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Rational Verification for Nash and Subgame-Perfect Equilibria in Graph Games. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPICs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.MFCS.2023.26.
- 8 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, and Jean-François Raskin. Constrained existence problem for weak subgame perfect equilibria with ω -regular Boolean objectives. *Inf. Comput.*, 278:104594, 2021. doi:10.1016/J.IC.2020.104594.
- 9 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie van den Bogaard. The Complexity of Subgame Perfect Equilibria in Quantitative Reachability Games. In Wan J. Fokink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 13:1–13:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.13.
- 10 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, and Nathan Thomasset. On relevant equilibria in reachability games. *J. Comput. Syst. Sci.*, 119:211–230, 2021. doi:10.1016/J.JCSS.2021.02.009.
- 11 Thomas Brihaye, Véronique Bruyère, and Gaspard Reghem. Quantitative Reachability Stackelberg-Pareto Synthesis is NEXPTIME-Complete. In Olivier Bournez, Enrico Formenti, and Igor Potapov, editors, *Reachability Problems – 17th International Conference, RP 2023, Nice, France, October 11-13, 2023, Proceedings*, volume 14235 of *Lecture Notes in Computer Science*, pages 70–84. Springer, 2023. doi:10.1007/978-3-031-45286-4_6.
- 12 Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer Cost Games with Simple Nash Equilibria. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science, International Symposium, LFCS 2013, San Diego, CA, USA, January 6-8, 2013. Proceedings*, volume 7734 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2013. doi:10.1007/978-3-642-35722-0_5.
- 13 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. To Reach or not to Reach? Efficient Algorithms for Total-Payoff Games. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1-4, 2015*, volume 42 of *LIPICs*, pages 297–310. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.CONCUR.2015.297.
- 14 Thomas Brihaye and Aline Goeminne. Multi-weighted Reachability Games. In Olivier Bournez, Enrico Formenti, and Igor Potapov, editors, *Reachability Problems – 17th International Conference, RP 2023, Nice, France, October 11-13, 2023, Proceedings*, volume 14235 of *Lecture Notes in Computer Science*, pages 85–97. Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-45286-4_7.
- 15 Thomas Brihaye, Aline Goeminne, James C. A. Main, and Mickael Randour. Reachability Games and Friends: A Journey Through the Lens of Memory and Complexity (Invited Talk). In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIIT Hyderabad, Telangana, India*, volume 284 of *LIPICs*, pages 1:1–1:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.FSTTCS.2023.1.
- 16 Véronique Bruyère. Synthesis of Equilibria in Infinite-Duration Games on Graphs. *ACM SIGLOG News*, 8(2):4–29, May 2021. doi:10.1145/3467001.3467003.
- 17 Véronique Bruyère, Christophe Grandmont, and Jean-François Raskin. As soon as possible but rationally. *CoRR*, abs/2403.00399, 2024. doi:10.48550/arXiv.2403.00399.

- 18 Véronique Bruyère, Jean-François Raskin, and Clément Tamines. Stackelberg-Pareto Synthesis. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 27:1–27:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.27.
- 19 Véronique Bruyère, Jean-François Raskin, and Clément Tamines. Pareto-Rational Verification. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPICs*, pages 33:1–33:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CONCUR.2022.33.
- 20 Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized Mean-payoff and Energy Games. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPICs*, pages 505–516. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICs.FSTTCS.2010.505.
- 21 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The Complexity of Rational Synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 121:1–121:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.121.
- 22 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, June 1979. doi:10.1007/BF01768705.
- 23 Diego Figueira and Leonid Libkin. Path Logics for Querying Graphs: Combining Expressiveness and Efficiency. In *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, Kyoto, Japan, July 2015. IEEE. doi:10.1109/LICS.2015.39.
- 24 Nathanaël Fijalkow and Florian Horn. Les jeux d’accessibilité généralisée. *Tech. Sci. Informatiques*, 32(9-10):931–949, 2013. doi:10.3166/TSI.32.931-949.
- 25 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The Adversarial Stackelberg Value in Quantitative Games. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 127:1–127:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ICALP.2020.127.
- 26 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational Synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.
- 27 Christophe Grandmont. Rational Synthesis and Verification in Multiplayer Reachability Games Played on Graphs. Master’s thesis, UMONS, June 2023.
- 28 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020. doi:10.1016/J.ARTINT.2020.103353.
- 29 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. On the complexity of rational verification. *Ann. Math. Artif. Intell.*, 91(4):409–430, 2023. doi:10.1007/S10472-022-09804-3.
- 30 David Hyland, Julian Gutierrez, Shankaranarayanan Krishna, and Michael J. Wooldridge. Rational verification with quantitative probabilistic goals. In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum, editors, *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, pages 871–879. ACM, 2024. doi:10.5555/3635637.3662941.

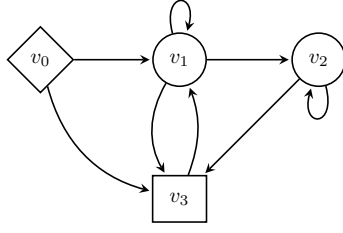
- 31 Marcin Jurdzinski, Francois Laroussinie, and Jeremy Sproston. Model Checking Probabilistic Timed Automata with One or Two Clocks. *Logical Methods in Computer Science*, Volume 4, Issue 3, September 2008. doi:10.2168/LMCS-4(3:12)2008.
- 32 Felix Klaedtke and Harald Rueß. Monadic Second-Order Logics with Cardinalities. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, pages 681–696, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- 33 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with Rational Environments. In Nils Bulling, editor, *Multi-Agent Systems – 12th European Conference, EUMAS 2014, Prague, Czech Republic, December 18-19, 2014, Revised Selected Papers*, volume 8953 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2014. doi:10.1007/978-3-319-17130-2_15.
- 34 Orna Kupferman and Noam Shenwald. The Complexity of LTL Rational Synthesis. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 25–45, Cham, 2022. Springer International Publishing.
- 35 John F. Nash. Equilibrium points in n -person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950. doi:10.1073/pnas.36.1.48.
- 36 Martin J. Osborne. *An introduction to game theory*. Oxford Univ. Press, 2004.
- 37 Christos H. Papadimitriou and Mihalis Yannakakis. On the Approximability of Trade-offs and Optimal Access of Web Sources. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 86–92. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892068.
- 38 Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.
- 39 Senthil Rajasekaran, Suguman Bansal, and Moshe Y. Vardi. Multi-Agent Systems with Quantitative Satisficing Goals. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 280–288. ijcai.org, 2023. doi:10.24963/IJCAI.2023/32.
- 40 Mickael Randour. Games with multiple objectives. In Nathanaël Fijalkow, editor, *Games on Graphs*. Online, 2023. doi:10.48550/arxiv.2305.10546.
- 41 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods Syst. Des.*, 50(2-3):207–248, 2017. doi:10.1007/S10703-016-0262-7.
- 42 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976. doi:10.1016/0304-3975(76)90061-X.
- 43 Stephen Travers. The complexity of membership problems for circuits over sets of integers. *Theoretical Computer Science*, 369(1):211–229, 2006. doi:10.1016/j.tcs.2006.08.017.

A Example of a Nondeterministic Mealy Machine and Product Game

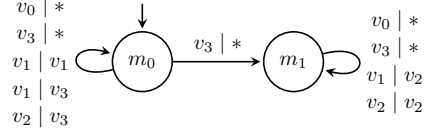
We first provide an example of a nondeterministic Mealy machine and the way it encodes strategies.

► **Example 16.** Consider the arena in Figure 4 and the nondeterministic Mealy machine \mathcal{M}_0 of player 0 illustrated in Figure 5, formally defined as $\mathcal{M}_0 = (M, m_0, \delta, \tau)$ such that

- $M = \{m_0, m_1\}$,
- $\delta(m_0, v_3) = \{m_0, m_1\}$ and $\delta(m, v) = \{m\}$ for every $(m, v) \neq (m_0, v_3)$,
- $\tau(m_0, v) = \begin{cases} \{v_1, v_3\} & \text{if } v = v_1 \\ \{v_3\} & \text{if } v = v_2 \end{cases}$, and $\tau(m_1, v) = \{v_2\}$ if $v = v_1$ or $v = v_2$.



■ **Figure 4** An arena with player 0, player \square , and player \diamond , with no weight displayed.



■ **Figure 5** A nondeterministic Mealy machine of player 0. The notation $v \mid v'$ on the transitions (m, m') indicates that $m' \in \delta(m, v)$, and if $v \in V_0$, that $v' \in \tau(m, v)$, otherwise $v' = *$.

The idea is to start and stay in the memory state m_0 and then, once v_3 has been visited, to nondeterministically switch to the memory state m_1 , or continue staying in the memory state m_0 . The memory state defines which edge player 0 is able to choose from v_1 : either a nondeterministic choice between v_1 and v_3 in m_0 , or v_2 in m_1 .

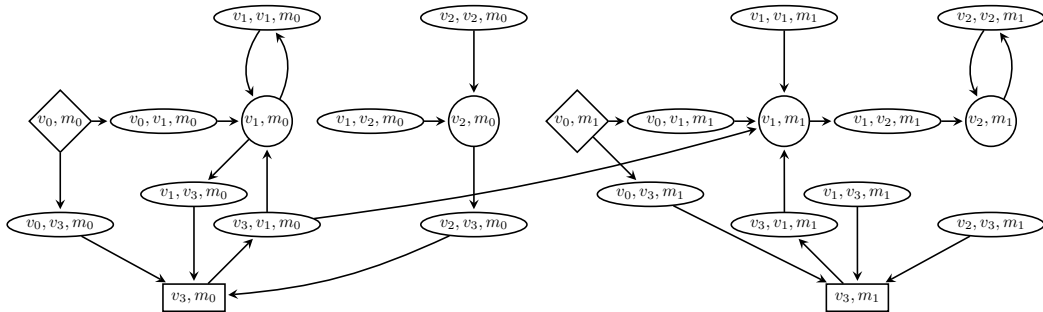
We now formally define the notion of product arena. Let $\mathcal{A} = (V, E, \mathcal{P}, (V_i)_{i \in \mathcal{P}}, (w_i)_{i \in \mathcal{P}})$ be a weighted arena and $\mathcal{M}_j = (M, m_0, \delta, \tau)$ be a (nondeterministic) Mealy machine for player $j \in \mathcal{P}$. Then, the *product arena* $\mathcal{A} \times \mathcal{M}_j$ is the weighted arena $\mathcal{A} \times \mathcal{M}_j = (V', E', \mathcal{P}, (V'_i)_{i \in \mathcal{P}}, (w'_i)_{i \in \mathcal{P}})$ where

- $V' = (V \times M) \cup (V \times V \times M)$,
- $V'_i = V_i \times M$ for all $i \in \mathcal{P} \setminus \{j\}$, and $V'_j = (V_j \times M) \cup (V \times V \times M)$,
- E' is the set of edges defined as
 - $(v, m) \rightarrow (v, v', m)$ if $(v, v') \in E$, and when $v \in V_j$, it must hold that $v' \in \tau(m, v)$,
 - $(v, v', m) \rightarrow (v', m')$ if $m' \in \delta(m, v)$,
- For the edges $e' \in E'$ of the form $(v, m) \rightarrow (v, v', m)$, $w'_i(e') = w_i((v, v'))$, while for the edges e' of the form $(v, v', m) \rightarrow (v', m')$, $w'_i(e') = 0$, for all players $i \in \mathcal{P}$.

Intuitively, in vertices (v, v', m) , it is player j who decides how to update the memory state m according to δ .

When \mathcal{A} is initialized with v_0 as initial vertex, then the product arena is also initialized with (v_0, m_0) as initial vertex. Given a reachability game $\mathcal{G} = (\mathcal{A}, (T_i)_{i \in \mathcal{P}})$, we also define the *product game* $\mathcal{G} \times \mathcal{M}_j$ as the reachability game $(\mathcal{A} \times \mathcal{M}_j, (T'_i)_{i \in \mathcal{P}})$ such that $T'_i = T_i \times M$ for all $i \in \mathcal{P}$.

Back to Example 16, the product arena $\mathcal{A}' = \mathcal{A} \times \mathcal{M}_0$ is depicted in Figure 6. We can see that player 0 has several strategies $\sigma_0 \in \llbracket \mathcal{M}_0 \rrbracket$ whose behavior changes according to the memory state m_0 or m_1 .



■ **Figure 6** The product arena of the arena in Figure 4 and the Mealy machine in Figure 5.


RobTL: Robustness Temporal Logic for CPS

Valentina Castiglioni ✉ 

Eindhoven University of Technology, The Netherlands

Michele Loreti ✉ 

University of Camerino, Italy

Simone Tini ✉ 

University of Insubria, Italy

Abstract

We propose *Robustness Temporal Logic (RobTL)*, a novel temporal logic for the specification and analysis of distances between the behaviours of Cyber-Physical Systems (CPS) over a finite time horizon. RobTL specifications allow us to *measure* the differences in the behaviours of systems with respect to various objectives and temporal constraints, and to study how those differences *evolve in time*. Specifically, the unique features of RobTL allow us to specify *robustness properties* of CPS against uncertainty and perturbations. As an example, we use RobTL to analyse the robustness of an engine system that is subject to attacks aimed at inflicting overstress of equipment.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Modal and temporal logics

Keywords and phrases Cyber-physical systems, robustness, temporal logic, uncertainty

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.15

Supplementary Material *Software*: <https://github.com/quasylab/jspear/tree/working> [9]
archived at `swh:1:dir:ddf418d5a080b8e83323a1b2c38d9f7065e2554`

Funding *Simone Tini*: This study received funding from the European Union – Next-GenerationEU – National Recovery and Resilience Plan (NRRP) – MISSION 4 COMPONENT 2, INVESTMENT N. 1.1, CALL PRIN 2022 D.D. 104 02-02-2022 – MEDICA Project, CUP N. J53D23007180006.

This publication is part of the project *NODES* which has received funding from the MUR – M4C2 1.5 of PNRR with grant agreement no. ECS00000036.

1 Introduction

When systems are subject to *uncertainty* and *perturbations*, like Cyber-Physical Systems (CPS) [35] in which software components, or *agents*, must interact with an unpredictable *environment*, it is crucial to provide some guarantees on their *robustness*. This is the ability of a system to function correctly even in presence of uncontrollable events affecting its behaviour, as, e.g., unexpected physical phenomena, failures, or cyber-physical attacks.

In the literature, we can find a wealth of proposals of robustness properties, that differ in the underlying model (including how uncertainty is modelled), in the formalisation, or in whether they are designed to analyse a specific feature of the behaviour of systems. We refer to [24, 38, 40, 43] for an overview of these notions. Although it seems natural to us that different application contexts call for different formalisations of robustness, the downside of this variety is the lack of a general tool for the verification of robustness properties.

In this paper we provide a formal framework for the verification of robustness properties of CPS. In this setting, robustness is usually formalised as a *measure* of the capability of agents to tolerate perturbations in the environmental conditions and still fulfil their tasks. This boils down to *quantifying the differences* between the behaviour of the system with its behaviour under the effect of perturbations, possibly at *different moments in time*. Intuitively, the system is robust if whenever the two behaviours are initially at a bounded distance, then



© Valentina Castiglioni, Michele Loreti, and Simone Tini;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 15; pp. 15:1–15:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

their distance after a given amount of time should always be smaller than a given threshold. This means that whenever we require a CPS to be robust against perturbations, we are actually specifying a *property on the evolution in time of distances between behaviours*.

Hence, a formal framework for the specification of similar properties should include:

- A model for the specification of the behaviour of CPS.
- A mechanism for the specification of the effects of perturbations on their behaviour.
- A mechanism to define distances between the behaviours of CPS.
- A temporal logic for the specification of properties on the evolution of those distances.

We adopt the model from the literature: the *evolution sequence model* introduced in [10,11]. The two mechanisms and the logic are introduced in this paper.

The evolution sequence model. The evolution sequence model follows a discrete-time, data-driven approach: the behaviour of the system is modelled in terms of the modifications that the interaction of the agents with the environment induces on a set of application-relevant data, called *data state*. Due to the unpredictability of the environment and potential approximations in the specification of agents, those modifications are modelled as continuous *distributions* on the attainable data states. The *evolution sequence* of a system is then defined as the sequence of the distributions over data states that are obtained at each time step.

The reason why we chose this model over classical and more established ones, like Labelled Markov Chains and Stochastic Hybrid Systems [7,27], is purely technical. The most prominent consequence of the design choices in this model is that the behaviour of the system is not given by a set of traces/trajectories, but by the combination of their effects. In other words, an evolution sequence is the discrete-time version of the cylinder of all possible trajectories of the system. This means that a property of an evolution sequence takes into account the outcomes of all possible observations, at a given time step, on the system. This is fundamental in the verification of robustness, since even the slightest modification induced by uncertainty on behaviour is taken into account.

Robustness Temporal Logic. We introduce *Robustness Temporal Logic (RobTL)* that allows us to compare distances between nominal and perturbed evolution sequences over a finite time horizon, by also providing the means to specify the perturbations and the distances. Specifically, RobTL offers:

- A class of *distance expressions* for the definition of arbitrary distances between evolution sequences. This freedom allows us to compare systems with respect to different aspects of behaviour in time, as well as to combine distances having different formulations.
- A class of *perturbations* for the definition of the effects of unpredictable events on the behaviour of the system.
- Atomic propositions $\Delta(\text{exp}, \mathbf{p}) \bowtie \eta$ to evaluate, at a given time step, the *distance*, specified by a *distance expression* exp , between a given evolution sequence and its perturbed version, obtained by some *perturbation* \mathbf{p} , and to compare it with the threshold η .
- Classical Boolean and temporal operators for the analysis of the evolution of the specified distances over a finite time horizon.

We provide a *statistical model checking algorithm* for the verification of RobTL specifications. As our algorithms are based on statistical inference, we need to account for the statistical error when checking formulae. Hence, we also propose a three-valued semantics for RobTL, in which the truth value *unknown* suggests that the parameters in the property need some tuning, or that a larger number of samples is needed to obtain a precise evaluation of the

distances. To showcase its features, we apply our framework to the analysis of a case-study from Industrial Control Systems: an engine system that is subject to cyber-physical attacks aimed at inflicting overstress of equipment [25].

All the algorithms and examples have been implemented in the tool STARK [12,14].

Why a new logic? RobTL is the only existing temporal logic expressing properties of distances between systems behaviours. Usually, even in logic equipped with a real-valued semantics, the behaviour of a *single given system* is compared to the desired property. Conversely, in RobTL the behaviours of *two systems* are taken into account. More precisely, we distinguish three approaches to the specification of properties in the quantitative setting:

- Specification of properties over a *single trajectory* of the system. This is the classic approach of PCTL, probabilistic LTL, and their variants [4,30,39].
- Specification of properties across the trajectories of the system. This is the *hyper-property* [15] approach of, e.g., HyperPCTL [2] or HPSTL [3], where one can express quantitative dependencies, in the form of bounds on probabilistic weights, between *different independent trajectories* of the system.
- Specification of properties based on the comparison of *all possible trajectories* of *two different systems*. This is the approach of RobTL, where one system is the perturbed version of the other. This feature also distinguishes our approach to robustness from classical ones, like those in [18,22]. Our properties are based on the comparison of the evolution sequences of two different systems, whereas [18,22] compare a single trajectory of a single system with the set of the behaviours that satisfy a given property, which is specified by means of a formula expressed in a suitable temporal logic, like, e.g., STL.

2 The Evolution Sequence Model

We recall the main elements of the evolution sequence model [11]. Systems consist of a set of *agents* and an *environment*, whose interaction produces changes on a *data space* \mathcal{D} , containing the values assumed by *variables*, from a *finite* set Var , representing:

- (i) physical quantities,
- (ii) sensors,
- (iii) actuators, and
- (iv) internal variables of agents.

For each $x \in \text{Var}$, the domain $\mathcal{D}_x \subseteq \mathbb{R}$ is either *finite*, or a *compact* subset of \mathbb{R} (and thus Polish), and equipped with the Borel σ -algebra \mathcal{B}_x . Then $\mathcal{D} = \prod_{x \in \text{Var}} \mathcal{D}_x$ and we equip it with the product σ -algebra $\mathcal{B}_{\mathcal{D}} = \otimes_{x \in \text{Var}} \mathcal{B}_x$ [6]. Let $\Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ be the set of distributions over $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$.

We call *data state* the current state of the data space, and represent it by a mapping $\mathbf{d}: \text{Var} \rightarrow \mathbb{R}$, with $\mathbf{d}(x) \in \mathcal{D}_x$ for all $x \in \text{Var}$. At each step, the agents and the environment induce some changes on the data state, providing a new data state at the next step. Those modifications are also subject to the presence of uncertainties, meaning that it is not always possible to determine exactly the values assumed by the data at the next step. Hence, we model the changes induced at each step as a distribution on the attainable data states. The behaviour of the system is then expressed by its *evolution sequence*, i.e., the sequence of distributions over the data states obtained at each step.

In this paper we do not focus on how evolution sequences are generated. In [11, Prop. 3.15] it was proved that the function defining the combined behaviour of the agents and the environment, specified according to the framework, is a *Markov kernel*. Hence, we simply assume a Markov kernel $\text{step}: \mathcal{D} \rightarrow \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ governing the evolution of the system, and define the evolution sequence as the *Markov process* generated by step (Definition 1): $\text{step}(\mathbf{d})(\mathbb{D})$

expresses the probability to reach a data state in \mathbb{D} from \mathbf{d} in one computation step. Indeed, each system is characterised by a particular function **step** starting from an *initial distribution* over \mathcal{D} . For instance, for the engine system of our case study (presented below), the initial distribution is a Dirac distribution over a chosen data state, and function **step** is obtained by combining the effects of the agents in Figure 1c and the environment in Figure 1d.

► **Definition 1.** Let $\text{step}: \mathcal{D} \rightarrow \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ be the Markov kernel generating the behaviour of a system \mathbf{s} having μ as initial distribution. The evolution sequence of \mathbf{s} is a countable sequence $\mathcal{S}_{\mu} = \mathcal{S}_{\mu}^0, \mathcal{S}_{\mu}^1, \dots$ of distributions in $\Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ such that, for all $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$:

$$\mathcal{S}_{\mu}^0(\mathbb{D}) = \mu(\mathbb{D}) \quad \text{and} \quad \mathcal{S}_{\mu}^{i+1}(\mathbb{D}) = \int_{\mathcal{D}} \text{step}(\mathbf{d})(\mathbb{D}) d\mathcal{S}_{\mu}^i(\mathbf{d}).$$

► **Remark 2.** We shall write $\mathcal{S}, \mathcal{S}_1$ in place of, respectively, $\mathcal{S}_{\mu}, \mathcal{S}_{\mu_1}$, whenever the formalisation of the initial distributions μ, μ_1 does not play a direct role in the discussion.

Case study: the engine system. As a running example, we consider a refrigerated engine system [31], sketched in Figure 1. There is one agent with three tasks:

- (i) regulate the speed,
- (ii) maintain the temperature within a specific range by means of a cooling system, and
- (iii) detect anomalies.

The first two tasks are on charge of a controller, the other is took over by an intrusion detection system, henceforth IDS. Figure 1a shows that these two components use **channels** to exchange information and to communicate with other agents. The variables used in the system, and their role, are listed in Figure 1b. Variable *stress* quantifies the level of *equipment stress*, which increases when the temperature stays too often above the threshold 100: the higher the stress, the higher the probability of a wreckage. The agent and the environment acting on these data have been specified in STARK, and are reported in Figure 1c and 1d, respectively. At each scan cycle the controller sets **internal variables** and **actuators** according to the values received from **sensors**, the IDS raises a warning if the status of **sensors** and **actuators** is unexpected, and the environment models the probabilistic evolution of the temperature. Notice that the controller must use channel *ch_temp* to receive data from sensor *temp*. Even though the use of channels is a common feature in CPS, it exposes them to attacks, as we will discuss in Example 11. We assume that the engine can cooperate with other engines (e.g., in an aircraft with a *left* and a *right* engine), by receiving values on channel *ch_in* and sending values on *ch_out*. If the engine is required to work at slow speed, it asks to other engines to proceed at full speed to compensate the lack of performance.

3 Robustness Temporal Logic

Robustness Temporal Logic (RobTL) allows us to express temporal properties of distances over systems behaviour, and, thus, to specify and verify robustness properties of CPS against uncertainty and perturbations. RobTL uses atomic propositions of the form $\Delta(\text{exp}, \mathbf{p}) \bowtie \eta$ to evaluate, at a given time step, the *distance*, specified by an *expression* **exp**, between a given evolution sequence and its perturbed version, obtained by some *perturbation* **p**, and to compare it with the threshold η . Atomic propositions are then combined with classic Boolean and temporal operators, in order to extend and compare these evaluations over the chosen time horizon. Hence, RobTL formulae are defined over three main components:

1. A language **DistExp** to specify *distances*.
2. A language **Pert** to specify *perturbations*.
3. Classic Boolean and temporal operators to specify requirements on the *evolution of distances in time*.

Then, we introduce the language `DistExp` whose operators allow us to combine various instances of this ground distance and define more complex distances over evolution sequences, while possibly taking into account *different objectives* of the system and *temporal constraints*.

Ground distance on distributions. In our setting, as in most application contexts, the objectives of the system can be expressed in a purely data-driven fashion: at any step, any difference between the desired value of some parameters of interest and the data actually obtained can be interpreted as a flaw in the behaviour of the system. Hence, following [10], to capture a particular objective, we use *penalty functions* $\rho: \mathcal{D} \times \mathbb{N} \rightarrow [0, 1]$ assigning to each pair \mathbf{d} , τ a *penalty* in $[0, 1]$, expressing how far the values of the parameters related to the considered task in \mathbf{d} are from their desired ones at step τ . Hence, $\rho(\mathbf{d}, \tau) = 0$ if \mathbf{d} respects all the parameters at step τ . For brevity, we let $\rho_\tau(\mathbf{d}) = \rho(\mathbf{d}, \tau)$. We remark that since we are in a the discrete-time setting, we can safely identify time steps with natural numbers.

► **Example 3.** Consider the engine system from our case study. We define the penalty functions ρ^w , ρ^t , and ρ^s , for all time steps τ , by:

$$\rho_\tau^w(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{d}(ch_wrn) = \text{hot,} \\ 0 & \text{if } \mathbf{d}(ch_wrn) = \text{ok} \end{cases} \quad \begin{cases} \rho_\tau^t(\mathbf{d}) = |\mathbf{d}(ch_temp) - \mathbf{d}(temp)|/150 \\ \rho_\tau^s(\mathbf{d}) = \mathbf{d}(stress) \end{cases}$$

They express, respectively, how far the level of alert raised by the IDS, the value carried by channel `ch_temp`, and the level of stress are from their desired value. These coincide with the value `ok`, the value of sensor `temp`, and zero, respectively.

Penalty functions can be used also to express *false negatives* and *false positives*, representing, respectively, the average *effectiveness*, and the average *precision* of the IDS to signal through channel `ch_wrn` that the engine system is under stress. We use two new variables fn and fp to quantify false negatives and false positives depending on *stress* and *ch_wrn*. Both variables are initialised to 0 and updated as follows:

$$fn(\tau+1) = \frac{\tau * fn(\tau) + \max(0, stress(\tau) - \gamma)}{\tau + 1} \quad fp(\tau+1) = \frac{\tau * fp(\tau) + \max(0, \gamma - stress(\tau))}{\tau + 1}$$

where γ is 0 if `ch_wrn`(τ) is `ok`, and γ is 1 if `ch_wrn`(τ) is `hot`. Then, penalties $\rho_\tau^{fn}(\mathbf{d}) = \mathbf{d}(fn)$ and $\rho_\tau^{fp}(\mathbf{d}) = \mathbf{d}(fp)$ express how far fn and fp are from their ideal value 0.

We can then make use of penalty functions to define a *distance on data states*:

► **Definition 4.** Let ρ be a penalty function, and $\tau \in \mathbb{N}$. The metric on data states in \mathcal{D} , $\mathbf{m}_\tau^\rho: \mathcal{D} \times \mathcal{D} \rightarrow [0, 1]$, is defined, for all $\mathbf{d}_1, \mathbf{d}_2 \in \mathcal{D}$, by $\mathbf{m}_\tau^\rho(\mathbf{d}_1, \mathbf{d}_2) = \max\{\rho_\tau(\mathbf{d}_2) - \rho_\tau(\mathbf{d}_1), 0\}$.

Note that $\mathbf{m}_\tau^\rho(\mathbf{d}_1, \mathbf{d}_2)$ is a hemimetric, i.e., a pseudometric that is not required to be symmetric, expressing how much \mathbf{d}_2 is *worse* than \mathbf{d}_1 according to ρ_τ .

Finally, we need to lift the hemimetric \mathbf{m}_τ^ρ to a hemimetric over $\Pi(\mathcal{D}, \mathcal{B}_\mathcal{D})$. In the literature, we can find a wealth of notions of function lifting doing so (see [34] for a survey). Among those, the *Wasserstein lifting* [47] has the following advantages:

- (i) it preserves the properties of the ground metric, and
- (ii) one can apply statistical inference to obtain good approximations of it, whose exact computation is tractable [11, 44, 46].

► **Definition 5.** Let ρ be a penalty function, and $\tau \in \mathbb{N}$. For any $\mu, \nu \in \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$, the Wasserstein lifting of \mathbf{m}_{τ}^{ρ} to a distance between μ and ν is defined by:

$$\mathbf{W}(\mathbf{m}_{\tau}^{\rho})(\mu, \nu) = \inf_{\mathfrak{w} \in \mathfrak{W}(\mu, \nu)} \int_{\mathcal{D} \times \mathcal{D}} \mathbf{m}_{\tau}^{\rho}(\mathbf{d}, \mathbf{d}') d\mathfrak{w}(\mathbf{d}, \mathbf{d}')$$

where $\mathfrak{W}(\mu, \nu)$ is the set of the couplings of μ and ν , namely the set of joint distributions \mathfrak{w} over the product space $(\mathcal{D} \times \mathcal{D}, \mathcal{B}_{\mathcal{D} \times \mathcal{D}})$ having μ and ν as left and right marginal, respectively, i.e., $\mathfrak{w}(\mathbb{D} \times \mathcal{D}) = \mu(\mathbb{D})$ and $\mathfrak{w}(\mathcal{D} \times \mathbb{D}) = \nu(\mathbb{D})$, for all $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$. (See [11, 23] and Appendix A for an in depth discussion on the definition of the Wasserstein lifting over hemimetrics.)

► **Proposition 6** ([11]). For each penalty function ρ , and time step $\tau \in \mathbb{N}$, function $\mathbf{W}(\mathbf{m}_{\tau}^{\rho})$ is a 1-bounded hemimetric on $\Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$.

The language DistExp. We can now proceed to define distances that take into account several time steps in the evolution of systems.

► **Definition 7.** Expressions in **DistExp** are defined as follows:

$$\begin{aligned} \text{exp} ::= & \langle^{\rho} \mid \rangle^{\rho} \mid \mathbf{F}^I \text{exp} \mid \mathbf{G}^I \text{exp} \mid \text{exp} \mathbf{U}^I \text{exp} \mid \\ & \min(\text{exp}, \text{exp}) \mid \max(\text{exp}, \text{exp}) \mid \sum_{k \in K} w_k \cdot \text{exp}_k \mid \sigma(\text{exp}, \bowtie \zeta) \end{aligned}$$

where ρ ranges over penalty functions, I is an interval, K is a finite set of indexes, $w_k \in (0, 1]$ for each $k \in K$, $\sum_{k \in K} w_k = 1$, $\bowtie \in \{<, \leq, \geq, >\}$ and $\zeta \in [0, 1]$.

Atomic expressions \langle^{ρ} and \rangle^{ρ} are used to evaluate the ground distance with respect to the penalty function ρ . We have two distinct atomic expressions because the ground distance is a hemimetric: the direction of the arrowhead in \langle^{ρ} and \rangle^{ρ} identifies which argument is considered as the *first* one in the evaluation (cf. Definition 8 below). Moreover, having penalty functions as parameters allows us to study the differences in the behaviour of systems with respect to different data and objectives in time. Then, we provide three *temporal expression* operators, namely \mathbf{F}^I , \mathbf{G}^I and \mathbf{U}^I , allowing for the evaluation of minimal and maximal distances over a given time interval I . The *comparison operator* $\sigma(\text{exp}, \bowtie \zeta)$ returns a value in $\{0, 1\}$ used to establish whether the evaluation of **exp** is in relation \bowtie with the threshold ζ .

Distance expressions are evaluated over two evolution sequences and a time τ , representing the time step at which (or starting from which) the differences are computed.

► **Definition 8.** Let $\mathcal{S}_1, \mathcal{S}_2$ be to evolution sequences, and τ be a time step. The evaluation of distance expressions in the triple $\mathcal{S}_1, \mathcal{S}_2, \tau$ is the function $\llbracket \cdot \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} : \mathbf{DistExp} \rightarrow [0, 1]$ defined inductively on the structure of expressions as follows:

- $\llbracket \langle^{\rho} \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \mathbf{W}(\mathbf{m}_{\tau}^{\rho})(\mathcal{S}_1^{\tau}, \mathcal{S}_2^{\tau});$
- $\llbracket \rangle^{\rho} \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \mathbf{W}(\mathbf{m}_{\tau}^{\rho})(\mathcal{S}_2^{\tau}, \mathcal{S}_1^{\tau});$
- $\llbracket \mathbf{F}^I \text{exp} \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \min_{t \in I + \tau} \llbracket \text{exp} \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^t;$
- $\llbracket \mathbf{G}^I \text{exp} \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \max_{t \in I + \tau} \llbracket \text{exp} \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^t;$
- $\llbracket \text{exp}_1 \mathbf{U}^I \text{exp}_2 \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \min_{t \in I + \tau} \max \left\{ \llbracket \text{exp}_2 \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^t, \max_{t' \in I + \tau, t' < t} \llbracket \text{exp}_1 \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{t'} \right\};$
- $\llbracket \min(\text{exp}_1, \text{exp}_2) \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \min \{ \llbracket \text{exp}_1 \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau}, \llbracket \text{exp}_2 \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} \};$
- $\llbracket \max(\text{exp}_1, \text{exp}_2) \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \max \{ \llbracket \text{exp}_1 \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau}, \llbracket \text{exp}_2 \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} \};$
- $\llbracket \sum_{k \in K} w_k \text{exp}_k \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \sum_{k \in K} w_k \cdot \llbracket \text{exp}_k \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau};$
- $\llbracket \sigma(\text{exp}, \bowtie \zeta) \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} = \begin{cases} 0 & \text{if } \llbracket \text{exp} \rrbracket_{\mathcal{S}_1, \mathcal{S}_2}^{\tau} \bowtie \zeta, \\ 1 & \text{otherwise.} \end{cases}$

The evaluation bases on the two atomic expressions. We use $<^\rho$ to measure the distance between the distributions reached by \mathcal{S}_1 and \mathcal{S}_2 at time τ (\mathcal{S}_τ^1 and \mathcal{S}_τ^2) with respect to the penalty function ρ , i.e., $\mathbf{W}(\mathbf{m}_\tau^\rho)(\mathcal{S}_\tau^1, \mathcal{S}_\tau^2)$. Conversely, $>^\rho$ measures the distance $\mathbf{W}(\mathbf{m}_\tau^\rho)(\mathcal{S}_\tau^2, \mathcal{S}_\tau^1)$. Operators F^I , G^I , and U^I can be thought of as the quantitative versions of the corresponding bounded temporal operators, respectively, *eventually*, *always*, and *until*. Their semantics follows by associating existential quantification with minima, and universal quantification with maxima. Hence, the evaluation of $F^I \mathbf{exp}$ is obtained as the minimum value of the distance \mathbf{exp} over the time interval I . Dually, $G^I \mathbf{exp}$ gives us the maximum value of \mathbf{exp} over I . Then, the evaluation of $\mathbf{exp}_1 U^I \mathbf{exp}_2$ follows from that of bounded until (see Definition 14), accordingly. The expression $\sigma(\mathbf{exp}, \bowtie \zeta)$ evaluates to 0 if the evaluation of \mathbf{exp} is $\bowtie \zeta$; otherwise it evaluates to 1. Informally, the comparison operator σ can be combined with temporal expression operators to check if several constraints of the form $\bowtie \zeta_i$ are satisfied over a time interval under a single application of a perturbation function (see Example 16 below).

3.2 Perturbations

A *perturbation* is the effect of unpredictable events on the current state of the system. Hence, we model it as a function that maps a data state into a distribution over data states. To account for possibly repeated, or different effects in time of a single perturbation, we make the definition of perturbation function also time-dependent: a perturbation function \mathbf{p} is a list of mappings in which the i -th element describes the effects of \mathbf{p} at time i .

► **Definition 9.** A perturbation function is a mapping $\mathbf{p}: \mathcal{D} \times \mathbb{N} \rightarrow \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ such that, for each $\tau \in \mathbb{N}$, the mapping $\mathbf{d} \mapsto \mathbf{p}(\mathbf{d}, \tau)(\mathbb{D})$ is $\mathcal{B}_{\mathcal{D}}$ -measurable for all $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$.

To describe the perturbed behaviour of a system, we need to account for the effects of a function \mathbf{p} on the evolution sequence. Hence, we combine \mathbf{p} with the Markov kernel **step**:

► **Definition 10.** Given an evolution sequence \mathcal{S}_μ generated by kernel **step**, and a perturbation function \mathbf{p} , the perturbation of \mathcal{S}_μ via \mathbf{p} is the evolution sequence $\mathcal{S}_\mu^{\mathbf{p}}$ obtained by:

$$\mathcal{S}_\mu^{\mathbf{p},0}(\mathbb{D}) = \int_{\mathcal{D}} \mathbf{p}(\mathbf{d}, 0)(\mathbb{D}) d\mu(\mathbf{d}), \quad \mathcal{S}_\mu^{\mathbf{p},i+1}(\mathbb{D}) = \int_{\mathcal{D}} \int_{\mathcal{D}} \mathbf{p}(\mathbf{d}', i+1)(\mathbb{D}) d\mathbf{step}(\mathbf{d})(\mathbf{d}') d\mathcal{S}_\mu^{\mathbf{p},i}(\mathbf{d}).$$

Specifying perturbations. We specify a perturbation function \mathbf{p} via a (syntactic) perturbation \mathbf{p} in the language **Pert**:

$$\mathbf{p} ::= \text{nil} \quad | \quad \mathbf{f}@_\tau \quad | \quad \mathbf{p}_1; \mathbf{p}_2 \quad | \quad \mathbf{p}^n$$

where \mathbf{p} ranges over **Pert**, n and τ are finite natural numbers, and:

- **nil** is the perturbation with *no effects*, i.e., at each time step it behaves like the identity function $\text{id}: \mathcal{D} \rightarrow \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ such that $\text{id}(\mathbf{d}) = \delta_{\mathbf{d}}$ for all $\mathbf{d} \in \mathcal{D}$;
- **f@ τ** is an *atomic perturbation*, i.e., a function $\mathbf{f}: \mathcal{D} \rightarrow \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ such that the mapping $\mathbf{d} \mapsto \mathbf{f}(\mathbf{d})(\mathbb{D})$ is $\mathcal{B}_{\mathcal{D}}$ -measurable for all $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$, and that is applied precisely after τ time steps from the current instant;
- **p₁; p₂** is a *sequential perturbation*, i.e., perturbation \mathbf{p}_2 is applied at the time step subsequent to the (final) application of \mathbf{p}_1 ;
- **pⁿ** is an *iterated perturbation*, i.e., perturbation \mathbf{p} is applied for a total of n times.

Despite its simplicity, this language allows us to define some non-trivial perturbation functions that we can use to test systems behaviour. (Unspecified perturbations behave like **id**.)

► **Example 11.** In [31] several cyber-physical attacks tampering with sensors or actuators of the engine system aiming to inflict *overstress of equipment* [25] were described. Here we show how those attacks can be modelled by employing our perturbations.

Consider sensor *temp*. There is an attack that aims at delaying the cooling phase, forcing the system to work for several instants at high temperatures and accumulate stress. It tricks the controller by adding a negative offset $o \in \mathbb{R}^{\leq 0}$ to the value carried by the insecure channel *ch_temp*. If $[100, 200]$ is the attack window, the attack is modelled by perturbation $\mathbf{p}_{temp,o} = (\mathbf{id}@0)^{100}; (\mathbf{f}_{temp,o}@0)^{100}$, where $\mathbf{f}_{temp,o}(\mathbf{d}) = \delta_{\mathbf{d}'}$ with $\mathbf{d}'(ch_temp) = \mathbf{d}(ch_temp) + \mathbf{rnd} * o$, with \mathbf{rnd} uniformly distributed in $[0, 1]$, and $\mathbf{d}'(x) = \mathbf{d}(x)$ for all other variables in Figure 1b.

Consider now actuator *cool*. There is an attack aims at forcing the system to reach quickly high temperatures after the start of a cooling phase. It switches off the cooling system as soon as the temperatures goes below $99.8 - t$ degrees, for some $t \in \mathbb{R}^{\geq 0}$. This attack, is *stealth*, meaning that the IDS does not detect it. If $[0, 100]$ is the attack window, the attack is modelled by perturbation $\mathbf{p}_{cool,t}$ defined by $\mathbf{p}_{cool,t} = (\mathbf{f}_{cool,t}@0)^{100}$, where $\mathbf{f}_{cool,t}(\mathbf{d}) = \delta_{\mathbf{d}}$, if $\mathbf{d}(temp) \geq 99.8 - t$, and $\mathbf{f}_{cool,t}(\mathbf{d}) = \delta_{\mathbf{d}'}$, otherwise, with $\mathbf{d}''(cool) = \mathbf{off}$, and $\mathbf{d}''(x) = \mathbf{d}(x)$ for all other variables in Figure 1b.

Each $\mathbf{p} \in \mathbf{Pert}$ denotes a perturbation function as in Definition 9. To obtain it, we exploit function $\mathbf{effect}(\mathbf{p})$, describing the effects of \mathbf{p} at the current step, and function $\mathbf{next}(\mathbf{p})$, identifying the perturbation to be applied at next step. Both functions are defined inductively on the structure of perturbations.

$$\begin{aligned} \mathbf{effect}(\mathbf{nil}) &= \mathbf{id} & \mathbf{effect}(\mathbf{f}@_\tau) &= \begin{cases} \mathbf{id} & \text{if } \tau > 0, \\ \mathbf{f} & \text{if } \tau = 0 \end{cases} \\ \mathbf{effect}(\mathbf{p}^n) &= \mathbf{effect}(\mathbf{p}) & \mathbf{effect}(\mathbf{p}_1; \mathbf{p}_2) &= \mathbf{effect}(\mathbf{p}_1) \\ \mathbf{next}(\mathbf{nil}) &= \mathbf{nil} & \mathbf{next}(\mathbf{f}@_\tau) &= \begin{cases} \mathbf{f}@(\tau - 1) & \text{if } \tau > 0, \\ \mathbf{nil} & \text{otherwise} \end{cases} \\ \mathbf{next}(\mathbf{p}^n) &= \begin{cases} \mathbf{next}(\mathbf{p}); \mathbf{p}^{n-1} & \text{if } n > 0, \\ \mathbf{nil} & \text{otherwise} \end{cases} & \mathbf{next}(\mathbf{p}_1; \mathbf{p}_2) &= \begin{cases} \mathbf{next}(\mathbf{p}_1); \mathbf{p}_2 & \text{if } \mathbf{next}(\mathbf{p}_1) \neq \mathbf{nil}, \\ \mathbf{p}_2 & \text{otherwise.} \end{cases} \end{aligned}$$

We define the semantics of perturbations as the mapping $\langle\langle \cdot \rangle\rangle : \mathbf{Pert} \rightarrow (\mathcal{D} \times \mathbb{N} \rightarrow \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}}))$ such that, for all $\mathbf{d} \in \mathcal{D}$ and $i \in \mathbb{N}$, $\langle\langle \mathbf{p} \rangle\rangle(\mathbf{d}, i) = \mathbf{effect}(\mathbf{next}^i(\mathbf{p}))(\mathbf{d})$, where $\mathbf{next}^0(\mathbf{p}) = \mathbf{p}$ and $\mathbf{next}^i(\mathbf{p}) = \mathbf{next}(\mathbf{next}^{i-1}(\mathbf{p}))$, for all $i > 0$.

► **Proposition 12.** For each $\mathbf{p} \in \mathbf{Pert}$, $\langle\langle \mathbf{p} \rangle\rangle$ is a well defined perturbation function.

Proof. Since, by definition, each \mathbf{f} occurring in atomic perturbations is such that $\mathbf{d} \mapsto \mathbf{f}(\mathbf{d})(\mathbb{D})$ is $\mathcal{B}_{\mathcal{D}}$ -measurable for all $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$, and the same property trivially holds for the identity function \mathbf{id} , it is immediate to conclude that $\langle\langle \mathbf{p} \rangle\rangle$ satisfies Definition 9 for each $\mathbf{p} \in \mathbf{Pert}$. ◀

3.3 RobTL Formulae

We use RobTL formulae to specify and analyse distances between nominal and perturbed evolution sequences over a finite time horizon \mathfrak{h} . By combining atomic propositions with temporal operators, we can apply (possibly) different distance expressions and perturbations at different steps, thus allowing for an analysis of behaviour in complex scenarios.

15:10 RobTL: Robustness Temporal Logic for CPS

► **Definition 13.** *RobTL consists in the set of formulae L defined by:*

$$\varphi ::= \top \mid \Delta(\text{exp}, \mathbf{p}) \bowtie \eta \mid \neg \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U}^I \varphi$$

where φ ranges over L , exp ranges over distance expressions in DistExp , \mathbf{p} ranges over perturbations in Pert , $\bowtie \in \{<, \leq, \geq, >\}$, $\eta \in [0, 1]$, and $I \subseteq [0, h]$ is a bounded time interval.

Formulae are evaluated in an evolution sequence and a time step.

► **Definition 14.** *Let \mathcal{S} be an evolution sequence, and τ a time step. The satisfaction relation \models is defined inductively on the structure of formulae as:*

- $\mathcal{S}, \tau \models \top$ for all \mathcal{S}, τ ;
- $\mathcal{S}, \tau \models \Delta(\text{exp}, \mathbf{p}) \bowtie \eta$ iff $[\text{exp}]_{\mathcal{S}, \mathcal{S}_{|\langle\langle \mathbf{p} \rangle\rangle, \tau}^\tau} \bowtie \eta$, with evolution sequence $\mathcal{S}_{|\langle\langle \mathbf{p} \rangle\rangle, \tau}$ defined as:

$$(\mathcal{S}_{|\langle\langle \mathbf{p} \rangle\rangle, \tau})^t = \begin{cases} \mathcal{S}^t & \text{if } t < \tau, \\ \mathcal{S}_{\mathcal{S}^\tau}^{\langle\langle \mathbf{p} \rangle\rangle, t-\tau} & \text{if } t \geq \tau; \end{cases}$$

- $\mathcal{S}, \tau \models \neg \varphi$ iff $\mathcal{S}, \tau \not\models \varphi$;
- $\mathcal{S}, \tau \models \varphi_1 \wedge \varphi_2$ iff $\mathcal{S}, \tau \models \varphi_1$ and $\mathcal{S}, \tau \models \varphi_2$;
- $\mathcal{S}, \tau \models \varphi_1 \mathcal{U}^I \varphi_2$ iff there is a $\tau' \in I + \tau$ such that $\mathcal{S}, \tau' \models \varphi_2$ and for all $\tau'' \in I + \tau$, $\tau'' < \tau'$ it holds that $\mathcal{S}, \tau'' \models \varphi_1$, where, for $I = [a, b]$, we let $I + \tau = [\min\{a + \tau, h\}, \min\{b + \tau, h\}]$.

Let us focus on atomic propositions. The evolution sequence \mathcal{S} at time τ satisfies the formula $\Delta(\text{exp}, \mathbf{p}) \bowtie \eta$ if the distance defined by exp between \mathcal{S} and $\mathcal{S}_{|\langle\langle \mathbf{p} \rangle\rangle, \tau}$ is $\bowtie \eta$, where $\mathcal{S}_{|\langle\langle \mathbf{p} \rangle\rangle, \tau}$ is the evolution sequence obtained by applying the perturbation \mathbf{p} to \mathcal{S} at time τ . For the first $\tau - 1$ steps $\mathcal{S}_{|\langle\langle \mathbf{p} \rangle\rangle, \tau}$ is identical to \mathcal{S} . At time τ the perturbation \mathbf{p} is applied, and the distributions in $\mathcal{S}_{|\langle\langle \mathbf{p} \rangle\rangle, \tau}$ are thus given by the perturbation via $\langle\langle \mathbf{p} \rangle\rangle$ of the evolution sequence having \mathcal{S}^τ as initial distribution (Definition 10). It is worth noticing that by combining atomic propositions with temporal operators we can apply (possibly) different perturbations at different time steps, thus allowing for the analysis of systems behaviour in complex scenarios.

Naturally, other operators can be defined as macros in our logic:

$$\varphi_1 \vee \varphi_2 \equiv \neg(\neg \varphi_1 \wedge \neg \varphi_2) \quad \varphi_1 \implies \varphi_2 \equiv \neg \varphi_1 \vee \varphi_2 \quad \diamond^I \varphi \equiv \top \mathcal{U}^I \varphi \quad \square^I \varphi \equiv \neg \diamond^I \neg \varphi.$$

We now provide some examples of robustness properties that can be expressed in RobTL.

► **Example 15.** By using the penalty functions in Example 3 and the perturbations from Example 11, we can build a formula φ_1 expressing that the attack on the insecure channel ch_temp is successful. This happens if, whenever the difference observed along the attack window $I = [100, 200]$ between the physical value of temperature and that read by the controller is in the interval $[\eta_1, \eta_2]$, for suitable η_1 and η_2 , then the level of the alarm raised by the IDS remains below a given stealthiness threshold η_3 , and the level of system stress overcomes a danger threshold η_4 within the time interval $J = [100, 210]$:

$$\begin{aligned} \varphi_1 &= \diamond^{[0, h]}(\varphi'_1 \implies \varphi''_1) \\ \varphi'_1 &= \Delta(\mathbf{F}^I \langle^{\rho^t}, \mathbf{p}_{temp, o}\rangle \geq \eta_1 \wedge \Delta(\mathbf{G}^I \langle^{\rho^t}, \mathbf{p}_{temp, o}\rangle \leq \eta_2) \\ \varphi''_1 &= \Delta(\mathbf{G}^J \langle^{\rho^w}, \mathbf{p}_{temp, o}\rangle \leq \eta_3 \wedge \Delta(\mathbf{G}^J \langle^{\rho^s}, \mathbf{p}_{temp, o}\rangle \geq \eta_4). \end{aligned}$$

► **Example 16.** Consider the attack on actuator *cool* described in Example 11. We give a formula φ_2 expressing that such an attack fails within 210 units of time. This happens if the level of the alarm raised by the IDS goes above a given threshold ζ_2 at some instant $\tau' \in [0, 210]$, i.e., the attack is detected, while the level of stress remains below an acceptable threshold ζ_1 :

$$\varphi_2 = \Delta \left(\sigma(\langle \rho^s, \zeta_1 \rangle) \mathbf{U}^{[0,210]} \sigma(\langle \rho^w, \zeta_2 \rangle), \mathbf{p}_{cool,t} \right) < 1.$$

Notice that in the formula φ_2 the perturbation $\mathbf{p}_{cool,t}$ is applied only once, at time 0, and by means of the comparison and until operators on expressions we can evaluate all the distances along the considered interval between the original evolution sequence and its perturbation via $\mathbf{p}_{cool,t}$. Conversely, in the formula φ_3 below, the time step at which a perturbation is applied is determined by the bounded until operator:

$$\varphi_3 = \varphi_2 \mathbf{U}^{[\tau_1, \tau_2]} \Delta(\langle \rho^{fn}, \mathbf{p}_{cool,t} \rangle \leq \eta_3).$$

This formula is satisfied if there is a $\tilde{\tau} \in [\tau_1, \tau_2]$ s.t.:

1. the attack on actuator *cool* is detected regardless of the time step in $[\tau_1, \tilde{\tau})$ at which $\mathbf{p}_{cool,t}$ is applied, and
2. the IDS is effective, up to tolerance η_3 , against an application of $\mathbf{p}_{cool,t}$ at time $\tilde{\tau}$.

The effectiveness is measured in terms of the penalty function ρ^{fn} on false negatives presented in Example 3.

4 RobTL model checker

A RobTL model checker has been implemented as part of the *Software Tool for the Analysis of Robustness in the unKnown environment* (STARK) [12, 14], available at <https://github.com/quasylab/jspear/tree/working>, and the related, detailed, documentation can be found at <https://github.com/quasylab/jspear/wiki>.

It consists of the following four procedures, based on *statistical techniques* and *simulation*:

- (i) Simulation of the evolution sequence of system, assuming an initial distribution μ .
- (ii) Simulation of the effects of a perturbation on a given evolution sequence.
- (iii) Syntax driven estimation of the evaluation of distance expressions.
- (iv) Satisfaction of a given RobTL formula by a given evolution sequence.

These four procedures are presented with more details below. Due to space constraints, since all the algorithms are implemented in STARK, we only report them in Appendix B. We also remark that the simulation procedure in (i) and that for the estimation of the Wasserstein distance that is part of (iii) were already discussed at length in [11]. We briefly report them here for the sake of readability. We discuss the time complexity of the model checking algorithm in Section 4.1. Since the procedures outlined above are based on statistical inference, we need to take into account the statistical error when checking the satisfaction of formulae. Hence, in Section 4.2 we discuss a classical algorithm for the evaluation of confidence intervals in the evaluation of distances. Then, we propose a three-valued semantics for RobTL specifications, in which the truth value *unknown* is added to true and false.

Throughout the section, we also provide some examples of an application of the algorithms to the analysis of the engine system. The interested reader can find the script used to generate the plots and results as the `Main.java` file at <https://github.com/quasylab/jspear/blob/working/examples/engine/>.

Simulation of evolution sequences. Given a distribution μ and $N, k \in \mathbb{N}$, we use function SIM (Algorithm 1) to obtain an *empirical evolution sequence* of the form E_0, \dots, E_k of size N and length k , starting from μ . Each E_i is a tuple $\mathbf{d}_i^1, \dots, \mathbf{d}_i^N$ of data states that are used to estimate the probability distribution reached at step i . E_0 is a sample of size N of μ obtained by means of a function SAMPLEDISTR. Then, E_{i+1} is obtained by simulating one computation step from each element in E_i via a function SIMSTEP that mimics the behaviour of step (Section 2): for any \mathbf{d} and $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$ it holds that $\Pr\{\text{SIMSTEP}(\mathbf{d}) \in \mathbb{D}\} = \text{step}(\mathbf{d})(\mathbb{D})$. For any $i \in [0, k]$, we let $\hat{\mathcal{S}}_{E_0}^{i,N}$ be the distribution such that $\hat{\mathcal{S}}_{E_0}^{i,N}(\mathbb{D}) = |E_i \cap \mathbb{D}|/N$ for any $\mathbb{D} \in \mathcal{B}_{\mathcal{D}}$. By applying the weak law of large numbers to the i.i.d. samples, we get $\lim_{N \rightarrow \infty} \hat{\mathcal{S}}_{E_0}^{i,N} = \mathcal{S}_{\mu}^i$. Henceforth, we identify $\hat{\mathcal{S}}_{E_0}^N$ with the estimated evolution sequence of size N , $\bar{E} = E_0, \dots, E_k$.

Applying perturbations to evolution sequences. We use function SIMPER (Algorithm 2) to simulate the effect of a perturbation \mathbf{p} on an estimated evolution sequence \bar{E} of size N . This function takes two integers as parameters: τ and ℓ . τ is the time step at which \mathbf{p} is applied. ℓ is the number of additional samples that we generate to evaluate the effect of \mathbf{p} on each data state, to guarantee statistical relevance of the collected data. Given E , we denote by $\ell \cdot E$ the sample set obtained from E by replicating each of its elements ℓ times. Function SIMPER is similar to SIM. They differ in constructing the tuple E_{i+1} , as in SIMPER we need first to sample the effect of \mathbf{p} . This is done by function SAMPLE($f(\mathbf{d})$) whose definition is standard and therefore omitted. According to Section 3.2, function f in SAMPLE($f(\mathbf{d})$) is effect(\mathbf{p}), and the perturbation used at next time step is next(\mathbf{p}).

Evaluation of distance expressions. Distance expressions are estimated following a syntax driven algorithm (Algorithm 4). To deal with atomic expressions \langle^{ρ} and \rangle^{ρ} , we rely on an existing approach [46] to estimate the Wasserstein distance between two distributions $\mu, \nu \in \Pi(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$. We consider N independent samples $\{\mathbf{d}_1^1, \dots, \mathbf{d}_1^N\}$ taken from μ , and ℓN independent samples $\{\mathbf{d}_2^1, \dots, \mathbf{d}_2^{\ell N}\}$ taken from ν , for some integers N, ℓ . Then, we exploit the penalty function ρ_i to map each sampled data state onto \mathbb{R} , so that, to capture the minimisation over the couplings, it is enough to consider the reordered sequences of values $\{\omega_j = \rho_i(\mathbf{d}_1^j) \mid \omega_j \leq \omega_{j+1}\}$ and $\{\nu_h = \rho_i(\mathbf{d}_2^h) \mid \nu_h \leq \nu_{h+1}\}$. Then $\mathbf{W}(m_{\rho,i})(\nu, \mu) = \frac{1}{\ell N} \sum_{h=1}^{\ell N} \max\{\nu_h - \omega_{\lceil \frac{h}{M} \rceil}, 0\}$ [10, 11].

Function WASS (Algorithm 3) implements the procedure outlined above. Parameter $op \in \{<, >\}$ of WASS allows us to choose which between \langle^{ρ} and \rangle^{ρ} we want to approximate: if op is $<$, we approximate $\mathbf{W}(m_{\rho,i})(\mu, \nu)$; if op is $>$, we approximate $\mathbf{W}(m_{\rho,i})(\nu, \mu)$. Since penalty functions allow us to evaluate \mathbf{W} on \mathbb{R} , rather than on \mathbb{R}^n , the complexity of the outlined procedure is $\mathcal{O}(\ell N \log(\ell N))$ [46], due to the sorting of $\{\nu_h \mid h \in [1, \dots, \ell N]\}$. We refer to [44, Corollary 3.5, Equation (3.10)] for an estimation of the approximation error on the evaluation of the Wasserstein distance over $N, \ell N$ samples.

Using the estimation of the atomic expressions as base case, we define function EVALEXP (Algorithm 4) recursively on the syntax of **exp**, following Definition 8.

Checking formulae satisfaction. Function SAT (Algorithm 5) allows us to verify whether a given evolution sequence satisfies a given RobTL formula at a given time step. It takes five parameters: the initial distribution μ , the time step τ , the formula φ , the two integers N and ℓ identifying the number of samples. Function SAT consists of three steps. Firstly, we compute, using structural induction, the *time horizon* k of φ to identify the number of steps needed to evaluate it. Then, function SIM is used to simulate the evolution sequence from μ . Finally, φ is evaluated over \bar{E} and τ by calling function EVAL (Algorithm 6), that yields the Boolean evaluation of φ computed recursively on its structure following Definition 14.

(a) Difference with respect to $temp$.(b) Difference with respect to $stress$.

■ **Figure 2** Differences with respect to the values of $temp$ and $stress$, under $p_{temp,o}$ for $o \in \{-2, -1.5, -1\}$.

(a) \exp_1 .(b) \exp_2 .

■ **Figure 3** Evaluation of \exp_1 and \exp_2 over the time interval $[0, 50]$.

► **Example 17.** Consider the attack on the sensor $temp$, modelled by perturbation $p = p_{temp,o}$ (Example 11), and let 0 be the current step. To give an idea of the impact of p on the behaviour \mathcal{S} of the engine, in Figure 2a we report the pointwise evaluation of the distance $\langle \rho$, where $\rho(\mathbf{d}) = \mathbf{d}(temp)/150$ for all $\mathbf{d} \in \mathcal{D}$, over the time window $[90, 300]$, between the temperature in \mathcal{S} and that in three perturbations of it, obtained by three variations of p with $o \in \{-2, -1.5, -1\}$. In all cases, the difference is greater in $[100, 200]$, i.e., while $f_{temp,o}$ is active, and the smaller differences detected after 200 steps are due to the delays induced by the perturbations in the regular behaviour. Clearly, the larger the offset interval, the greater the difference. This is even more evident in Figure 2b, depicting the pointwise evaluations of the distances $\langle \rho^s$ between \mathcal{S} and its three perturbed versions, for the penalty function ρ^s defined as $\rho^s(\mathbf{d}) = \mathbf{d}(stress)$ in Example 3.

Let us now fix $o = -1.5$. Consider expressions $\exp_1 = G^J \langle \rho^w$ and $\exp_2 = G^J \langle \rho^s$, where $J = [100, 210]$ and both penalty functions ρ^w and ρ^s are defined in Example 3. In Figure 3 we report the variation of the evaluation of the two expressions over \mathcal{S} and its 51 perturbations via p , each obtained by applying p at a different $\tau' \in [0, 50]$. We associate the coordinate $x = \tau'$ with $\llbracket \exp_1 \rrbracket_{\mathcal{S}, S_{\langle p \rangle}, \tau'}$ in Figure 3a, and with $\llbracket \exp_2 \rrbracket_{\mathcal{S}, S_{\langle p \rangle}, \tau'}$ in Figure 3b. The two plots show that by applying p at different time steps, we get different effects on system behaviour, with variations of the order of 10^{-3} . We run several experiments to infer for which stealthiness threshold η_3 and danger threshold η_4 , the formula φ_1 in Example 15 is satisfied. We concluded that for $\eta_3 \geq 0.06$ and $\eta_4 \leq 0.45$ the attack is successful (Example 20).

4.1 Complexity

We can assume that the evaluation of $\text{SIMSTEP}(\mathbf{d})$ needs a number of steps that is linear with the number of variables in \mathbf{d} . The same applies for the application of a perturbation or penalty function to a data state \mathbf{d} . Under these assumptions it is not hard to see that

to evaluate $\text{SIM}(\mu, N, k)$ we need $\mathcal{O}(kN \cdot |\text{Var}|)$ steps, while $\mathcal{O}(k\ell N \cdot |\text{Var}|)$ steps are needed to evaluate $\text{SIMP}ER(\{E_1, \dots, E_k\}, \mathbf{p}, \tau, \ell)$. Moreover, $\mathcal{O}(|E_2| \log |E_2|)$ steps are needed for $\text{WASS}(E_1, E_2, \text{op}, \rho)$. This is dominated by the number of steps needed to order the sequences ω_i and ν_h (lines 4 and 5). For the sake of simplicity, the algorithms in Algorithm 4 and Algorithm 6 are presented following a *forward* approach where to compute the value at time i , all the values in an interval $[i + a, i + b]$ could be considered. This means that to compute $\text{EVAL}(\overline{E}, \tau, \varphi, \ell)$ (resp. $\text{EVAL}(\overline{E}, \overline{E}', \tau, \text{exp})$), we need a number of steps that, in the worst case, are linear with the length of \overline{E} and exponential with the size of φ (resp. exp). However, if a *backward* approach is used as in [17], the same functions can be computed with a number of steps that is linear with both the length of \overline{E} and the size of φ (resp. exp).

4.2 Statistical error

We provide an algorithm for the evaluation of a *confidence interval* CI on the estimation of the value of a distance expression exp . This means that, given exp , a nominal evolution sequence \mathcal{S} , a perturbation \mathbf{p} , two time steps τ and τ' , and a coverage probability α , the probability that the real value $\llbracket \text{exp} \rrbracket_{\mathcal{S}, \mathcal{S}_{1 \llcorner (\mathbf{p})}, \tau}^{\tau'}$ of the distance is in CI is at least α .

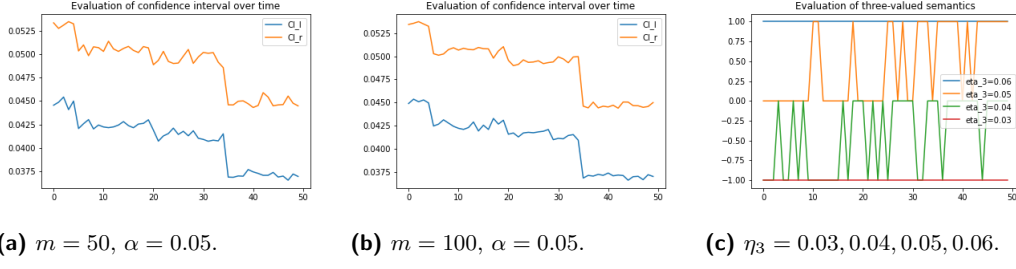
We start by evaluating the confidence intervals on $\mathbf{W}(\mathbf{m}_\tau^\ell)(\mu, \nu)$, obtained by applying the *empirical bootstrap* method [19, 20]:

1. Generate m bootstrap samples for μ and ν : these are obtained by drawing with replacement a sample of size N from the elements of the original sampling of μ , and one of size ℓN from those for ν . Let μ_1, \dots, μ_m and ν_1, \dots, ν_m the resulting bootstrap distributions.
2. Apply the procedure WASS m -times to evaluate the Wasserstein distances between the bootstrap distributions. Let W_1, \dots, W_m be the resulting bootstrap distances.
3. Evaluate the mean of the bootstrap distances $\overline{W} = \sum_{i=1}^m W_i / m$.
4. Evaluate the standard error $SE = (\sum_{i=1}^m (W_i - \overline{W})^2 / (m - 1))^{1/2}$.
5. Let $\text{CI} = \overline{W} \pm z_{1-\frac{\alpha}{2}} SE$, with $z_{1-\frac{\alpha}{2}}$ the $1 - \frac{\alpha}{2}$ quantile of the standard normal distribution.

► **Remark 18.** In [45] the *bias-corrected, accelerated percentile intervals* (BC_α) is used. We chose to use the empirical bootstrap method to find a balance between accuracy and computational complexity. Empirical bootstrap can be subject to bias in the samples, and more accurate techniques, like BC_α , were proposed [16]. However, to reach the desired accuracy with the BC_α method, it is necessary to use $m \geq \mathcal{O}(1000)$ bootstrap samples. Given the cost $\mathcal{O}(\ell N \log(\ell N))$ of a single evaluation of \mathbf{W} , and considering that in our formulae this distance is evaluated thousands of times, this approach would be computationally unfeasible. In our examples, $m \leq 100$ is sufficient to obtain reasonable confidence intervals (Example 19).

The evaluation of the confidence interval for the Wasserstein distance is then extended to distance expressions: once we have determined the bounds of the confidence intervals of the sub-expressions occurring in exp , the interval of exp is obtained by applying the function defining the evaluation of exp to them. For instance, if $\text{exp} = \max(\text{exp}_1, \text{exp}_2)$, $\text{CI}_{\text{exp}_1} = (l_1, r_1)$, and $\text{CI}_{\text{exp}_2} = (l_2, r_2)$, then $\text{CI}_{\text{exp}} = (\max\{l_1, l_2\}, \max\{r_1, r_2\})$.

► **Example 19.** In Figure 4 we report the 95% confidence intervals for $\llbracket \text{exp}_1 \rrbracket_{\mathcal{S}, \mathcal{S}_{1 \llcorner (\mathbf{p})}, 0}^{\tau'}$, where $\tau' \in [0, 50]$, with exp_1 and \mathbf{p} as in Example 17. The intervals in Figure 4a have been obtained by means of $m = 50$ bootstrap samplings, whereas for those in Figure 4b we used $m = 100$. In the former case, the maximal width of the interval is $9.39 \cdot 10^{-3}$, with an average width of $8.07 \cdot 10^{-3}$; in the latter case, those number become, respectively, $9.03 \cdot 10^{-3}$ and $7.93 \cdot 10^{-3}$.



■ **Figure 4** Confidence intervals of exp_1 , and three-valued evaluation of φ_{η_3} , over $[0, 50]$.

A three-valued semantics for RobTL. Given the presence of errors in the evaluation of expressions we extend our model checking algorithm with the possibility to assign a three-valued semantics to formulae. Alongside the classical Boolean evaluations true (\top) and false (\perp), a RobTL formula can assume the value *unknown* (Ψ). Intuitively, unknown is generated by the comparison between the distance and the chosen threshold in atomic propositions: if the threshold η does not lie in the confidence interval of the evaluation of the distance, then the formula will evaluate to \top or \perp according to the validity of the relation $\bowtie \eta$. Conversely, if η belongs to the confidence interval, then the atomic proposition evaluates to Ψ , since the validity of the relation $\bowtie \eta$ may depend on the particular samples obtained in the simulation.

Starting from atomic propositions, the three-valued semantics is extended to the Boolean operators via truth tables in the standard way [29, 48]. Then, we assign a three-valued semantics to RobTL formulae via the satisfaction function $\Omega_S: L \times [0, \mathfrak{h}] \rightarrow \{\top, \Psi, \perp\}$, defined inductively on the structure of RobTL formulae, starting from atomic propositions as follows:

$$\begin{aligned} \Omega_S(\top, \tau) &= \top \\ \Omega_S(\Delta(\text{exp}, \mathfrak{p}) \bowtie \eta, \tau) &= \begin{cases} \Psi & \text{if } \eta \in \text{CI}_{\text{exp}} \\ \mathcal{S}, \tau \models \Delta(\text{exp}, \mathfrak{p}) \bowtie \eta & \text{otherwise.} \end{cases} \\ \Omega_S(\neg \varphi, \tau) &= \neg \Omega_S(\varphi, \tau) \\ \Omega_S(\varphi_1 \wedge \varphi_2, \tau) &= \Omega_S(\varphi_1, \tau) \wedge \Omega_S(\varphi_2, \tau) \\ \Omega_S(\varphi_1 \mathcal{U}^I \varphi_2, \tau) &= \bigvee_{\tau' \in I} \left(\Omega_S(\varphi_2, \tau') \wedge \bigwedge_{\tau'' \in I, \tau'' < \tau'} \Omega_S(\varphi_1, \tau'') \right). \end{aligned}$$

The algorithm for the evaluation of function Ω_S is obtained in a straightforward manner from the Boolean evaluation (Algorithm 6).

► **Example 20.** Consider the formula $\varphi_{\eta_3} = \Delta(\text{exp}_1, \mathfrak{p}) \leq \eta_3$ for exp_1 and \mathfrak{p} as in Example 17. In Figure 4c we report the variation of the evaluation of $\Omega_S(\varphi_{\eta_3}, \tau')$ with respect to $\tau' \in [0, 50]$ and $\eta_3 \in \{0.03, 0.04, 0.05, 0.06\}$, where we let $\top \mapsto 1$, $\Psi \mapsto 0$, and $\perp \mapsto -1$. The plot confirms the validity of the empirical tuning of parameter η_3 that we carried out in Example 17.

5 Concluding remarks

The term robustness is used in several contexts, from control theory [50] to biology [28], and not always with the same meaning. Since our objective was to provide a formal tool for the verification of general robustness properties, we limit ourselves to recall that, in the context of CPS, we can distinguish five categories of robustness [24]:

- (i) input/output robustness;
- (ii) robustness with respect to system parameters;
- (iii) robustness in real-time system implementation;
- (iv) robustness due to unpredictable environment;
- (v) robustness to faults.

Our framework is designed for properties of type (iv), and we plan to extend it to the others.

[49] presents a PCTL statistical model checker based on stratified sampling. This allows for the generation of negatively correlated samples, thus considerably reducing the number of samples needed to obtain confident verdicts, provided the PCTL formulae are of a particular form. While direct comparison of the two algorithms would not be meaningful given the disparity in the logics, we will study the use of stratified sampling in our model checker.

In [1] the model of discrete time stochastic hybrid systems is introduced and used to formalise finite-horizon probabilistic reachability problems. Specifically, maximal probabilities of reaching (and maintaining) a safe set of states are considered. There are two main differences in between this model and the evolution sequence model that we would like to highlight:

- The purely data-driven characterisation of systems behaviour of the evolution sequence model has two crucial consequences. The first consequence is that the specification of the behaviour of the agent and that of the environment are independent, and there is no need to specify a system's state space, as opposed to what happens with the model in the proposed paper. The second consequence is that the behaviour of the system is not given by a set of traces/trajectories, but by the combination of their effects.
- The paper [1] only presents the analysis of probabilistic reachability properties, based on the evaluation of the desired safety property on each single trace of the system and the consequent computation of the total probability of executing those traces. In this paper, we introduce a logic that allows us to specify robustness properties based on the evaluation of distances between the behaviour of two different systems, the nominal and the perturbed one.

We plan to apply our framework to the analysis of biological systems. Some quantitative temporal logics have already been proposed in that setting [21,36,37] to capture some notions of robustness in system biology [5,28,32,41,42]. We are confident that the use of RobTL and evolution sequences can lead to new results, as shown in the preliminary work [13]. Moreover, we will apply our work to Medical CPS. In this context, statistical inference and learning methods have been combined in the synthesis of controllers, in order to deal with uncertainties [33]. The idea is then to use our tool to test the obtained controllers and verify their robustness against uncertainties.

Finally, we plan to apply our work to the evaluation of the effectiveness of digital twins [26]. To this end, we will enrich STARK with a special construct, similar to perturbations, that will allow us to model the communications, and their effects, between the digital and the real-world (perturbed) twin in a concise, clean, fashion. A preliminary result in this direction can be found in [8].

References

- 1 Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Autom.*, 44(11):2724–2734, 2008. doi:10.1016/J.AUTOMATICA.2008.03.027.
- 2 Erika Ábrahám and Borzoo Bonakdarpour. HyperPCTL: A temporal logic for probabilistic hyperproperties. In *Proceedings of QEST 2018*, volume 11024 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2018. doi:10.1007/978-3-319-99154-2_2.

- 3 Shiraj Arora, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, and Danny Bøgsted Poulsen. Statistical model checking for probabilistic hyperproperties of real-valued signals. In *Proceedings of SPIN 2022*, volume 13255 of *Lecture Notes in Computer Science*, pages 61–78. Springer, 2022. doi:10.1007/978-3-031-15077-7_4.
- 4 Christel Baier. Probabilistic model checking. In Javier Esparza, Orna Grumberg, and Salomon Sickert, editors, *Dependable Software Systems Engineering*, volume 45 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 1–23. IOS Press, 2016. doi:10.3233/978-1-61499-627-9-1.
- 5 Naama Barkai and Stanislas Leibler. Robustness in simple biochemical networks. *Nature*, 387:913–917, 1997. doi:10.1038/43199.
- 6 Vladimir I. Bogachev. *Measure Theory, vol. 2.*. Measure Theory. Springer-Verlag, Berlin/Heidelberg, 2007. doi:10.1007/978-3-540-34514-5.
- 7 Christos G. Cassandras, John Lygeros, and (eds.). *Stochastic Hybrid Systems*. Number 24 in Control Engineering. CRC Press, Boca Raton, 1st edition, 2007. doi:10.1201/9781315221625.
- 8 Valentina Castiglioni, Ruggero Lanotte, Michele Loreti, and Simone Tini. Evaluating the effectiveness of digital twins through statistical model checking with feedback and perturbations. In *Proceedings of FMICS 2024*, Lecture Notes in Computer Science. Springer, 2024.
- 9 Valentina Castiglioni, Michele Loreti, and Simone Tini. STARK: A Software Tool for the Analysis of Robusness in the unKnown environment. Software, swbId: swb:1:dir:ddfb418d5a080b8e83323a1b2c38d9f7065e2554 (visited on 2024-08-21). URL: <https://github.com/quasylab/jspear/tree/working>.
- 10 Valentina Castiglioni, Michele Loreti, and Simone Tini. How adaptive and reliable is your program? In *Proceedings of FORTE 2021*, volume 12719 of *LNCS*, pages 60–79. Springer, 2021. doi:10.1007/978-3-030-78089-0_4.
- 11 Valentina Castiglioni, Michele Loreti, and Simone Tini. A framework to measure the robustness of programs in the unpredictable environment. *Log. Methods Comput. Sci.*, 19(3), 2023. doi:10.46298/LMCS-19(3:2)2023.
- 12 Valentina Castiglioni, Michele Loreti, and Simone Tini. STARK: A Software Tool for the Analysis of Robustness in the unKnown environment. In *Proceedings of COORDINATION 2023*, volume 13908 of *Lecture Notes in Computer Science*, pages 115–132. Springer, 2023. doi:10.1007/978-3-031-35361-1_6.
- 13 Valentina Castiglioni, Michele Loreti, and Simone Tini. Bio-STARK: A tool for the time-point robustness analysis of biological systems. In *Proceedings of CMSB 2024*, Lecture Notes in Computer Science. Springer, 2024. To appear.
- 14 Valentina Castiglioni, Michele Loreti, and Simone Tini. STARK: A tool for the analysis of CPSs robustness. *Science of Computer Programming*, 236:103134, 2024. doi:10.1016/j.scico.2024.103134.
- 15 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
- 16 Thomas J. DiCiccio and Bradley Efron. Bootstrap confidence intervals. *Statistical Science*, 11(3):189–228, 1996. doi:10.1214/ss/1032280214.
- 17 Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 264–279, Berlin, Heidelberg, 2013. Springer.
- 18 Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proceedings of FORMATS 2010*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010. doi:10.1007/978-3-642-15297-9_9.
- 19 Bradley Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. doi:10.1214/aos/1176344552.
- 20 Bradley Efron. Nonparametric standard errors and confidence intervals. *Canadian Journal of Statistics*, 9(2):139–158, 1981. doi:10.2307/3314608.

- 21 François Fages and Aurélien Rizk. On temporal logic constraint solving for analyzing numerical data time series. *Theor. Comput. Sci.*, 408(1):55–65, 2008. doi:10.1016/j.tcs.2008.07.004.
- 22 Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009. doi:10.1016/j.tcs.2009.06.021.
- 23 Olivier P. Faugeras and Ludeger Rüschemdorf. Risk excess measures induced by hemi-metrics. *Probability, Uncertainty and Quantitative Risk*, 3:6, 2018. doi:10.1186/s41546-018-0032-0.
- 24 Martin Fränzle, James Kapinski, and Pavithra Prabhakar. Robustness in cyber-physical systems. *Dagstuhl Reports*, 6(9):29–45, 2016.
- 25 Dieter Gollmann, Pavel Gurikov, Alexander Isakov, Marina Krotofil, Jason Larsen, and Alexander Winnicki. Cyber-Physical Systems Security: Experimental Analysis of a Vinyl Acetate Monomer Plant. In *Proceedings of CPSS 2015*, pages 1–12. ACM, 2015.
- 26 Michael Grieves and John Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, pages 85–113. Springer, 2017. doi:10.1007/978-3-319-38756-7_4.
- 27 Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *Proceedings of HSCC 2000*, volume 1790 of *LNCS*, pages 160–173, 2000. doi:10.1007/3-540-46430-1_16.
- 28 Hiroaki Kitano. Towards a theory of biological robustness. *Molecular Systems Biology*, 3(1):137, 2007. doi:10.1038/msb4100179.
- 29 Stephen Cole Kleene. *Introduction to Metamathematics*. Princeton, NJ, USA: North Holland, 1952. doi:10.2307/2268620.
- 30 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Proceedings of SFM 2007*, volume 4486 of *LNCS*, pages 220–270. Springer, 2007. doi:10.1007/978-3-540-72522-0_6.
- 31 Ruggero Lanotte, Massimo Merro, Andrei Munteanu, and Simone Tini. Formal impact metrics for cyber-physical attacks. In *Proceedings of CSF 2021*, pages 1–16. IEEE, 2021. doi:10.1109/CSF51468.2021.00040.
- 32 Lucia Nasti, Roberta Gori, and Paolo Milazzo. Formalizing a notion of concentration robustness for biochemical networks. In *Proceedings of STAF 2018*, volume 11176 of *LNCS*, pages 81–97. Springer, 2018. doi:10.1007/978-3-030-04771-9_8.
- 33 Nicola Paoletti, Kin Sum Liu, Hongkai Chen, Scott A. Smolka, and Shan Lin. Data-driven robust control for a closed-loop artificial pancreas. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 17(6):1981–1993, 2020. doi:10.1109/TCBB.2019.2912609.
- 34 Svetlozar T. Rachev, Lev B. Klebanov, Stoyan V. Stoyanov, and Frank J. Fabozzi. *The Methods of Distances in the Theory of Probability and Statistics*. Springer, 2013.
- 35 Ragnathan Rajkumar, Insup Lee, Lui Sha, and John A. Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of DAC 2010*, pages 731–736. ACM, 2010. doi:10.1145/1837274.1837461.
- 36 Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. A general computational method for robustness analysis with applications to synthetic gene networks. *Bioinform.*, 25(12), 2009. doi:10.1093/bioinformatics/btp200.
- 37 Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures. *Theor. Comput. Sci.*, 412(26):2827–2839, 2011. doi:10.1016/j.tcs.2010.05.008.
- 38 Matthias Rungger and Paulo Tabuada. A notion of robustness for cyber-physical systems. *IEEE Trans. Autom. Control.*, 61(8):2108–2123, 2016.
- 39 Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *Proceedings of CAV 2005*, volume 3576 of *LNCS*, pages 266–280. Springer, 2005. doi:10.1007/11513988_26.
- 40 Ali Shahrokni and Robert Feldt. A systematic review of software robustness. *Information and Software Technology*, 55(1):1–17, 2013. doi:10.1016/j.infsof.2012.06.002.

- 41 Guy Shinar and Martin Feinberg. Structural sources of robustness in biochemical reaction networks. *Science*, 327(5971):1389–1391, 2010. doi:10.1126/science.1183372.
- 42 Guy Shinar and Martin Feinberg. Design principles for robust biochemical reaction networks: what works, what cannot work, and what might almost work. *Mathe. Biosci*, 231(1):39–48, 2011. doi:10.1016/j.mbs.
- 43 Eduardo D. Sontag. *Input to State Stability: Basic Concepts and Results*, pages 163–220. Springer, 2008. doi:10.1007/978-3-540-77653-6_3.
- 44 Bharath K. Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernard Schölkopf, and Gert R. G. Lanckriet. On the empirical estimation of integral probability metrics. *Electronic Journal of Statistics*, 6:1550–1599, 2021. doi:10.1214/12-EJS722.
- 45 David Thorsley and Eric Klavins. Model reduction of stochastic processes using Wasserstein pseudometrics. In *2008 American Control Conference*, pages 1374–1381. IEEE, 2008. doi:10.1109/ACC.2008.4586684.
- 46 David Thorsley and Eric Klavins. Approximating stochastic biochemical processes with Wasserstein pseudometrics. *IET Syst. Biol.*, 4(3):193–211, 2010. doi:10.1049/iet-syb.2009.0039.
- 47 Leonid N. Vaserstein. Markovian processes on countable space product describing large systems of automata. *Probl. Peredachi Inf.*, 5(3):64–72, 1969.
- 48 Ludovica Luisa Vissat, Michele Loreti, Laura Nenzi, Jane Hillston, and Glenn Marion. Analysis of spatio-temporal properties of stochastic systems using TSTL. *ACM Trans. Model. Comput. Simul.*, 29(4):20:1–20:24, 2019. doi:10.1145/3326168.
- 49 Yu Wang, Nima Roohi, Matthew West, Mahesh Viswanathan, and Geir E. Dullerud. Statistical verification of PCTL using antithetic and stratified samples. *Formal Methods Syst. Des.*, 54(2):145–163, 2019. doi:10.1007/s10703-019-00339-8.
- 50 Kemin Zhou and John C. Doyle. *Essentials of Robust Control*. Prentice-Hall, 1997.

A The Wasserstein hemimetric

Given a (pseudo-, hemi-)metric space (Ω, m) , the (pseudo-, hemi-)metric m induces a natural topology over Ω , namely the topology generated by the open ε -balls, for $\varepsilon > 0$, $B_m(\omega, \varepsilon) = \{\omega' \in \Omega \mid m(\omega, \omega') < \varepsilon\}$. We can then naturally obtain the Borel σ -algebra over Ω from this topology.

In this paper we are interested in defining a *hemimetric on distributions*. To this end we will make use of the Wasserstein lifting [47] whose definition is based on the following notions and results. Given a set Ω and a topology T on Ω , the topological space (Ω, T) is said to be *completely metrisable* if there exists at least one metric m on Ω such that (Ω, m) is a complete metric space and m induces the topology T . A *Polish space* is a separable completely metrisable topological space. In particular, we recall that:

- (i) \mathbb{R} is a Polish space; and
- (ii) every closed subset of a Polish space is in turn a Polish space.

Moreover, for any $n \in \mathbb{N}$, if $\Omega_1, \dots, \Omega_n$ are Polish spaces, then the Borel σ -algebra on their product coincides with the product σ -algebra generated by their Borel σ -algebras, namely

$$\mathcal{B}\left(\prod_{i=1}^n \Omega_i\right) = \prod_{i=1}^n \mathcal{B}(\Omega_i).$$

(This is proved, e.g., in [6] as Lemma 6.4.2 whose hypothesis are satisfied by Polish spaces since they are second countable.) These properties of Polish spaces are interesting for us since they guarantee that all the distributions we consider in this paper are Radon measures

and, thus, the Wasserstein lifting is well-defined on them. For this reason, we also directly present the Wasserstein hemimetric by considering only distributions on Borel sets.

► **Definition 21** (Wasserstein hemimetric). *Consider a Polish space Ω and let m be a hemimetric on Ω . For any two distributions μ and ν on $(\Omega, \mathcal{B}(\Omega))$, the Wasserstein lifting of m to a distance between μ and ν is defined by*

$$\mathbf{W}(m)(\mu, \nu) = \inf_{\mathfrak{w} \in \mathfrak{W}(\mu, \nu)} \int_{\Omega \times \Omega} m(\omega, \omega') d\mathfrak{w}(\omega, \omega')$$

where $\mathfrak{W}(\mu, \nu)$ is the set of the couplings of μ and ν , namely the set of joint distributions \mathfrak{w} over the product space $(\Omega \times \Omega, \mathcal{B}(\Omega \times \Omega))$ having μ and ν as left and right marginal, respectively, namely $\mathfrak{w}(\mathbb{A} \times \Omega) = \mu(\mathbb{A})$ and $\mathfrak{w}(\Omega \times \mathbb{A}) = \nu(\mathbb{A})$, for all $\mathbb{A} \in \mathcal{B}(\Omega)$.

Despite the original version of the Wasserstein distance being defined on a metric on Ω , the Wasserstein hemimetric given above is well-defined. We refer the interested reader to [23] and the references therein for a formal proof of this fact. In particular, the Wasserstein hemimetric is given in [23] as Definition 7 (considering the compound risk excess metric defined in Equation (31) of that paper), and Proposition 4 in [23] guarantees that it is indeed a well-defined hemimetric on $\Pi(\Omega, \mathcal{B}(\Omega))$. Moreover, Proposition 6 in [23] guarantees that the same result holds for the hemimetric $m(x, y) = \max\{y - x, 0\}$.

B The algorithms

In this section we report the algorithms described in Section 4.

■ **Algorithm 1** Simulation of a evolution sequence.

```

1: function SIM( $\mu, N, k$ )
2:    $i \leftarrow 0$ 
3:    $E_0 \leftarrow \text{SAMPLEDISTR}(\mu, N)$ 
4:   while  $i < k$  do
5:      $E_{i+1} \leftarrow \emptyset$ 
6:     for  $\mathbf{d} \in E_i$  do
7:        $E_{i+1} \leftarrow \text{SIMSTEP}(\mathbf{d}), E_{i+1}$ 
8:     end for
9:      $i \leftarrow i + 1$ 
10:  end while
11:  return  $E_0, \dots, E_k$ 
12: end function

```

■ **Algorithm 2** Computation of the effect of a perturbation.

```

1: function SIMPER( $\overline{E}, p, \tau, \ell$ )
2:    $\forall i < \tau. E'_i \leftarrow E_i$ 
3:    $E'_\tau \leftarrow \ell \cdot E_\tau$ 
4:    $i \leftarrow \tau$ 
5:   while  $i < k$  do
6:      $f \leftarrow \text{effect}(p)$ 
7:      $p \leftarrow \text{next}(p)$ 
8:     for  $d \in E'_i$  do
9:        $d' \leftarrow \text{SAMPLE}(f(d))$ 
10:       $E'_{i+1} \leftarrow E'_{i+1}, \text{SIMSTEP}(d')$ 
11:    end for
12:     $i \leftarrow i + 1$ 
13:  end while
14:  return  $E'_0, \dots, E'_k$ 
15: end function

```

■ **Algorithm 3** Evaluation of the Wasserstein distance.

```

1: function WASS( $E_1, E_2, op, \rho$ )
2:    $(d_1^1, \dots, d_1^N) \leftarrow E_1$ 
3:    $(d_2^1, \dots, d_2^{\ell N}) \leftarrow E_2$ 
4:    $\forall j : (1 \leq j \leq N) : \omega_j \leftarrow \rho(d_1^j)$ 
5:    $\forall h : (1 \leq h \leq \ell N) : \nu_h \leftarrow \rho(d_2^h)$ 
6:   re index  $\{\omega_j\}$  s.t.  $\omega_j \leq \omega_{j+1}$ 
7:   re index  $\{\nu_h\}$  s.t.  $\nu_h \leq \nu_{h+1}$ 
8:   if  $op = <$  then
9:     return  $\frac{1}{\ell N} \sum_{h=1}^{\ell N} \max\{\nu_h - \omega_{\lceil \frac{h}{\ell} \rceil}, 0\}$ 
10:  else
11:    return  $\frac{1}{\ell N} \sum_{h=1}^{\ell N} \max\{\omega_{\lceil \frac{h}{\ell} \rceil} - \nu_h, 0\}$ 
12:  end if
13: end function

```

■ **Algorithm 4** Evaluation of distance expressions.

```

1: function EVAL Expr( $\overline{E}, \overline{E}', \tau, \text{exp}$ )
2:   match exp
3:     with  $<^\rho$  :
4:       return WASS( $E_\tau, E'_\tau, <, \rho$ )
5:     with  $>^\rho$  :
6:       return WASS( $E_\tau, E'_\tau, >, \rho$ )
7:     with  $F^I$  exp :
8:       return  $\min_{i \in \tau+I} \{\text{EVAL Expr}(\overline{E}, \overline{E}', i, \text{exp})\}$ 
9:     with  $G^I$  exp :
10:      return  $\max_{i \in \tau+I} \{\text{EVAL Expr}(\overline{E}, \overline{E}', i, \text{exp})\}$ 
11:     with  $\text{exp}_1 \cup^{[\tau_1, \tau_2]} \text{exp}_2$  :
12:        $\forall i \in [\tau + \tau_1, \tau + \tau_2] d_i^2 \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', i, \text{exp}_2)$ 
13:        $\forall j \in [\tau + \tau_1, \tau + \tau_2] d_j^1 \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', j, \text{exp}_1)$ 
14:       return  $\min_{\tau+\tau_1 \leq i \leq \tau+\tau_2} \{\max\{d_i^2, \max_{0 \leq j < i} \{d_j^1\}\}\}$ 
15:     with  $\min(\text{exp}_1, \text{exp}_2)$  :
16:        $v_1 \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', \tau, \text{exp}_1)$ 
17:        $v_2 \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', \tau, \text{exp}_2)$ 
18:       return  $\min\{v_1, v_2\}$ 
19:     with  $\max(\text{exp}_1, \text{exp}_2)$  :
20:        $v_1 \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', \tau, \text{exp}_1)$ 
21:        $v_2 \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', \tau, \text{exp}_2)$ 
22:       return  $\max\{v_1, v_2\}$ 
23:     with  $\sum_{i \in K} w_i \cdot \text{exp}_i$  :
24:        $v_i \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', \tau, \text{exp}_i)$ 
25:       return  $\sum_{i \in K} w_i \cdot v_i$ 
26:     with  $\sigma(\text{exp}, \bowtie \zeta)$  :
27:        $v \leftarrow \text{EVAL Expr}(\overline{E}, \overline{E}', \tau, \text{exp})$ 
28:       if  $v \bowtie \zeta$  then
29:         return 0
30:       else
31:         return 1
32:       end if
33: end function

```

■ **Algorithm 5** Checking the satisfaction of a formula.

```

1: function SAT( $\mu, \tau, \varphi, N, \ell$ )
2:    $k \leftarrow \text{HORIZON}(\varphi)$ 
3:    $\overline{E} \leftarrow \text{SIM}(\mu, N, k)$ 
4:   return EVAL( $\overline{E}, \tau, \varphi, \ell$ )
5: end function

```

■ **Algorithm 6** Evaluation of RobTL formulae.

```

1: function EVAL( $\overline{E}, \tau, \varphi, \ell$ )
2:   match  $\varphi$ 
3:     with  $\varphi = \top$  :
4:       return true
5:     with  $\varphi = \Delta(\mathbf{exp}, \mathbf{p}) \bowtie \eta$  :
6:        $\overline{E}' \leftarrow \text{SIMPER}(\overline{E}, \mathbf{p}, \tau, \ell)$ 
7:        $v \leftarrow \text{EVALEXPR}(\overline{E}, \overline{E}', \tau, \mathbf{exp})$ 
8:       return  $v \bowtie \eta$ 
9:     with  $\varphi = \neg\varphi_1$  :
10:      return  $\neg\text{EVAL}(\overline{E}, \tau, \varphi_1, \ell)$ 
11:    with  $\varphi_1 \wedge \varphi_2$  :
12:      return  $\text{EVAL}(\overline{E}, \tau, \varphi_1, \ell) \wedge \text{EVAL}(\overline{E}, \tau, \varphi_2, \ell)$ 
13:    with  $\varphi_1 \mathcal{U}^{[\tau_1, \tau_2]} \varphi_2$  :
14:       $j \leftarrow \tau + \tau_1$ 
15:       $i \leftarrow j - 1$ 
16:       $\text{res} \leftarrow \textit{false}$ 
17:       $\text{res}' \leftarrow \textit{true}$ 
18:      while  $j \leq \tau + \tau_2 \wedge \neg\text{res} \wedge \text{res}'$  do
19:         $\text{res} \leftarrow \text{EVAL}(\overline{E}, j, \varphi_2, \ell)$ 
20:         $i \leftarrow i + 1$ 
21:         $\text{res}' \leftarrow \text{EVAL}(\overline{E}, i, \varphi_1, \ell)$ 
22:         $j \leftarrow j + 1$ 
23:      end while
24:      return  $\text{res}$ 
25: end function

```

Effect Semantics for Quantum Process Calculi

Lorenzo Ceragioli ✉ 

IMT School for Advanced Studies Lucca, Italy

Fabio Gadducci ✉ 

University of Pisa, Italy

Giuseppe Lomurno ✉ 

University of Pisa, Italy

Gabriele Tedeschi ✉ 

University of Pisa, Italy

Abstract

The development of quantum communication protocols sparked the interest in quantum extensions of process calculi and behavioural equivalences, but defining a bisimilarity that matches the observational properties of a quantum-capable system is a surprisingly difficult task. The two proposals explicitly addressing this issue, qCCS and lqCCS, do not define an algorithmic verification scheme: the bisimilarity of two processes is proven by comparing their behaviour under all input states. We introduce a new semantic model based on effects, i.e. probabilistic predicates on quantum states that represent their observable properties. We define and investigate the properties of effect distributions and effect labelled transition systems (eLTSs), generalizing probability distributions and probabilistic labelled transition systems (pLTSs), respectively. As a proof of concept, we provide an eLTS-based semantics for a minimal quantum process algebra, which we prove sound and complete with respect to the observable probabilistic behaviour of quantum processes. To the best of our knowledge, ours is the first algorithmically verifiable proposal that abides to the properties of quantum theory.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Process calculi; Theory of computation → Operational semantics

Keywords and phrases Quantum process calculi, probabilistic LTSs, effect LTSs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.16

Funding This study was carried out within the National Centre on HPC, Big Data and Quantum Computing - SPOKE 10 (Quantum Computing) and received funding from the European Union Next-GenerationEU - National Recovery and Resilience Plan (NRRP) – MISSION 4 COMPONENT 2, INVESTMENT N. 1.4 – CUP N. I53C22000690001.

1 Introduction

Recent years have seen a flourishing development of quantum technologies for computer science, in the form of *quantum computation* and *quantum communication*. Both of them exploit quantum phenomena like superposition and entanglement: the former is interested in harvesting the (supposedly) higher computational power of quantum computers, while the latter strives to achieve secure and reliable communication, featuring solutions for key distribution [31], cryptographic coin tossing [2], direct communication [28], and private information retrieval [14]. Protocols like BB84 QKD [2] are *unconditionally secure* [29], meaning that they are protected against all physically possible attackers. Quantum communication also promises to allow linking multiple computers via the *Quantum Internet* [5, 35], therefore providing quantum algorithms with large enough memories for practical applications.

Despite the rich theory and the potential applications, there is no accepted standard to model and verify quantum concurrent systems and protocols. Numerous works [25, 15, 12, 34, 7] rely on *process calculi*, an algebraic formalism that has been successfully applied to



© Lorenzo Ceragioli, Fabio Gadducci, Giuseppe Lomurno, and Gabriele Tedeschi; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 16; pp. 16:1–16:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

classical protocols and concurrent systems. The semantics of a calculus is given by means of a *labelled transition system* (LTS), i.e. a triple (S, Act, \rightarrow) with S a set of states, Act a set of actions, and \rightarrow a transition relation that specifies how states evolve. The standard equivalence for LTSs is *bisimilarity*, the largest relation on states that is “stable” for \rightarrow , meaning that bisimilar states evolve with the same action in bisimilar states.

There have been several attempts [24, 8, 10, 9, 7] to adapt known techniques to the quantum setting, mainly in terms of *probabilistic LTSs* (pLTSs) $(Conf, Act, \rightarrow)$, where $Conf = \mathcal{H} \times S$ is a set of *configurations* $\langle \rho, P \rangle$ composed by a quantum state ρ (an element of a Hilbert space \mathcal{H}) and a process P , and $\rightarrow \subseteq Conf \times Act \times \mathcal{D}(Conf)$ with $\mathcal{D}(Conf)$ probability distributions of configurations. This approach led to a plethora of different bisimilarities, most of them unsatisfactory since they distinguish processes that are deemed indistinguishable by the prescriptions of quantum theory [8, 23, 13]. Moreover, only configuration bisimilarity is directly considered in these works. Two processes P and Q are instead deemed bisimilar if and only if for any quantum state ρ the configurations $\langle \rho, P \rangle$ and $\langle \rho, Q \rangle$ are bisimilar. Assessing bisimilarity of processes thus requires comparing infinitely many pLTSs (one for each quantum state), and algorithmic verification is still missing. In [7], the root of these problems is identified in the peculiarities of the semantic model described above, a non-deterministic pLTS made of quantum states and processes.

We propose *effect labelled transition systems* (eLTSs) as a new semantic model for quantum systems, generalizing pLTSs. In physics, *effects* represent the observable behaviour of quantum states, as they model atomic experiments that you can perform on a quantum system. Building on them allows us to express the correct observable properties of quantum processes. *Effect distributions* generalize probability distributions by using effects as weights, and the transition relation of an eLTS associates states with effect distributions. We study effect distributions and eLTSs, either generalizing the known results on probabilistic systems when possible, or proving they do not hold otherwise. We explore different notions of bisimilarity, namely Aczel-Mendler and Larsen-Skou, and show that they disagree on which quantum processes should be bisimilar, even if they coincide in the probabilistic case. Then, we consider two correctness criteria for quantum bisimilarity, through which we show that a Larsen-Skou-style bisimilarity is adequate for comparing quantum systems, as it is correct and complete with respect to the observable probabilistic behaviour of quantum protocols.

To assess our proposal, we define a *minimal quantum process algebra* (mQPA) featuring actions, restriction, synchronization, non-determinism, parallel composition, destructive measurements, and unitary transformations, and we equip it with two different semantics: a stateful Schrödinger-style semantics that given a quantum state returns a pLTS representing the observable behaviour of the system; and a Heisenberg-style semantics in the form of an eLTS that is independent of the quantum input, in the style of [20, 11]. We show that the Heisenberg eLTS is indeed the “symbolic” version of the Schrödinger pLTSs of the same system, thus proving bisimilarity just once for the Heisenberg-style semantics makes it automatically verified for all “ground” systems obtained by instantiating the quantum input. Notably, our notion of bisimilarity can be efficiently verified with standard techniques [22].

Synopsis. Section 2 provides some background on probability distributions and quantum theory. Section 3 introduces effect distributions and eLTSs, investigating their properties and comparing eLTS bisimilarities. Section 4 presents our minimal process algebra with both a stateful and a stateless semantics, which are proved to coincide. Finally, Section 5 compares related works and Section 6 draws our conclusions. The full proofs are in the Appendix.

2 Background

We recall some background on probability distributions, quantum computing, and effects, referring the reader to [18, 30] for further information.

2.1 Probability Distributions

A *probability (sub)distribution* over a set S is a function $\Delta: S \rightarrow [0, 1]$ such that $\sum_{s \in S} \Delta(s) \leq 1$. We write \mathcal{DS} for the set of finitely supported distributions over S , i.e. with $\Delta(s) = 0$ for all but a finite set of elements. We let \bar{s} be the point distribution $\bar{s}(s) = 1$.

Probability distributions form a *convex set* [4]: for any two distributions Δ, Θ and real $p \in [0, 1]$ there is a distribution $\Delta \oplus_p \Theta$ defined as $p \cdot \Delta + (1 - p) \cdot \Theta$. A function f between convex sets is convex if it preserves the \oplus_p operator, i.e. if $f(x_1 \oplus_p x_2) = f(x_1) \oplus_p f(x_2)$. We denote as $\mathbf{Conv}(X, Y)$ the set of convex functions between X and Y .

2.2 Quantum Computing

We assume a denumerable set of indexed qubits $\mathbf{Q} = \{q_0, q_1, \dots\}$, the quantum mechanical analogues of classical bits. The states of a qubit q are *unit vectors* $|\psi\rangle$ in the Hilbert space \mathcal{H}_q , i.e. column vectors in \mathbb{C}^2 such that $\langle \psi | \psi \rangle = 1$, with $\langle \psi |$ the conjugate transpose of $|\psi\rangle$, and $\langle \cdot | \cdot \rangle$ the usual *inner product*. The vectors $|0\rangle = (1, 0)^T$ and $|1\rangle = (0, 1)^T$ form an orthonormal basis of \mathbb{C}^2 , called the *computational basis*. Other important states are $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, which form the *Hadamard basis*.

The *Kronecker product* \otimes is defined as follows (we often write $|\psi\phi\rangle$ for $|\psi\rangle \otimes |\phi\rangle$)

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} \otimes L = \begin{bmatrix} x_{1,1}L & \cdots & x_{1,n}L \\ \vdots & \ddots & \vdots \\ x_{m,1}L & \cdots & x_{m,n}L \end{bmatrix}$$

Note that \otimes is not commutative. Given \mathcal{H}_q with $\{|\psi_i\rangle\}_{i \in I}$ one of its bases, and $\mathcal{H}_{q'}$ with $\{|\phi_j\rangle\}_{j \in J}$ one of its bases, we let $\mathcal{H}_q \otimes \mathcal{H}_{q'}$ be the Hilbert space with bases $\{|\psi_i\rangle \otimes |\phi_j\rangle\}_{(i,j) \in I \times J}$. A quantum register is a finite set of n qubits $Q \subseteq \mathbf{Q}$, representing composite physical systems. Its states are in $\mathcal{H}_Q = \bigotimes_{i=1, q_i \in Q}^{\infty} \mathcal{H}_{q_i} = \mathbb{C}^{2^n}$. Note that the Kronecker product is applied in an ordered manner, according to the indexing of \mathbf{Q} . The state of a quantum register $\mathcal{H}_{\{q, q'\}}$ is *separable* when it can be expressed as the tensor of two vectors of \mathcal{H}_q and $\mathcal{H}_{q'}$. Otherwise, it is *entangled*, like the Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

For each linear operator A on \mathcal{H}_Q , its *adjoint* A^\dagger is the unique linear operator such that $\langle \psi | A | \phi \rangle = \langle A^\dagger \psi | \phi \rangle$. A linear operator U is *unitary* when $UU^\dagger = U^\dagger U = \mathbb{I}$. In quantum physics, the evolution of a closed system is described by a unitary transformation: the state $|\psi\rangle$ at time t_0 is related to $|\psi'\rangle$ at time t_1 by a unitary operator U , which only depends on t_0 and t_1 , i.e. $|\psi'\rangle = U |\psi\rangle$. Accordingly, quantum computers allow the programmer to manipulate registers via unitaries like H , X , Z and $CNOT$, satisfying: $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$; $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$; $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$; $CNOT|10\rangle = |11\rangle$, $CNOT|11\rangle = |10\rangle$ and $CNOT|0\psi\rangle = |0\psi\rangle$ (all the other cases are defined by linearity).

Let Q and Q' be sets of qubits, we write $Q \uplus Q'$ for $Q \cup Q'$ when $Q \cap Q' = \emptyset$, and we allow composing a pair of states in \mathcal{H}_Q and $\mathcal{H}_{Q'}$ to obtain a state in $\mathcal{H}_{Q \uplus Q'}$. To preserve the ordering induced by the indices, we build on top of \otimes to define a partial commutative tensor product \boxtimes . We let \boxtimes be the operation that applies \otimes and then sorts the qubits according to their indices: $|\psi\rangle \boxtimes |\phi\rangle = \mathit{Sort}(|\psi\rangle \otimes |\phi\rangle) \in \mathcal{H}_{Q \uplus Q'}$, with Sort a unitary operator.

The density operator formalism puts together quantum systems and probability by considering mixed states, i.e. *probability sub-distributions of quantum states*. A point distribution $|\psi\rangle$ (called a pure state) is represented by the matrix $|\psi\rangle\langle\psi|$. In general, a probability distribution Δ is represented as the matrix $\rho = \sum_i \Delta(\psi_i) |\psi_i\rangle\langle\psi_i|$, known as its (*partial density operator*). Recall that a complex matrix N is called *positive semi-definite*, shortly positive, when $\langle\psi|N|\psi\rangle \geq 0$ for any $|\psi\rangle$. The *Löwner order* is the partial order defined by $L \sqsubseteq L'$ whenever $L' - L$ is positive. Given \mathcal{H}_Q of dimension d , the density operators over \mathcal{H}_Q coincide with the positive matrices in $\mathbb{C}^{d \times d}$ of trace smaller or equal to one, and we denote them as $DM_Q = \{ \rho \in \mathbb{C}^{d \times d} \mid \rho \sqsupseteq 0_Q, \text{tr}(\rho) \leq 1 \}$, where 0_Q is the $d \times d$ all-zero operator on \mathcal{H}_Q . Density operators form a convex set, meaning that for any real $p \in [0, 1]$ and any $\rho, \sigma \in DM_Q$, there is a *convex combination* $\rho \oplus_p \sigma \in DM_Q$ defined as $p\rho + (1-p)\sigma$. A function between convex sets is called convex if it preserves the \oplus_p operator.

Given \mathcal{H}_Q and $\mathcal{H}_{Q'}$ of dimensions n and m respectively, a *trace non-increasing superoperator* $\mathcal{E} : DM_Q \rightarrow DM_{Q'}$ is defined as $\mathcal{E}(\rho) = \sum_i K_i \rho K_i^\dagger$ for a set of operators $\{K_i \in \mathbb{C}^{m \times n}\}_{i=1, \dots, n \times m}$ (called *Kraus operators*), such that $\sum_i K_i^\dagger K_i \sqsubseteq \mathbb{I}_Q$. Superoperators model the evolution of mixed quantum states, and are closed with respect to composition. Any unitary transformation U is represented as the superoperator with Kraus decomposition $\{U\}$. The Kronecker product also defines composition of mixed states and superoperators on different quantum registers. We lift our commutative tensor product \boxtimes to density operators and to superoperators by reordering the qubits when needed.

Density operators can be used to describe the state of a subsystem of a composite quantum system. Given a $\rho \in DM_{Q \uplus Q'}$, the *reduced density operator* of Q , $\rho_Q = \text{tr}_{Q'}(\rho) \in DM_Q$, describes the state of Q , with $\text{tr}_{Q'}$ the *partial trace over Q'* , defined as the linear transformation such that $\text{tr}_{Q'}(|\psi\rangle\langle\psi'| \boxtimes |\phi\rangle\langle\phi'|) = |\psi\rangle\langle\psi'| \text{tr}(|\phi\rangle\langle\phi'|)$ for each $|\psi\rangle\langle\psi'| \in DM_Q$ and $|\phi\rangle\langle\phi'| \in DM_{Q'}$.

2.3 Quantum Effects

Quantum measurements allow describing systems that exchange information with the environment. Performing a measurement on a quantum register returns a probabilistic result and it either destroys or changes the qubits. We focus on destructive measurements.

The simplest kind of measurements are *quantum effects* (simply called effects in quantum textbooks [18]), i.e. yes-no tests over quantum systems. Each effect can be represented as a positive matrix smaller than the identity in the Löwner order. We denote the set of effects on the d -dimensional \mathcal{H}_Q as $Ef_Q = \{ E \in \mathbb{C}^{d \times d} \mid 0_Q \sqsubseteq E \sqsubseteq \mathbb{I}_Q \}$, where \mathbb{I}_Q is the $d \times d$ identity operator over the Hilbert space \mathcal{H}_Q . The probability of getting a “yes” outcome when measuring an effect E on a state ρ is $\text{tr}(E\rho)$, as given by the *Born rule*.

Density operators and effects are dual, as effects are isomorphic (via the Born rule) to the convex functions from the set of density operators to the probability interval.

► **Theorem 1** ([18]). $Ef_Q \cong \text{Conv}(DM_Q, [0, 1])$ through the isomorphism $E \mapsto \lambda\rho. \text{tr}(E\rho)$.

Effects can thus be seen as probabilities *parameterized* on an unknown quantum state.

Following this duality, to each superoperator $\mathcal{E} : DM_Q \rightarrow DM_{Q'}$ with Kraus operators $\{K_i\}$ corresponds a *dual superoperator* $\mathcal{Z} : Ef_{Q'} \rightarrow Ef_Q$ (note the inversion), whose Kraus operators are $\{K_i^\dagger\}$. The defining property of such superoperators is $\text{tr}(E \cdot \mathcal{E}(\rho)) = \text{tr}(\mathcal{Z}(E) \cdot \rho)$.

In general, a measurement with n different outcomes is a set $\{E_1, \dots, E_n\}$ of effects satisfying the *completeness* equation $\sum_{i=1}^n E_i = \mathbb{I}$. The probability of the i outcome occurring is again given by the Born rule $p_i = \text{tr}(E_i\rho)$.

As examples of measurements, consider M_{01} and M_{\pm} that project a qubit state into the elements of the computational and Hadamard basis, respectively, with M_{01} defined as $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ and M_{\pm} as $\{|+\rangle\langle +|, |-\rangle\langle -|\}$. Applying the measurement M_{01} on $|0\rangle$ returns the outcome associated with $|0\rangle\langle 0|$ with probability 1. When measuring instead $|+\rangle$ the same result occurs with probability $\frac{1}{2}$. For measurements over registers, we allow composing effects via the \boxtimes tensor product. Note that a measurement may measure only some of the qubits of a register, e.g. $\{|0\rangle\langle 0| \boxtimes \mathbb{I}, |1\rangle\langle 1| \boxtimes \mathbb{I}\}$ measures (in the computational basis) the first qubit.

3 Effect-Based Models

We generalize probability distributions and pLTSs to effect distributions and eLTSs, and we investigate which properties of probability distributions can be lifted to the quantum case. We adapt the two most used definitions of bisimilarity for pLTS to eLTS, namely, the *Aczel-Mendler* and *Larsen-Skou* bisimilarities. Even if the two coincide in the probabilistic case, this is not the same for eLTSs, and we argue that the latter is adequate for comparing the behaviour of quantum systems.

3.1 Effect Distribution

Given a set of qubits Q , we introduce effect distributions, i.e. functions associating each element of a given set X with some effect in Ef_Q .

► **Definition 2.** Let $Q \subseteq \mathbf{Q}$. The set of finite Ef_Q -(sub)distributions over a set X is

$$\mathcal{Q}_Q X = \left\{ \mathfrak{D} \in Ef_Q^X \mid \text{supp}(\mathfrak{D}) \text{ is finite and } \sum_{x \in \text{supp}(\mathfrak{D})} \mathfrak{D}(x) \sqsubseteq \mathbb{I}_Q \right\}$$

where $\text{supp}(\mathfrak{D})$ is the support of \mathfrak{D} , i.e. the set $\{x \in X \mid \mathfrak{D}(x) \neq 0_Q\}$. We say that a distribution \mathfrak{D} is full when $\sum_{x \in \text{supp}(\mathfrak{D})} \mathfrak{D}(x) = \mathbb{I}_Q$.

Effect distributions are a conservative generalization of probability distributions. More in detail, 1×1 positive matrices are isomorphic to real numbers, hence $\mathcal{Q}_{\emptyset} X$ coincides with the usual set of probability distributions $\mathcal{D}X$.

We represent effect distributions as sets of pairs $\mathfrak{D} = \{x_1 \triangleright E_1, x_2 \triangleright E_2, \dots, x_n \triangleright E_n\}$ with possibly repeated x_i , meaning $\mathfrak{D}(x) = \sum_{x_i=x} E_i$. For example, $\{x \triangleright E_1, x \triangleright E_2, y \triangleright E_3\}$ and $\{x \triangleright E_1 + E_2, y \triangleright E_3\}$ denote the same distribution.

► **Example 3.** Let $X = \{x, y\}$. The distribution $\mathfrak{D} = \{x \triangleright \frac{1}{2}, y \triangleright \frac{1}{2}\}$ is indeed just a uniform probability distribution, i.e. an effect distribution in the 1-dimensional Hilbert space \mathcal{H}_{\emptyset} .

Two more interesting effect distributions, in the two-dimensional Hilbert space $\mathcal{H}_{\{q_1\}}$, are $\mathfrak{G} = \{x \triangleright \frac{1}{2}\mathbb{I}, y \triangleright \frac{1}{2}\mathbb{I}\}$ and $\mathfrak{T} = \{x \triangleright |0\rangle\langle 0|, y \triangleright |1\rangle\langle 1|\}$. Both represent a measurement performed on q_1 : \mathfrak{G} returns the outcomes x and y with the same probability, regardless of the state of q_1 ; while \mathfrak{T} returns x when it observes $|0\rangle\langle 0|$ and y when it observes $|1\rangle\langle 1|$.

Since effects can be regarded as functions from states to probabilities, an effect distribution $\mathfrak{D} \in \mathcal{Q}_Q X$ denotes a function $\mathfrak{D}_{\downarrow} \in (\mathcal{D}X)^{DM_Q}$ associating a $\rho \in DM_Q$ with the probability distribution $\mathfrak{D}_{\downarrow\rho}$ such that $\mathfrak{D}_{\downarrow\rho}(x) = \text{tr}(\mathfrak{D}(x) \cdot \rho)$ for any $x \in X$. Hence, an effect distribution corresponds to the parameterized probabilistic outcome of performing a finite destructive measurement on some unknown input quantum state.

In particular, we have the following isomorphism (formally, a convex set isomorphism).

► **Theorem 4.** *Effect distributions correspond to all and only the parameterized sub-probability distributions that are convex and have an “overall” finite support.*

$$\mathcal{Q}_Q \cong \left\{ \mathfrak{D} \downarrow_{-} \in (\mathcal{D}X)^{DM_Q} \mid \mathfrak{D} \downarrow_{\rho_p \oplus \sigma} = (\mathfrak{D} \downarrow_{\rho}) \downarrow_p \oplus (\mathfrak{D} \downarrow_{\sigma}) \text{ and } \bigcup_{\rho \in DM_Q} \text{supp}(\mathfrak{D} \downarrow_{\rho}) \text{ is finite} \right\}$$

This isomorphism tells us that we can see effect distributions as measurements.

► **Example 5.** Consider \mathfrak{T} of Example 3 and the quantum input $\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +|$. The probability distribution $\mathfrak{T} \downarrow_{\rho}$ maps x to $\frac{3}{4}$ and y to $\frac{1}{4}$. Intuitively, $\mathfrak{T} \downarrow_{\rho}$ corresponds to the probabilistic outcome of performing the measurement \mathfrak{T} over a system in state ρ .

As for probabilities, we compose effect distributions via an effect-weighted sum, provided that they are defined over different qubits. This is a partial operation, being $E \boxtimes F$ defined only when E and F uses disjoint sets of qubits.

► **Definition 6.** *Given a family of Ef_Q -distributions $\{\mathfrak{D}_i\}_{i \in I}$ and effects $\{E_i\}_{i \in I}$ in $Ef_{Q'}$ where $Q \cap Q' = \emptyset$ and such that $\sum_{i \in I} E_i \sqsubseteq \mathbb{I}$, the weighted sum $\sum_{i \in I} E_i \boxtimes \mathfrak{D}_i$ is the $Ef_{Q \uplus Q'}$ -distribution defined as $(\sum_{i \in I} E_i \boxtimes \mathfrak{D}_i)(x) = \sum_{i \in I} E_i \boxtimes \mathfrak{D}_i(x)$.*

This composition coincides with the usual weighted sum of probability distributions if $Q = Q' = \emptyset$. Intuitively, \mathfrak{D} measures some qubits to choose between the distributions \mathfrak{D}_i (which in turn behave accordingly to the remaining qubits). We will sometimes write $E_1 \boxtimes \mathfrak{D}_1 + \dots + E_n \boxtimes \mathfrak{D}_n$ for $\sum_i E_i \boxtimes \mathfrak{D}_i$.

► **Example 7.** Take $\mathfrak{G}, \mathfrak{T}$ of Example 3. The $Ef_{\{q_1, q_2\}}$ -distribution $(|+\rangle\langle +| \boxtimes \mathfrak{G}) + (|-\rangle\langle -| \boxtimes \mathfrak{T})$ can be rewritten as $\{x \triangleright \mathbb{I} \otimes \frac{1}{2}|+\rangle\langle +|, y \triangleright \mathbb{I} \otimes \frac{1}{2}|+\rangle\langle +|, x \triangleright |0-\rangle\langle 0-|, y \triangleright |1-\rangle\langle 1-|\}$. Intuitively, this represents the following cascade of two measurements: first measure the qubit q_2 over the Hadamard basis, if it is in $|+\rangle$ then return either x or y with the same probability, otherwise measure the qubit q_1 in the computational basis and return x or y accordingly.

In the probabilistic case, it is usual to consider just the binary composition $\Delta \downarrow_p \oplus \Theta$. This is a safe simplification as any finite probability distribution can be obtained by repeatedly applying $\downarrow_p \oplus$ over point distributions. Unfortunately, this is not the case for effect distributions in general, as we show in the following.

► **Definition 8.** *Let $\mathfrak{D} \boxplus_E \mathfrak{T}$ be the weighted sum $E \boxtimes \mathfrak{D} + (\mathbb{I} - E) \boxtimes \mathfrak{T}$.*

Some effect distributions with support bigger than two can be defined by a nesting of \boxplus_E expressions over point distributions.

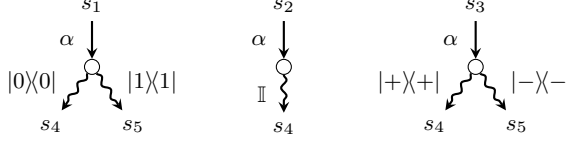
► **Example 9.** The distribution $\{x_1 \triangleright |0+\rangle\langle 0+|, x_2 \triangleright |0-\rangle\langle 0-|, x_3 \triangleright |1+\rangle\langle 1+|, x_4 \triangleright |1-\rangle\langle 1-|\}$ over $S = \{x_1, x_2, x_3, x_4\}$ can be obtained as $(\bar{x}_1 \downarrow_{|+\rangle\langle +|} \boxplus \bar{x}_2) \downarrow_{|0\rangle\langle 0|} \boxplus (\bar{x}_3 \downarrow_{|+\rangle\langle +|} \boxplus \bar{x}_4)$.

We define now the set of distributions that can be obtained starting from point distributions and applying (an arbitrary number of times) the binary operator \boxplus .

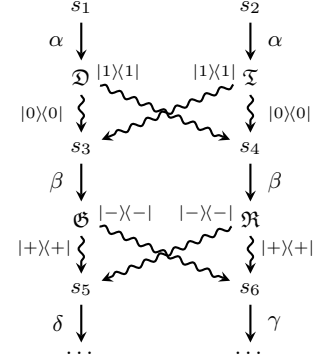
► **Definition 10.** *Given a set X , let $\mathcal{Q}^{\boxplus} X$ be the least family of sets $\mathcal{Q}_Q^{\boxplus} X \subseteq \mathcal{Q}_Q X$ such that $\bar{x} \in \mathcal{Q}_0^{\boxplus} X$ for any $x \in X$, and if $\mathfrak{D}, \mathfrak{T} \in \mathcal{Q}_Q^{\boxplus} X$ then $\mathfrak{D} \boxplus_E \mathfrak{T} \in \mathcal{Q}_{Q \uplus Q'}^{\boxplus} X$ for any $E \in Ef_{Q'}$.*

Despite having finite support, some effect distributions cannot be defined using \boxplus , roughly because of entangled pairs. Hence, we will use the general n-ary composition.

► **Theorem 11.** *If $|X| \geq 4$ and $|Q| \geq 2$, with $|\cdot|$ the cardinality, then $\mathcal{Q}_Q^{\boxplus} X \neq \mathcal{Q}_Q X$.*



(a) The eLTS of Example 16.

(b) An eLTS where $s_1 \not\sim_{lpp} s_2$.

■ **Figure 1** Examples of eLTSs.

As it is common for the probabilistic case, it is sometimes useful to see a relation between elements of a given set X as a relation over effect distributions over X . In particular, we lift a relation on states to one on effect distributions of states by taking the smallest relation that pairs the point distributions of related states and that is closed for weighted composition.

► **Definition 12.** Given $\mathcal{R} \subseteq X \times X$, we let the effect liftings of $\mathcal{R} \subseteq X \times X$ be the least family of relations $\widehat{\mathcal{R}}_Q \subseteq \mathcal{Q}_Q X \times \mathcal{Q}_Q X$ such that $\bar{s} \widehat{\mathcal{R}}_0 \bar{t}$ if $s \mathcal{R} t$, and for each $E_i \in \text{Ef}_Q$, $\mathcal{D}_i \widehat{\mathcal{R}}_Q \mathcal{T}_i$ implies $(\sum_{i \in I} E_i \boxtimes \mathcal{D}_i) \widehat{\mathcal{R}}_{Q \uplus Q'} (\sum_{i \in I} E_i \boxtimes \mathcal{T}_i)$.

Note that $\widehat{\mathcal{R}}_0$ is the usual probabilistic lifting of [19], and we denote it as $\overset{\circ}{\mathcal{R}}$. In the following we often omit Q when clear from the context. We recover the following property, known as decomposability, roughly stating that two distributions are paired by the lifting of a relation when they can be decomposed in such a way that they associate related states with the same effects.

► **Lemma 13.** For all \mathcal{R} , $\mathcal{D} \widehat{\mathcal{R}}_Q \mathcal{T}$ iff there exist a set of indices I and a set of effects $\{E_i \in \text{Ef}_Q\}_{i \in I}$ such that $\mathcal{D} = \{x_i \triangleright E_i\}_{i \in I}$, $\mathcal{T} = \{y_i \triangleright E_i\}_{i \in I}$, and $x_i \mathcal{R} y_i$ for any $i \in I$.

3.2 Effect Transition Systems and their Bisimilarity

To model quantum systems and protocols we introduce effect labelled transition systems (eLTSs). Then we investigate different notions of bisimilarity.

► **Definition 14.** An eLTS over Ef_Q is a triple $(S, \text{Act}, \rightarrow)$ where S is a set of states, Act is a set of labels, and $\rightarrow \subseteq S \times \text{Act} \times \mathcal{Q}_Q S$ is a transition relation.

Hereafter, we assume a set of qubits Q and an eLTS $(S, \text{Act}, \rightarrow)$ over Ef_Q , and we write $s \xrightarrow{\mu} \mathcal{D}$ for $(s, \mu, \mathcal{D}) \in \rightarrow$.

We instantiate two distinct definitions of semantic equivalence on quantum systems: *Aczel-Mendler* and *Larsen-Skou* bisimilarities [33]. They are known to coincide on classical probabilistic processes [19]. Notably, they do not in the quantum case.

► **Definition 15.** A symmetric relation $\mathcal{R} \subseteq S \times S$ is an AM-bisimulation if for any $s \mathcal{R} t$

if $s \xrightarrow{\mu} \mathcal{D}$ then $t \xrightarrow{\mu} \mathcal{T}$ for some \mathcal{T} such that $\mathcal{D} \widehat{\mathcal{R}}_Q \mathcal{T}$

Let AM-bisimilarity \sim_{am} be the largest AM-bisimulation.

► **Example 16.** Consider the eLTS in Figure 1a with states $\{s_1, s_2, s_3, s_4, s_5\}$ and transitions $s_1 \xrightarrow{\alpha} \mathfrak{D} = \{s_4 \triangleright |0\rangle\langle 0|, s_5 \triangleright |1\rangle\langle 1|\}$, $s_2 \xrightarrow{\alpha} \mathfrak{G} = \{s_4 \triangleright \mathbb{I}\}$, $s_3 \xrightarrow{\alpha} \mathfrak{T} = \{s_4 \triangleright |+\rangle\langle +|, s_5 \triangleright |-\rangle\langle -|\}$. Note that s_4 and s_5 are deadlock states, hence $s_4 \sim_{am} s_5$. Moreover, $s_1 \sim_{am} s_2 \sim_{am} s_3$, because $|0\rangle\langle 0| + |1\rangle\langle 1| = I = |+\rangle\langle +| + |-\rangle\langle -|$, and hence

$$\mathfrak{D} \sim_{am} \{s_4 \triangleright |0\rangle\langle 0|, s_4 \triangleright |1\rangle\langle 1|\} = \mathfrak{G} = \{s_4 \triangleright |+\rangle\langle +|, s_4 \triangleright |-\rangle\langle -|\} \sim_{am} \mathfrak{T}.$$

Still, $s_1 \not\sim_{am} s_3$, as we cannot write \mathfrak{D} and \mathfrak{T} with the same effects as required by Lemma 13.

This example, inspired by [32], proves that \sim_{am} is not transitive. We thus generalize *Larsen-Skou bisimilarity* [26] (named kernel bisimilarity in [33]) to the quantum case.

► **Definition 17.** An equivalence relation $\mathcal{R} \subseteq S \times S$ is an LS-bisimulation if for any $s \mathcal{R} t$

$$\text{if } s \xrightarrow{\mu} \mathfrak{D} \text{ then } t \xrightarrow{\mu} \mathfrak{T} \text{ for some } \mathfrak{T} \text{ such that } \forall C \in S/\mathcal{R} \quad \sum_{x \in C} \mathfrak{D}(x) = \sum_{x \in C} \mathfrak{T}(x)$$

with S/\mathcal{R} the equivalence classes of S . Let LS-bisimilarity \sim_{ls} be the largest LS-bisimulation.

We show that \sim_{ls} behaves differently from \sim_{am} , and indeed it is strictly coarser.

► **Example 18.** Consider Example 16. We can see that $s_1 \sim_{ls} s_3$ as both \mathfrak{D} and \mathfrak{T} associate the equivalence class $\{s_4, s_5\}$ with the effect \mathbb{I} .

► **Theorem 19.** For any eLTS over Ef_Q with states S , $\sim_{am} \subseteq \sim_{ls}$. Moreover, $\sim_{am} = \sim_{ls}$ if $Q = \emptyset$, and $\sim_{am} \subsetneq \sim_{ls}$ if Q is of dimension at least 2 and S of cardinality at least 4.

LS-bisimilarity is also trivially an equivalence relation. In the following we discuss its adequacy as quantum semantic equivalence.

Our ground truth is that bisimilar processes must exhibit the same probabilistic behaviour, as it is the only observable property of quantum systems. We therefore define a parameterized version of probabilistic bisimilarity for eLTSs, stating that equivalent states should express the same probabilistic behaviour when instantiated with any possible quantum state. More precisely, for each ρ , we define a ρ -bisimilarity equating states that are probabilistically bisimilar when each effect distribution is instantiated with ρ to obtain a probability distribution.

► **Definition 20.** Given $\rho \in DM_Q$, a symmetric relation $\mathcal{R} \subseteq S \times S$ is a ρ -bisimulation if for any $s \mathcal{R} t$

$$\text{if } s \xrightarrow{\mu} \mathfrak{D} \text{ then } t \xrightarrow{\mu} \mathfrak{T} \text{ for some } \mathfrak{T} \text{ such that } \mathfrak{D} \downarrow_{\rho} \overset{\circ}{\mathcal{R}} \mathfrak{T} \downarrow_{\rho}$$

Let ρ -bisimilarity \sim_{ρ} be the largest ρ -bisimulation. We define probabilistic behavioural equivalence \simeq_{pbe} as the relation pairing states that are indistinguishable when every possible quantum state is considered, i.e. $\simeq_{pbe} = \bigcap_{\rho \in DM_Q} \sim_{\rho}$.

In other words, an adversary trying to disprove $s \simeq_{pbe} t$ can test their probabilistic behaviour on any arbitrary input state ρ , looking for one such that $s \not\sim_{\rho} t$. One could hypothesize an even stronger adversary, with the faculty of choosing a different input state at each step of the computation, not just once at the beginning as for \simeq_{pbe} . We formalize this notion as *locally-parameterized probabilistic bisimilarity*, and we investigate how \sim_{ls} relates with both these behavioural equivalences.

► **Definition 21.** A symmetric relation $\mathcal{R} \subseteq S \times S$ is a lpp-bisimulation if for any $s \mathcal{R} t$ if $s \xrightarrow{\mu} \mathcal{D}$ then $t \xrightarrow{\mu} \mathcal{T}$ for some \mathcal{T} such that $\mathcal{D} \downarrow_{\rho} \overset{\circ}{\mathcal{R}} \mathcal{T} \downarrow_{\rho}$ for any $\rho \in DM_Q$

Let lpp-bisimilarity \sim_{lpp} be the largest lpp-bisimulation.

We exemplify the difference between \simeq_{pbe} and \sim_{lpp} below.

► **Example 22.** Consider the eLTS in Figure 1b, where s_5 and s_6 are immediately distinguishable as they perform different visible actions. To show that $s_1 \not\sim_{lpp} s_2$ it suffices to choose $|0\rangle\langle 0|$ for their first reduction and $|+\rangle\langle +|$ for the second one. Formally, since $\mathcal{D} \downarrow_{|0\rangle\langle 0|} = \overline{s_3}$ and $\mathcal{T} \downarrow_{|0\rangle\langle 0|} = \overline{s_4}$, we must have that $s_3 \sim_{lpp} s_4$. But $\mathcal{G} \downarrow_{|+\rangle\langle +|} = \overline{s_5}$ and $\mathcal{R} \downarrow_{|+\rangle\langle +|} = \overline{s_6}$. Thus, $s_3 \sim_{lpp} s_4$ requires $s_5 \sim_{lpp} s_6$, which does not hold.

Finally, note that neither $\sim_{|0\rangle\langle 0|}$ nor $\sim_{|+\rangle\langle +|}$ are capable of distinguishing s_1 and s_2 , as indeed $\mathcal{G} \downarrow_{|0\rangle\langle 0|} = \mathcal{R} \downarrow_{|0\rangle\langle 0|}$ and $\mathcal{D} \downarrow_{|+\rangle\langle +|} = \mathcal{T} \downarrow_{|+\rangle\langle +|}$.

Using Theorem 4, we prove that \sim_{l_s} is adequate for characterizing \sim_{lpp} .

► **Theorem 23.** For any $s, t \in S$, $s \sim_{l_s} t$ if and only if $s \sim_{lpp} t$.

Quite surprisingly, for finite eLTSs the two relations \sim_{lpp} and \simeq_{pbe} coincide.

► **Theorem 24.** For any $s, t \in S$, $s \sim_{l_s} t$ implies $s \simeq_{pbe} t$. Moreover, if S is finitely dimensional, then $s \simeq_{pbe} t$ implies $s \sim_{l_s} t$.

The interesting case is for $\simeq_{pbe} \subseteq \sim_{l_s}$, where we consider the (finite) set of effects \mathbb{E} that may appear in the eLTS, and we build a density operator $\rho_{\mathbb{E}}$ that distinguishes all the effects in \mathbb{E} . Roughly, \sim_{ρ} requires associating the same probability to all the equivalence classes of states, but this can only be the case when the associated effects are the same for $\rho = \rho_{\mathbb{E}}$. Indeed, a single quantum state is sufficient for distinguishing s_1 and s_2 of Example 22.

► **Example 25.** Consider Example 22, and let $\rho = \frac{1}{2} |0\rangle\langle 0| + \frac{1}{2} |+\rangle\langle +|$. Then $s_1 \not\sim_{\rho} s_2$ (and hence $s_1 \not\sim_{pbe} s_2$). Note that $\mathcal{D} \downarrow_{\rho} = \overline{s_3} \cdot \frac{3}{4} \oplus \overline{s_4}$ and $\mathcal{T} \downarrow_{\rho} = \overline{s_3} \cdot \frac{1}{4} \oplus \overline{s_4}$. For s_1 to be ρ -bisimilar to s_2 , it must be that $s_3 \sim_{\rho} s_4$, which is false since $\mathcal{G} \downarrow_{\rho} = \overline{s_5} \cdot \frac{3}{4} \oplus \overline{s_6}$ and $\mathcal{R} \downarrow_{\rho} = \overline{s_5} \cdot \frac{1}{4} \oplus \overline{s_6}$.

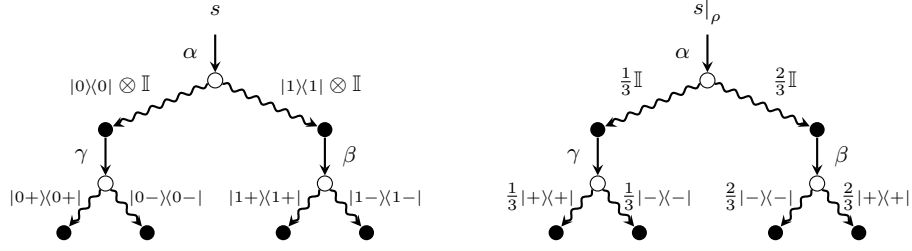
We thus have shown that two bisimilar processes behave the same under any possible quantum input. Nonetheless, LS-bisimilarity is still decidable in an efficient way, thanks to the finite representation of effects. The partition refinement algorithm proposed in [22], for example, could promptly be adapted to our eLTSs. More in detail, that algorithm is parametric with respect to the *functor* used to specify the visible labels and the weights of a generic transition system, which in the case of eLTSs are *Act* and the effects in $\mathbb{C}^{d \times d}$.

We conclude the section by introducing a *partial evaluation* operator relating eLTSs over different sets of qubits, namely instantiating some of the expected input qubits of the former to some specific state.

► **Definition 26.** Given an eLTS $\mathbb{S} = (S, Act, \rightarrow_1)$ over Ef_Q and $\rho \in DM_{Q'}$, with $Q' \subseteq Q$, the partial evaluation of \mathbb{S} with ρ is the eLTS over $Ef_{Q \setminus Q'}$ defined as (S', Act, \rightarrow) , where $S' = \{s|_{\rho} \mid s \in S\}$ and \rightarrow is the smallest relation satisfying the following rule.

$$\frac{s \xrightarrow{\mu} \{s_i \triangleright E_i\}_{i \in I}}{s|_{\rho} \xrightarrow{\mu} \{s_i|_{\rho} \triangleright \text{tr}_{Q'}(E_i(\rho \boxtimes \mathbb{I}_{Q \setminus Q'}))\}_{i \in I}} \text{PEVAL}$$

► **Example 27.** Figure 2 shows an eLTS over two qubits and its partial evaluation (of the first qubit) with $\rho = \frac{1}{3} |0\rangle\langle 0| + \frac{2}{3} |1\rangle\langle 1|$.



■ **Figure 2** Partial evaluation of the first qubit of the eLTS on the left with $\rho = \frac{1}{3} |0\rangle\langle 0| + \frac{2}{3} |1\rangle\langle 1|$.

LS-bisimilarity is preserved by partial evaluation.

► **Theorem 28.** *If $s \sim_{ls} t$ then $s|_\rho \sim_{ls} t|_\rho$.*

For ρ large enough, the partial evaluation returns a pLTS obtained by applying the same quantum input to each effect distribution of the eLTS. Hence, $s|_\rho \sim_{ls} t|_\rho$ corresponds to verifying $s \sim_\rho t$, and the following is a corollary of Theorem 24.

► **Corollary 29.** *Given a finite eLTS (S, Act, \rightarrow) over Ef_Q and $s, t \in S$, if for any $\rho \in DM_Q$ we have $s|_\rho \sim_{ls} t|_\rho$, then $s \sim_{ls} t$.*

Having found that \sim_{ls} satisfies all our desiderata for a quantum behavioural equivalence, we will denote it simply as \sim for the rest of the paper.

4 Modelling a Minimal Process Algebra with eLTSs

We explore how eLTSs can model concurrent communicating quantum systems by considering a *minimal Quantum Process Algebra (mQPA)* featuring non-deterministic and parallel composition of processes, synchronization, restriction, measurements and application of unitaries. For synchronization, we assume that the set of actions Act contains a distinguished element τ , and that every other label $\alpha \in Act$ has in inverse $\bar{\alpha}$ that is involutive, i.e. such that $\bar{\bar{\alpha}} = \alpha$. We equip our algebra with two distinct semantics: a standard *Schrödinger* stateful pLTS semantics that depends on the quantum input, and a *Heisenberg* eLTS semantics that does not. Both are based on configurations, pairing the processes with superoperators in the latter, and density operators in the former, as it is common in the literature [8, 9, 7]. We prove that the two coincide: we can use bisimilarity in the Heisenberg eLTS to prove probabilistic bisimilarity in all the Schrödinger pLTSs.

► **Definition 30.** *An mQPA process P is defined below, with $\mu \in Act$ an action and $\sum_i E_i = \mathbb{I}$.*

$$P ::= \mathbf{0} \mid P + P \mid P \parallel P \mid P \setminus \alpha \mid \mu.([E_i]P_i)_{i \in I} \mid U; P$$

As usual, $\mathbf{0}$ stands for a deadlock process, and the meaning of parallel, non-deterministic sum and restriction is as expected. A prefix $\mu.([E_i]P_i)_{i \in I}$ represents an action μ followed by a destructive measurement over the qubits of E_i , whose outcome controls the evolution of the process. Finally, $U; P$ behaves as P would over a state that has been modified by U . Recall that unitaries and effects symbols come with the set of qubits they act on. Moreover, we assume such sets disjoint when needed (enforced e.g. by a type system [7]). More in detail, the sets of qubits used in (unitaries and measurements of) the parallel processes P and R of $P \parallel R$ must be disjoint, and the qubits measured by E_i in $\mu.([E_i]P_i)$ cannot be used by P_i . On the same line, we let Q_P be the smallest set containing the qubits used in the effects and unitaries of P . Finally, we often write $\mu.P$ in place of $\mu.([1]P)$, and μ for the process $\mu.\mathbf{0}$.

$$\begin{array}{c}
\frac{\rho_i = \mathcal{M}_{E_i}(\rho)}{\langle \rho, \mu.([E_i]s_i)_{i \in I} \rangle \xrightarrow{\mu} \{\langle \rho_i, s_i \rangle \triangleright \text{tr}(\rho_i)\}_{i \in I}} \text{SPRE} \quad \frac{\langle \mathcal{U}(\rho), s \rangle \xrightarrow{\mu} \mathfrak{D}}{\langle \rho, U; s \rangle \xrightarrow{\mu} \mathfrak{D}} \text{SU} \\
\frac{\langle \rho, s \rangle \xrightarrow{\mu} \{\langle \rho_i, s_i \rangle \triangleright p_i\}_{i \in I} \quad \mu \neq \alpha \quad \mu \neq \bar{\alpha}}{\langle \rho, s \setminus \alpha \rangle \xrightarrow{\mu} \{\langle \rho_i, s_i \setminus \alpha \rangle \triangleright p_i\}_{i \in I}} \text{SRES} \\
\frac{\langle \rho, s \rangle \xrightarrow{\mu} \mathfrak{D}}{\langle \rho, s + t \rangle \xrightarrow{\mu} \mathfrak{D}} \text{SSUML} \quad \frac{\langle \rho, s \rangle \xrightarrow{\mu} \{\langle \rho_i, s_i \rangle \triangleright p_i\}_{i \in I}}{\langle \rho, s \parallel t \rangle \xrightarrow{\mu} \{\langle \rho_i, s_i \parallel t \rangle \triangleright p_i\}_{i \in I}} \text{SPARL} \\
\frac{\langle \rho, t \rangle \xrightarrow{\mu} \mathfrak{D}}{\langle \rho, s + t \rangle \xrightarrow{\mu} \mathfrak{D}} \text{SSUMR} \quad \frac{\langle \rho, t \rangle \xrightarrow{\mu} \{\langle \rho_j, t_j \rangle \triangleright p_j\}_{j \in J}}{\langle \rho, s \parallel t \rangle \xrightarrow{\mu} \{\langle \rho_j, s \parallel t_j \rangle \triangleright p_j\}_{j \in J}} \text{SPARR} \\
\frac{\langle \rho, s \rangle \xrightarrow{\mu} \{\langle \rho_i, s_i \rangle \triangleright p_i\}_{i \in I} \quad \langle \rho_i, t \rangle \xrightarrow{\bar{\mu}} \{\langle \rho_j, t_j \rangle \triangleright p_j\}_{j \in J_i}}{\langle \rho, s \parallel t \rangle \xrightarrow{\tau} \{\langle \rho_j, s_i \parallel t_j \rangle \triangleright p_j\}_{(i,j) \in I \times J_i}} \text{SSYNL} \\
\frac{\langle \rho, t \rangle \xrightarrow{\mu} \{\langle \rho_i, t_i \rangle \triangleright p_i\}_{i \in I} \quad \langle \rho_i, s \rangle \xrightarrow{\bar{\mu}} \{\langle \rho_{ij}, s_j \rangle \triangleright p_{ij}\}_{j \in J}}{\langle \rho, s \parallel t \rangle \xrightarrow{\tau} \{\langle \rho_{ij}, s_i \parallel t_j \rangle \triangleright p_{ij}\}_{(i,j) \in I \times J}} \text{SSYNCR}
\end{array}$$

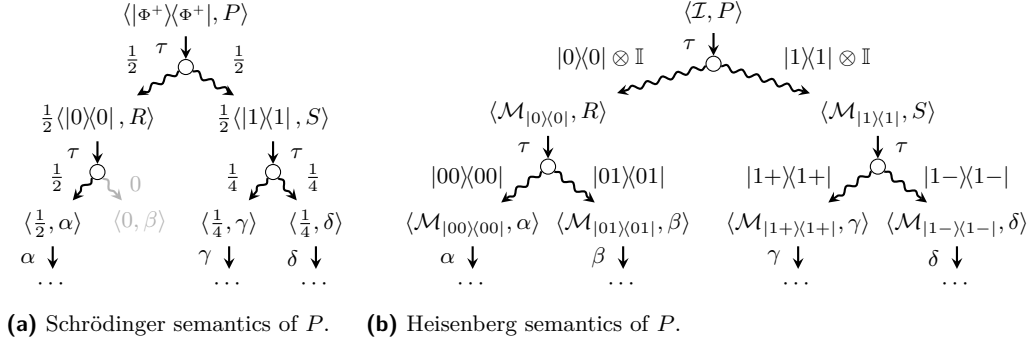
(a) Rules for Schrödinger stateful semantics.

$$\begin{array}{c}
\frac{}{\langle \mathcal{E}, \mu.([E_i]s_i)_{i \in I} \rangle \xrightarrow{\mu} \{\langle \mathcal{M}_{E_i} \circ \mathcal{E}, s_i \rangle \triangleright \mathcal{Z}(E_i \boxtimes \mathbb{I})\}_{i \in I}} \text{HPRE} \quad \frac{\langle \mathcal{U} \circ \mathcal{E}, s \rangle \xrightarrow{\mu} \mathfrak{D}}{\langle \mathcal{E}, U; s \rangle \xrightarrow{\mu} \mathfrak{D}} \text{HU} \\
\frac{\langle \mathcal{E}, s \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i, s_i \rangle \triangleright E_i\}_{i \in I} \quad \mu \neq \alpha \quad \mu \neq \bar{\alpha}}{\langle \mathcal{E}, s \setminus \alpha \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i, s_i \setminus \alpha \rangle \triangleright E_i\}_{i \in I}} \text{HRES} \\
\frac{\langle \mathcal{E}, s \rangle \xrightarrow{\mu} \mathfrak{D}}{\langle \mathcal{E}, s + t \rangle \xrightarrow{\mu} \mathfrak{D}} \text{HSUML} \quad \frac{\langle \mathcal{E}, s \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i, s_i \rangle \triangleright E_i\}_{i \in I}}{\langle \mathcal{E}, s \parallel t \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i, s_i \parallel t \rangle \triangleright E_i\}_{i \in I}} \text{HPARL} \\
\frac{\langle \mathcal{E}, t \rangle \xrightarrow{\mu} \mathfrak{D}}{\langle \mathcal{E}, s + t \rangle \xrightarrow{\mu} \mathfrak{D}} \text{HSUMR} \quad \frac{\langle \mathcal{E}, t \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_j, s_j \rangle \triangleright E_j\}_{j \in J}}{\langle \mathcal{E}, s \parallel t \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_j, s \parallel t_j \rangle \triangleright E_j\}_{j \in J}} \text{HPARR} \\
\frac{\langle \mathcal{E}, s \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i, s_i \rangle \triangleright E_i\}_{i \in I} \quad \langle \mathcal{E}_i, t \rangle \xrightarrow{\bar{\mu}} \{\langle \mathcal{E}_j, t_j \rangle \triangleright E_j\}_{j \in J_i}}{\langle \mathcal{E}, s \parallel t \rangle \xrightarrow{\tau} \{\langle \mathcal{E}_j, s_i \parallel t_j \rangle \triangleright E_j\}_{(i,j) \in I \times J_i}} \text{HSYNL} \\
\frac{\langle \mathcal{E}, t \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_j, t_j \rangle \triangleright E_j\}_{j \in J} \quad \langle \mathcal{E}_j, s \rangle \xrightarrow{\bar{\mu}} \{\langle \mathcal{E}_i, s_i \rangle \triangleright E_i\}_{i \in I_j}}{\langle \mathcal{E}, s \parallel t \rangle \xrightarrow{\tau} \{\langle \mathcal{E}_i, s_i \parallel t_j \rangle \triangleright E_i\}_{(j,i) \in J \times I_j}} \text{HSYNCR}
\end{array}$$

(b) Rules for Heisenberg stateless semantics.

■ **Figure 3** Rules for stateful and stateless semantics of mQPA.

16:12 Effect Semantics for Quantum Process Calculi



■ **Figure 4** Our two semantics for the process P of Example 32.

We consider an operational, stateful semantics in the style of [7, 9, 12] for mQPA given in terms of a pLTS, where each state is the pairing of a density operator and a process. Being state-based, we name this Schrödinger semantics.

► **Definition 31.** *The Schrödinger semantics of mQPA is given by a pLTS whose states are pairs $\langle \rho, P \rangle$ for P mQPA process and $\rho \in DM_{Q'}$ density operator with $Q' \supseteq Q_P$, and where the transition is the smallest relation satisfying the rules in Figure 3a.*

The SPRE rule updates the quantum state through the destructive measurement operator $\mathcal{M}_{E_i} : DM_Q \rightarrow DM_{Q'}$ associated to the effect $E_i \in Ef_{Q \setminus Q'}$ defined by $\mathcal{M}_{E_i}(\rho) = \text{tr}_{Q \setminus Q'}(\sqrt{E_i} \otimes \mathbb{I}_{Q'})\rho(\sqrt{E_i} \otimes \mathbb{I}_{Q'})$. Given the unitary U acting on qubits Q_U , the SU rule updates the state with the superoperator $\mathcal{U} : DM_Q \rightarrow DM_Q$, defined as $\mathcal{U}(\rho) = (U \otimes \mathbb{I}_{Q \setminus Q_U})\rho(U^\dagger \otimes \mathbb{I}_{Q \setminus Q_U})$.

Note that the resulting effect distribution is always a probability distribution, obtained by tracing the resulting density operator. We remark that SSYNCL and SSYNCR only differ in the order of the application of measurements between the two branches of the parallel operator, as both the orderings are possible.

► **Example 32.** Consider a process $P = \tau.([|0\rangle\langle 0|]R, [|1\rangle\langle 1|]S)$ with $R = \tau.([|0\rangle\langle 0|]\alpha, [|1\rangle\langle 1|]\beta)$ and $S = H; \tau.([|0\rangle\langle 0|]\gamma, [|1\rangle\langle 1|]\delta)$. First, P measures a qubit q_1 in the computational basis and then measures a qubit q_2 either in the computational or in the Hadamard basis. The stateful semantics of $\langle |\Phi^+\rangle\langle\Phi^+|, P \rangle$ is given in Figure 4a. Notice that measurements are destructive and are always prefixed by an action (which is not necessarily a τ as in [9, 7]).

For any process P , the stateful semantics results in infinitely many pLTSs according to the input quantum state ρ . We seek an alternative stateless characterization, hence adequate for algorithmic verification. We therefore give a new semantics for mQPA processes in terms of eLTSs. We name this Heisenberg semantics, because its focus is on the effects used as weights, rather than on the quantum state. Moreover, it is a symbolic semantics, as it is independent of the input state.

► **Definition 33.** *The Heisenberg semantics of mQPA with respect to a chosen set Q of qubits is given by an eLTS over Ef_Q whose states are pairs $\langle \mathcal{E}, P \rangle$ for P mQPA process and $\mathcal{E} \in DM_Q \rightarrow DM_{Q'}$ superoperator with $Q \supseteq Q' \supseteq Q_P$, and where the transition is the smallest relation satisfying the rules in Figure 3b.*

The Heisenberg semantics of a process P is the eLTS over Ef_{Q_P} rooted in $\langle \mathcal{I}, P \rangle$. The superoperator \mathcal{E} in the Heisenberg configuration $\langle \mathcal{E}, P \rangle$ records the performed measurements and unitaries. According to that, the weights of the subsequent effect distributions must be

updated through the corresponding dual superoperator \mathcal{Z} . In the HPRE rule, the superoperator $\mathcal{E} : DM_Q \rightarrow DM_{Q'}$ is updated by composition with the measurement superoperator associated to the effect $E_i \in Ef_{Q_m}$, $\mathcal{M}_{E_i} : DM_{Q'} \rightarrow DM_{Q' \setminus Q_m}$, where $Q_m \subseteq Q'$ are the measured qubits. The weight resulting from the measurement is $E_i \boxtimes \mathbb{I}$ with $\mathbb{I} \in Ef_{Q' \setminus Q_m}$ meaning that the qubits that are not measured are left unchanged. Finally, the effect is updated via the dual superoperator \mathcal{Z} representing the previously applied transformations. All the other rules mirror the Schrödinger semantics.

► **Example 34.** Consider the process P of Example 32. Figure 4b shows its Heisenberg semantics. Inside the lowest configurations, we have $\mathcal{M}_{|00\rangle\langle 00|}$, obtained by composing $\mathcal{M}_{|0\rangle\langle 0|}$ over the first qubit with $\mathcal{M}_{|0\rangle\langle 0|}$ over the second qubit, and similarly $\mathcal{M}_{|1+\rangle\langle 1+|} = \mathcal{M}_{|0\rangle\langle 0|} \circ \mathcal{H} \circ \mathcal{M}_{|1\rangle\langle 1|}$.

In order to fix an input state $\rho \in DM_{Q_P}$, and thus instantiate the semantics of the process P , we can use the partial evaluation $\cdot|_\rho$ in Definition 26. Since ρ defines a value for all the qubits used by P , this is a full evaluation: the resulting eLTS is indeed a pLTS.

► **Example 35.** Consider the process P of Example 32 and $\rho = |\Phi^+\rangle\langle\Phi^+|$. Notice that $\langle \mathcal{I}, P \rangle|_\rho$, with $\langle \mathcal{I}, P \rangle$ as in Figure 4b, is a pLTS isomorphic to the Schrödinger semantics $\langle \rho, P \rangle$ in Figure 4a. The labels coincide since they are syntax-driven. Furthermore, the weights are identical, since the probabilities of the Schrödinger semantics are exactly the result of applying ρ to the effects in the Heisenberg eLTS.

This example hints at a connection between the two semantics, which is to be expected given the duality between effects and states in quantum theory. Indeed, the eLTSs produced by instantiating the Heisenberg semantics have exactly the same transitions of the Schrödinger semantics, thus the following holds.

► **Theorem 36.** *For any process P and $\rho \in DM_{Q_P}$, $\langle \mathcal{I}, P \rangle|_\rho \sim \langle \rho, P \rangle$.*

It follows that we can verify whether two processes are bisimilar for any input just by looking at their Heisenberg semantics.

► **Corollary 37.** *Given two processes P and R , $\langle \mathcal{I}, P \rangle \sim \langle \mathcal{I}, R \rangle$ if and only if $\langle \rho, P \rangle \sim \langle \rho, R \rangle$ for any $\rho \in DM_{Q_P}$.*

We conclude with a real-world example: the well-known teleportation protocol [3].

► **Example 38.** Alice (**A**) and Bob (**B**) each have a qubit of q_2, q_3 , and **A** wants to send its additional qubit q_1 to **B** without a quantum channel. The agents just apply unitaries and measurements on their qubits locally, and synchronize over labels that represent the result of measurements. The protocol is encoded as the following mQPA process **Tel**.

$$\begin{aligned} \mathbf{Tel} &:= (\mathbf{A} \parallel \mathbf{B}) \setminus \alpha, \beta, \gamma, \delta \\ \mathbf{A} &:= CNOT_{q_1 q_2}; H_{q_1} \boxtimes \mathbb{I}_{q_2}; \tau.([00]\langle 00|]\alpha, [01]\langle 01|]\beta, [10]\langle 10|]\gamma, [11]\langle 11|]\delta) \\ \mathbf{B} &:= \bar{\alpha}. \mathbf{B}' + \bar{\beta}. X; \mathbf{B}' + \bar{\gamma}. Z; \mathbf{B}' + \bar{\delta}. Z; X; \mathbf{B}' \end{aligned}$$

where \mathbf{B}' is the unspecified continuation of \mathbf{B} with q_3 .

Then, if q_2 and q_3 are in state $|\Phi^+\rangle\langle\Phi^+|$, as prescribed by the protocol, **Tel** is indistinguishable to \mathbf{B}' acting on q_1 instead of q_3 : $\langle \mathcal{I}, \mathbf{Tel} \rangle|_{|\Phi^+\rangle\langle\Phi^+|} \sim \langle \mathcal{I}, \tau. \tau. \mathbf{B}'[q_3/q_1] \rangle$.

5 Related Works

In our work we follow a foundational approach to quantum bisimilarity. We employ effect distributions (a finite non-normalized version of positive operator-valued measures, POVMs [30]) as a generalization of sub-probability distributions, finding them well-suited to model the observable behaviour of quantum systems. Our notion generalizes the quantum monad of [1], which is based on projectors, and it instantiates the abstract “effect algebra monad” of [21]. More in depth, the author in [21] proposes effects monoids, i.e. effect algebras with multiplication, and use them as weights of distributions. Our effects do have tensoring as a multiplication operator, but it does not form a proper effect monoid since it changes the effects dimensions. These works come from the fields of quantum complexity and quantum logic. We apply similar concepts to quantum protocol semantics, introducing eLTSs and studying their composition and their behavioural equivalences.

Our eLTSs can be seen as a labelled, non-deterministic version of the effect-valued Quantum Markov chains of [16], where tensor product is used instead of sequential effect composition. The most general model of “quantum transition system” is the one of [32, 27], where the weights are superoperators instead of effects, to capture also non-destructive measurements and qubit initialization. The author of [32] introduces two different notions of bisimilarity, which we recover in our minimal, effect-based setting as AM and LS bisimilarity. However, none of these works feature non-determinism, nor do they apply the proposed coalgebraic model to process calculi suitable for expressing quantum protocols.

In the literature the semantics of quantum processes is usually described via pLTSs and probabilistic bisimilarity [24, 10, 8, 9, 7]. Despite their differences, these works all define a pLTS made of configurations, i.e. pairs of quantum values and syntactic processes, and they require bisimilar systems to exhibit the same probabilistic behaviour and observable quantum values. Many of the existing works have to tweak the natural definition of probabilistic bisimilarity in an *ad hoc* manner, in order to capture the peculiar observable properties of quantum values. We instead introduce purely quantum LTSs, and we manipulate quantum values only through effects, which represent their observable probabilistic behaviour. Moreover, in the previous proposals, verifying the equivalence of two processes requires instantiating them with each possible quantum input, impeding algorithmic verification. Using effects, we describe the “symbolic” semantics of protocols, abstracting away from the input.

Most similar to our work is [11], which introduces superoperator-valued quantum distributions, analogous to the ones in [17, 32, 27]. This allows modelling the more expressive non-destructive measurements and quantum communication, but their bisimilarity does not respect the observational properties prescribed by quantum theory [23, 7, 13]. For the operational semantics of their language, they employ configurations of superoperators and processes (as in our Heisenberg semantics), and they build a superoperator-weighted LTS. The bisimilarity that they propose is equivalent to the one in [10], and requires bisimilar configurations with the same weights, leading to a form of AM-bisimilarity finer than of our LS-bisimilarity. For example, it discriminates the following processes (in mQPA syntax).

► **Example 39.** Let $P = \tau.([|0\rangle\langle 0|]R, [|1\rangle\langle 1|]R')$ and $Q = \tau.([|+\rangle\langle +|]R, [|-\rangle\langle -|]R')$ with R and R' two deadlock processes that maintain the ownership of the measured qubit (recall that [11] considers non-destructive measurements), thus making it unobservable. In other words, P and Q perform some local measurement on their qubit, without leaking any classical information to an external observer. Nonetheless, P and Q are not bisimilar for the symbolic/open bisimilarity of [11, 10], as can be seen studying the ground behaviour of $\langle \Phi^+, P \rangle$ and $\langle \Phi^+, Q \rangle$. These processes are bisimilar in our proposal, as well as in other recent works [23, 9, 7].

The bisimilarity of [10] has been relaxed in [13, 9] to match the prescriptions of quantum theory, but no symbolic version of this coarser bisimilarity has been proposed.

6 Conclusions and future works

We proposed effect labelled transition systems (eLTSs), a new operational model that generalizes the probabilistic ones and is suitable for modeling quantum concurrent systems. We investigated bisimilarity, adapting two equivalent definitions of probabilistic bisimilarity to the quantum case, namely Aczel-Mendler and Larsen-Skou bisimilarity. Despite coinciding for classical systems, they disagree on quantum processes, and only the latter is guaranteed to be an equivalence relation. Then, we proved the adequacy of the Larsen-Skou bisimilarity, showing it correct and complete with respect to the observable probabilistic behaviour prescribed by quantum theory.

This model allows for a purely quantum-based semantics of quantum protocols, with the advantage of providing an algorithmically verifiable equivalence over processes. Indeed, eLTSs can be easily defined in a coalgebraic fashion, allowing e.g. to resort to the general algorithm for partition refinement of [22] for proving Larsen-Skou bisimilarity.

We assessed our approach in a process calculus with minimal features, like destructive measurements, unitaries, synchronization and non-determinism. In the standard probabilistic approach to quantum process calculi [24, 7, 8, 10, 9, 6], processes must be compared with respect to every possible input quantum state, thus considering a continuously infinite set of cases. We instead equipped our calculus with a stateless eLTS semantics, and proved that it is consistent with the natural stateful semantics: two processes are bisimilar in the eLTS if they are indistinguishable on every input quantum state.

As a future work, we plan to investigate quantum extensions of Hennessy-Milner logic for characterizing Larsen-Skou bisimilarity. Moreover, we aim to enrich our process calculus with quantum value passing, and to study its stateless semantics using superoperator-weighted models, like the one of [11].

References

- 1 Samson Abramsky, Rui Soares Barbosa, Nadish de Silva, and Octavio Zapata. The quantum monad on relational structures. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *MFCS 2017*, volume 83 of *LIPICs*, pages 35:1–35:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.MFCS.2017.35.
- 2 Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, 2014. doi:10.1016/j.tcs.2014.05.025.
- 3 Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70:1895–1899, 1993. doi:10.1103/PhysRevLett.70.1895.
- 4 Filippo Bonchi, Alexandra Silva, and Ana Sokolova. The power of convex algebras. In Roland Meyer, Uwe Nestmann, and Marc Herbstritt, editors, *CONCUR 2017*, volume 85 of *LIPICs*, pages 23:1–23:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.CONCUR.2017.23.
- 5 Marcello Caleffi, Angela Sara Cacciapuoti, and Giuseppe Bianchi. Quantum internet: from communication to distributed computing! In Jón Atli Benediktsson and Falko Dressler, editors, *NANOCOM 2018*, pages 3:1–3:4. ACM, 2018. doi:10.1145/3233188.3233224.

- 6 Lorenzo Ceragioli, Fabio Gadducci, Giuseppe Lomurno, and Gabriele Tedeschi. Quantum bisimilarity via barbs and contexts: Curbing the power of non-deterministic observers (extended version). *CoRR*, abs/2311.06116, 2023. doi:10.48550/arXiv.2311.06116.
- 7 Lorenzo Ceragioli, Fabio Gadducci, Giuseppe Lomurno, and Gabriele Tedeschi. Quantum bisimilarity via barbs and contexts: Curbing the power of non-deterministic observers. *Proceedings of the ACM on Programming Languages*, 8(POPL):43:1269–43:1297, 2024. doi:10.1145/3632885.
- 8 Timothy A. S. Davidson. *Formal Verification Techniques Using Quantum Process Calculus*. PhD thesis, University of Warwick, 2012.
- 9 Yuxin Deng. Bisimulations for probabilistic and quantum processes. In Sven Schewe and Lijun Zhang, editors, *CONCUR 2018*, volume 118 of *LIPICs*, pages 2:1–2:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.CONCUR.2018.2.
- 10 Yuxin Deng and Yuan Feng. Open bisimulation for quantum processes. In Jos C. M. Baeten, Thomas Ball, and Frank S. de Boer, editors, *TCS 2012*, volume 7604 of *LNCS*, pages 119–133. Springer, 2012. doi:10.1007/978-3-642-33475-7_9.
- 11 Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic*, 15(2):14:1–14:32, 2014. doi:10.1145/2579818.
- 12 Yuan Feng, Runyao Duan, and Mingsheng Ying. Bisimulation for quantum processes. *ACM Transactions on Programming Languages and Systems*, 34(4):17:1–17:43, 2012. doi:10.1145/2400676.2400680.
- 13 Yuan Feng and Mingsheng Ying. Toward automatic verification of quantum cryptographic protocols. In Luca Aceto and David de Frutos-Escrig, editors, *CONCUR 2015*, volume 42 of *LIPICs*, pages 441–455. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.CONCUR.2015.441.
- 14 Fei Gao, SuJuan Qin, Wei Huang, and QiaoYan Wen. Quantum private query: A new kind of practical quantum cryptographic protocol. *Science China Physics, Mechanics & Astronomy*, 62(7):70301, 2019. doi:10.1007/s11433-018-9324-6.
- 15 Simon J. Gay and Rajagopal Nagarajan. Communicating quantum processes. In Jens Palsberg and Martín Abadi, editors, *POPL 2005*, pages 145–157. ACM, 2005. doi:10.1145/1040305.1040318.
- 16 Stanley Gudder. Quantum Markov chains. *Journal of Mathematical Physics*, 49(7):072105, 2008. doi:10.1063/1.2953952.
- 17 Ichiro Hasuo and Naohiko Hoshino. Semantics of higher-order quantum computation via geometry of interaction. In *LICS 2011*, pages 237–246. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.26.
- 18 Teiko Heinosaari and Mário Ziman. *The Mathematical Language of Quantum Theory: From Uncertainty to Entanglement*. Cambridge University Press, 2011.
- 19 Matthew Hennessy. Exploring probabilistic bisimulations, part I. *Formal Aspects of Computing*, 24(4-6):749–768, 2012. doi:10.1007/s00165-012-0242-7.
- 20 Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995. doi:10.1016/0304-3975(94)00172-F.
- 21 Bart Jacobs. Probabilities, distribution monads, and convex categories. *Theoretical Computer Science*, 412(28):3323–3336, June 2011. doi:10.1016/j.tcs.2011.04.005.
- 22 Jules Jacobs and Thorsten Wißmann. Fast coalgebraic bisimilarity minimization. *Proceedings of the ACM on Programming Languages*, 7(POPL):52:1514–52:1541, 2023. doi:10.1145/3571245.
- 23 Takahiro Kubota, Yoshihiko Kakutani, Go Kato, Yasuhito Kawano, and Hideki Sakurada. Application of a process calculus to security proofs of quantum protocols. In Hamid R. Arabnia, George A. Gravvanis, and Ashu M. G. Solo, editors, *FCS 2012*, pages 141–147. CSREA Press, 2012.
- 24 Marie Lalire. Relations among quantum processes: Bisimilarity and congruence. *Mathematical Structures in Computer Science*, 16(3):407–428, 2006. doi:10.1017/S096012950600524X.

- 25 Marie Lalire and Philippe Jorrand. A process algebraic approach to concurrent and distributed quantum computation: Operational semantics. *CoRR*, quant-ph/0407005, 2004. arXiv: quant-ph/0407005.
- 26 Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.
- 27 Ai Liu and Meng Sun. A coalgebraic semantics framework for quantum systems. In Yamine Ait Ameer and Shengchao Qin, editors, *ICFEM 2019*, volume 11852 of *LNCS*, pages 387–402. Springer, 2019. doi:10.1007/978-3-030-32409-4_24.
- 28 Gui-lu Long, Fu-guo Deng, Chuan Wang, Xi-Han Li, Kai Wen, and Wan-Ying Wang. Quantum secure direct communication and deterministic secure quantum communication. *Frontiers of Physics in China*, 2(3):251–272, 2007. doi:10.1007/s11467-007-0050-3.
- 29 Dominic Mayers. Unconditional security in quantum cryptography. *Journal of the ACM*, 48(3):351–406, 2001. doi:10.1145/382780.382781.
- 30 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
- 31 Ali Ibnun Nurhadi and Nana Rachmana Syambas. Quantum key distribution (QKD) protocols: A survey. In *ICWT 2018*, pages 1–5. IEEE, 2018. doi:10.1109/ICWT.2018.8527822.
- 32 Hiroshi Ogawa. Coalgebraic approach to equivalences of quantum systems. Master’s thesis, University of Tokyo, 2014.
- 33 Sam Staton. Relating coalgebraic notions of bisimulation. *Logical Methods in Computer Science*, 7(1), 2011. doi:10.2168/LMCS-7(1:13)2011.
- 34 Yong Wang. Probabilistic process algebra to unifying quantum and classical computing in closed systems. *International Journal of Theoretical Physics*, 58(10):3436–3509, 2019. doi:10.1007/s10773-019-04216-2.
- 35 Peiying Zhang, Ning Chen, Shigen Shen, Shui Yu, Sheng Wu, and Neeraj Kumar. Future quantum communications and networking: A review and vision. *IEEE Wireless Communications*, 31(1):141–148, 2024. doi:10.1109/MWC.012.2200295.

A Proofs of Section 3

► **Proposition 40.** *Let E_1 and E_2 be two effects. If $E_1 + E_2 = |\psi\rangle\langle\psi|$ then $E_i = p_i |\psi\rangle\langle\psi|$ for some p_i , $i = 1, 2$. If $E_1 \oplus_p E_2 = |\psi\rangle\langle\psi|$ then $E_i = |\psi\rangle\langle\psi|$ for $i = 1, 2$.*

► **Theorem 4.** *Effect distributions correspond to all and only the parameterized sub-probability distributions that are convex and have an “overall” finite support.*

$$\mathcal{Q}_Q \cong \left\{ \mathfrak{D} \downarrow_{-} \in (\mathcal{D}X)^{DM_Q} \mid \mathfrak{D} \downarrow_{\rho_p \oplus \sigma} = (\mathfrak{D} \downarrow_{\rho}) \oplus_p (\mathfrak{D} \downarrow_{\sigma}) \text{ and } \bigcup_{\rho \in DM_Q} \text{supp}(\mathfrak{D} \downarrow_{\rho}) \text{ is finite} \right\}$$

Proof. Let d be the dimension of \mathcal{H}_Q . Recall that $(Ef_d, 0_d, +)$ is a Partial Commutative Monoid (PCM) [21]. Each PCM has a partial order, defined as $a \preceq b$ if and only if $\exists c. a + c = b$. In the case of Ef_d , \preceq is the Löwner order \sqsubseteq . We employ a known result in quantum theory [18]: Ef_d is isomorphic to $\mathbf{Conv}(DM_d, [0, 1])$. Moreover, $\mathbf{Conv}(DM_d, [0, 1])$ forms a PCM, where the monoid identity is $\lambda\rho.0$ and the summation of functions is defined pointwise. Since the isomorphism between Ef_d and $\mathbf{Conv}(DM_d, [0, 1])$ is a PCM isomorphism, it follows that

$$\mathcal{Q}_d \cong \left\{ \mathfrak{D} : X \rightarrow DM_d \rightarrow [0, 1] \mid \begin{array}{l} \forall x \mathfrak{D}(x) \text{ is convex, } \text{supp}(\mathfrak{D}) \text{ is finite} \\ \sum_{x \in \text{supp}(\mathfrak{D})} \mathfrak{D}(x) \preceq \lambda\rho.1 \end{array} \right\}$$

16:18 Effect Semantics for Quantum Process Calculi

where $\text{supp}(\mathfrak{D})$ is defined as $\{x \in X \mid \mathfrak{D}(x) \neq \lambda\rho.0\}$ and \preceq is the pointwise ordering between functions. The theorem follows by proving that the set above is isomorphic to

$$\left\{ \mathfrak{D} \downarrow_{-} : DM_d \rightarrow X \rightarrow [0, 1] \left| \begin{array}{l} \mathfrak{D} \downarrow_{-} \text{ is convex, } \bigcup_{\rho} \text{supp}(\mathfrak{D} \downarrow_{\rho}) \text{ is finite} \\ \forall \rho \sum_{x \in \text{supp}(\mathfrak{D} \downarrow_{\rho})} \mathfrak{D} \downarrow_{\rho}(x) \leq 1 \end{array} \right. \right\}$$

To prove this isomorphism, we provide an invertible function $f(\mathfrak{D}) = \lambda\rho.\lambda x.\mathfrak{D}(x)(\rho)$ that preserves and reflects the three properties we are interested in. For convexity, we have that

$$\begin{aligned} \forall x \mathfrak{D}(x) \text{ is convex} &\Leftrightarrow \forall x \mathfrak{D}(x)(\rho \oplus \sigma) = (\mathfrak{D}(x)(\rho)) \oplus (\mathfrak{D}(x)(\sigma)) \\ &\Leftrightarrow \forall x f(\mathfrak{D})(\rho \oplus \sigma)(x) = (f(\mathfrak{D})(\rho)(x)) \oplus (f(\mathfrak{D})(\sigma)(x)) \\ &\Leftrightarrow f(\mathfrak{D})(\rho \oplus \sigma) = f(\mathfrak{D})(\rho) \oplus f(\mathfrak{D})(\sigma) \Leftrightarrow f(\mathfrak{D}) \text{ is convex} \end{aligned}$$

For the finite support, we have that

$$\begin{aligned} \text{supp}(\mathfrak{D}) &= \{x \in X \mid \mathfrak{D}(x) \neq \lambda\rho.0\} = \{x \in X \mid \exists \rho. \mathfrak{D}(x)(\rho) \neq 0\} \\ &= \bigcup_{\rho} \{x \in X \mid \mathfrak{D}(x)(\rho) \neq 0\} = \bigcup_{\rho} \text{supp}(f(\mathfrak{D})) \end{aligned}$$

For the sum over the support, we have that

$$\begin{aligned} \sum_{x \in \text{supp}(\mathfrak{D})} \mathfrak{D}(x) \preceq \lambda\rho.1 &\Leftrightarrow \forall \rho. \sum_{x \in \text{supp}(\mathfrak{D})} \mathfrak{D}(x)(\rho) \leq 1 \Leftrightarrow \forall \rho. \sum_{\substack{x \in \text{supp}(\mathfrak{D}) \\ \mathfrak{D}(x)(\rho) \neq 0}} \mathfrak{D}(x)(\rho) \leq 1 \\ &\Leftrightarrow \forall \rho. \sum_{\text{supp}(f(\mathfrak{D})\rho)} \mathfrak{D}(x)(\rho) \leq 1 \Leftrightarrow \forall \rho. \sum_{\text{supp}(f(\mathfrak{D})\rho)} f(\mathfrak{D})(\rho)(x) \leq 1 \quad \blacktriangleleft \end{aligned}$$

► **Lemma 41.** *Let $\{s_{\alpha}, s_{\beta}, s_{\gamma}, s_{\delta}\} \subseteq X$, and let $\mathfrak{D} \in \mathcal{Q}_Q X$ be defined as $\mathfrak{D} = \{s_{\alpha} \triangleright |\Phi^+\rangle\langle\Phi^+|, s_{\beta} \triangleright |\Phi^-\rangle\langle\Phi^-|, s_{\gamma} \triangleright |\Psi^+\rangle\langle\Psi^+|, s_{\delta} \triangleright |\Psi^-\rangle\langle\Psi^-|\}$, where $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and $|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$. There is no $\mathfrak{T} \in \mathcal{Q}_Q^{\boxplus} X$ and subsets $X_{\alpha}, X_{\beta}, X_{\gamma}, X_{\delta}$ of X such that $\sum_{x \in X_y} \mathfrak{T}(x) = \mathfrak{D}(s_y)$ for $y \in \{\alpha, \beta, \gamma, \delta\}$.*

Proof. We proceed by induction on the number of application of \boxplus . No point distribution can verify this, hence the base case is trivial. Assume \mathfrak{T}_1 and \mathfrak{T}_2 can be defined by using \boxplus n times starting from point distributions, and let $\mathfrak{T} = \mathfrak{T}_1 \boxplus_E \mathfrak{T}_2$. We proceed by cases on the dimension d of the Hilbert space of the effect E . If $d = 1$, then $E = p$ for some p and

$$\sum_{x \in X_y} p \cdot \mathfrak{T}_1(x) + (1-p) \cdot \mathfrak{T}_2(x) = p \cdot \sum_{x \in X_y} \mathfrak{T}_1(x) + (1-p) \cdot \sum_{x \in X_y} \mathfrak{T}_2(x) = \mathfrak{D}(s_y).$$

If p is 0 or 1, then $\mathfrak{T} = \mathfrak{T}_1$ or \mathfrak{T}_2 , and the result directly follows from induction hypothesis. Otherwise, since $\mathfrak{D}(s_y)$ is of the form $|\psi\rangle\langle\psi|$ for each y , by Proposition 40 both $\sum_{x \in X_y} \mathfrak{T}_1(x)$ and $\sum_{x \in X_y} \mathfrak{T}_2(x)$ are equal to $\mathfrak{D}(s_y)$.

Consider now the case $d = 2$, then $\mathfrak{T}_1, \mathfrak{T}_2$ must be of dimension 2, and it must be that

$$\sum_{x \in X_{\alpha}} E \boxtimes \mathfrak{T}_1(x) + (\mathbb{I} - E) \boxtimes \mathfrak{T}_2(x) = E \boxtimes \sum_{x \in X_y} \mathfrak{T}_1(x) + (\mathbb{I} - E) \boxtimes \sum_{x \in X_y} \mathfrak{T}_2(x) = |\Phi^+\rangle\langle\Phi^+|.$$

By Proposition 40, $E \boxtimes \sum_{x \in X_y} \mathfrak{T}_1(x)$ must be equal to $p \cdot |\Phi^+\rangle\langle\Phi^+|$ for some p . But then, $\frac{1}{p} E \boxtimes \sum_{x \in X_y} \mathfrak{T}_1(x) = |\Phi^+\rangle\langle\Phi^+|$ contradicting the inseparability of $|\Phi^+\rangle\langle\Phi^+|$.

The dimension d cannot be 3 since \mathfrak{D} is of dimension 4.

If $d = 4$, then \mathfrak{T}_1 and \mathfrak{T}_2 can only be of dimension 1, and the effects in \mathfrak{D} must be all expressible as pE or $p(\mathbb{I} - E)$ for some probability p , but this is not the case.

Finally, note that d cannot be greater than 4, because \mathcal{H}_Q is of dimension 4. ◀

► **Theorem 11.** *If $|X| \geq 4$ and $|Q| \geq 2$, with $|\cdot|$ the cardinality, then $\mathcal{Q}_Q^{\boxplus} X \neq \mathcal{Q}_Q X$.*

Proof. For $|Q| = 2$, i.e. \mathcal{H}_Q of dimension 4, it is sufficient to note that this equivalence would contradict Lemma 41. This trivially generalizes to higher dimensional Hilbert spaces. ◀

► **Lemma 13.** *For all \mathcal{R} , $\mathcal{D} \widehat{\mathcal{R}}_Q \mathfrak{T}$ iff there exist a set of indices I and a set of effects $\{E_i \in \text{Ef}_Q\}_{i \in I}$ such that $\mathcal{D} = \{x_i \triangleright E_i\}_{i \in I}$, $\mathfrak{T} = \{y_i \triangleright E_i\}_{i \in I}$, and $x_i \mathcal{R} y_i$ for any $i \in I$.*

Proof. (\Leftarrow) Suppose there is a finite index set I such that (1) $\mathcal{D} = \{s_i \triangleright E_i\}_{i \in I}$, (2) $\mathfrak{T} = \{t_i \triangleright E_i\}_{i \in I}$ and (3) $s_i \mathcal{R} t_i$ for each $i \in I$. By (3) and by definition, it follows that $\bar{s}_i \widehat{\mathcal{R}}_{\emptyset} \bar{t}_i$ for each $i \in I$. Then, by Definition 12, $\mathcal{D} = (\sum_{i \in I} E_i \boxtimes \bar{s}_i) \widehat{\mathcal{R}}_Q (\sum_{i \in I} E_i \boxtimes \bar{t}_i) = \mathfrak{T}$.

(\Rightarrow) By induction on the rules for $\widehat{\mathcal{R}}_Q$. For the first rule, assume $s \mathcal{R} t$ and $\bar{s} \widehat{\mathcal{R}}_{\emptyset} \bar{t}$, then $\bar{s} = \{s \triangleright 1\}$ and $\bar{t} = \{t \triangleright 1\}$. For the second rule, assume $\mathcal{D}_i \widehat{\mathcal{R}}_Q \mathfrak{T}_i$. Then by induction hypothesis, for any $i \in I$, it holds that $\mathcal{D}_i = \{s_{i,j} \triangleright E_{i,j}\}_{j \in i_i}$ and $\mathfrak{T}_i = \{t_{i,j} \triangleright E_{i,j}\}_{j \in i_i}$, with $s_{i,j} \mathcal{R} t_{i,j}$. Hence it is true that $\sum_{i \in I} E_i \boxtimes \mathcal{D}_i = \{s_{i,j} \triangleright E_i \boxtimes E_{i,j}\}_{i \in I, j \in i_i}$ and $\sum_{i \in I} E_i \boxtimes \mathfrak{T}_i = \{t_{i,j} \triangleright E_i \boxtimes E_{i,j}\}_{i \in I, j \in i_i}$, thus the result follows by definition. ◀

► **Theorem 19.** *For any eLTS over Ef_Q with states S , $\sim_{am} \subseteq \sim_{ls}$. Moreover, $\sim_{am} = \sim_{ls}$ if $Q = \emptyset$, and $\sim_{am} \subsetneq \sim_{ls}$ if Q is of dimension at least 2 and S of cardinality at least 4.*

Proof. For \subseteq it is sufficient to show that $\mathcal{D} \widehat{\mathcal{R}} \mathfrak{T}$ requires \mathcal{D} and \mathfrak{T} to assign the same effect to each class in S/\mathcal{R} , by Lemma 13. The equality $\sim_{am} = \sim_{ls}$ in eLTSs of dimension one is a classical for pLTSs [19]. Then it suffices to consider Example 18. ◀

► **Theorem 23.** *For any $s, t \in S$, $s \sim_{ls} t$ if and only if $s \sim_{lpp} t$.*

Proof. It is easy to show that \sim_{ls} is a lpp-bisimulation and that \sim_{lpp} is a ls-bisimulation. For the first direction, take $s \sim_{ls} t$ and suppose that $s \xrightarrow{\mu} \mathcal{D}$, then there exists $t \xrightarrow{\mu} \mathfrak{T}$ such that $\forall C \in S/\sim_{ls} \mathcal{D}(C) = \mathfrak{T}(C)$, where $\mathcal{D}(C) = \sum_{x \in C} \mathcal{D}(x)$, and similarly for \mathfrak{T} . In other words, we know that \mathcal{D} and \mathfrak{T} are identical when considered as effect distributions on the set of equivalence classes. Thus, applying Theorem 4, we know that $\mathcal{D} \downarrow_{\sim} = \mathfrak{T} \downarrow_{\sim}$, i.e. that for any ρ they give the same probability distribution on equivalence classes, as required by the definition of lpp-bisimulation.

The other direction is identical, employing the isomorphism of Theorem 4. ◀

► **Lemma 42.** *Given a set of effects \mathbb{E} of a fixed dimension, there exists a state ρ such that $\forall i, j \in \mathbb{E}. \text{tr}(E_i \rho_{\mathbb{E}}) = \text{tr}(E_j \rho_{\mathbb{E}})$ iff $i = j$.*

Proof. Note that for any pair of distinct effects E_i, E_j there is a state $\rho_{i,j}$ such that $\text{tr}(E_i \rho_{i,j}) \neq \text{tr}(E_j \rho_{i,j})$. Let $p_{i,j}^k = \text{tr}(E_k \rho_{i,j})$. Note also that $\{p_{i,j}^k\}_{i,j,k}$ is in the algebraic closure of $\mathbb{Q} \cup T$ with T a finite set of transcendental numbers.

Let $q_{i,j}$ be transcendental numbers not in T such that for each i, j , $q_{i,j}$ is not in the algebraic closure of $\mathbb{Q} \cup T \cup \{q_{a,b} \mid a \neq i \text{ or } b \neq j\}$ (there are enough transcendental numbers, otherwise we could prove \mathbb{R} to be denumerable). We let q' be defined as $(1 - \sum_{i,j} q_{i,j})$, and we use it to scale the $q_{i,j}$ to the weights of a full probability distribution, letting $x_{i,j} = q_{i,j} q'$.

We let $\rho_{\mathbb{E}} = \sum_{i,j} x_{i,j} \rho_{i,j}$ and prove by refutation that it distinguishes all the effects in \mathbb{E} . Assume that $\text{tr}(E_a \rho_{\mathbb{E}}) = \text{tr}(E_b \rho_{\mathbb{E}})$ for some indices $a \neq b$. We observe that for $k \in \{a, b\}$

$$\text{tr}(E_k \rho_{\mathbb{E}}) = \sum_{i,j} x_{i,j} \text{tr}(E_k \rho_{i,j}) = \sum_{i,j} x_{i,j} p_{i,j}^k = q' \sum_{i,j} q_{i,j} p_{i,j}^k.$$

Hence, we can rewrite our assumption as $\sum_{i,j} q_{i,j} p_{i,j}^a = \sum_{i,j} q_{i,j} p_{i,j}^b$. Note that for each pair of indices c and d we can rewrite the formula above as

$$q_{c,d} (p_{c,d}^a - p_{c,d}^b) = \sum_{i,j \neq c,d} q_{i,j} p_{i,j}^b - \sum_{i,j \neq c,d} q_{i,j} p_{i,j}^a$$

If for some c or d , $p_{c,d}^a - p_{c,d}^b$ is not zero, then we can divide both sides for $p_{c,d}^a - p_{c,d}^b$, proving that $q_{c,d}$ is indeed in the algebraic closure of $\mathbb{Q} \cup T \cup \{q_{e,f} \mid e \neq c \text{ or } f \neq d\}$. Since this would contradict our hypothesis, we must assume that $p_{c,d}^a - p_{c,d}^b = 0$ for any choice of c and d , but this is a contradiction with the definition of $p_{i,j}^k$, since $p_{a,b}^a \neq p_{a,b}^b$ by construction. \blacktriangleleft

► **Theorem 24.** *For any $s, t \in S$, $s \sim_{ls} t$ implies $s \simeq_{pbe} t$. Moreover, if S is finitely dimensional, then $s \simeq_{pbe} t$ implies $s \sim_{ls} t$.*

Proof. By Theorem 23, for proving $\sim_{ls} \subseteq \simeq_{pbe}$ it suffices to show that $\sim_{lpp} \subseteq \simeq_{pbe}$, which holds by definition.

For $\simeq_{pbe} \subseteq \sim_{ls}$, let n be the cardinality of S , and consider the set of effects that appears in the eLTS $\mathbb{E}^0 = \{E \mid \exists s, s' \in S, \mu \in Act. s \xrightarrow{\mu} \mathfrak{D} \text{ and } \mathfrak{D}(s') = E\}$. We let \mathbb{E} be the set of the effects obtained by summing up to n effects in \mathbb{E}^0 , i.e. effects that are possibly associated with some equivalence class. By Lemma 42, there is a quantum state $\rho_{\mathbb{E}}$ such that $\forall E_i, E_j \in \mathbb{E}. tr(E_i \rho_{\mathbb{E}}) = tr(E_j \rho_{\mathbb{E}})$ iff $E_i = E_j$. Note that $\simeq_{pbe} \subseteq \sim_{\rho_{\mathbb{E}}}$ by definition of \simeq_{pbe} . Note also that by proving $\sim_{\rho_{\mathbb{E}}} \subseteq \sim_{ls}$ we would get the thesis by transitivity.

We will prove that $\sim_{\rho_{\mathbb{E}}}$ is a LS-bisimulation. Assume $s \sim_{\rho_{\mathbb{E}}} t$, and that $s \xrightarrow{\mu} \mathfrak{D}$, then $t \xrightarrow{\mu} \mathfrak{T}$ with $\mathfrak{D} \downarrow_{\rho_{\mathbb{E}}} \widehat{\sim}_{\rho_{\mathbb{E}^1}} \mathfrak{T} \downarrow_{\rho_{\mathbb{E}}}$. Note that, since LS and AM-bisimilarity coincides in the probabilistic case, the relation above implies that $\forall C \in S / \sim_{\rho_{\mathbb{E}}}. \sum_{x \in C} \mathfrak{D} \downarrow_{\rho_{\mathbb{E}}}(x) = \sum_{x \in C} \mathfrak{T} \downarrow_{\rho_{\mathbb{E}}}(x)$.

We now have to prove that $\forall C \in S / \sim_{\rho_{\mathbb{E}}}. \sum_{x \in C} \mathfrak{D}(x) = \sum_{x \in C} \mathfrak{T}(x)$. Assume by refutation that there is a C such that the condition does not hold. Then it suffices to note that

$$\begin{aligned} \sum_{x \in C} \mathfrak{D} \downarrow_{\rho_{\mathbb{E}}}(x) &= \sum_{x \in C} tr(\mathfrak{D}(x) \rho_{\mathbb{E}}) = tr\left(\left(\sum_{x \in C} \mathfrak{D}(x)\right) \rho_{\mathbb{E}}\right) \\ \sum_{x \in C} \mathfrak{T} \downarrow_{\rho_{\mathbb{E}}}(x) &= \sum_{x \in C} tr(\mathfrak{T}(x) \rho_{\mathbb{E}}) = tr\left(\left(\sum_{x \in C} \mathfrak{T}(x)\right) \rho_{\mathbb{E}}\right) \end{aligned}$$

Since $\sum_{x \in C} \mathfrak{D}(x)$ and $\sum_{x \in C} \mathfrak{T}(x)$ are both effects in \mathbb{E} , we have that $tr\left(\left(\sum_{x \in C} \mathfrak{D}(x)\right) \rho_{\mathbb{E}}\right) = tr\left(\left(\sum_{x \in C} \mathfrak{T}(x)\right) \rho_{\mathbb{E}}\right)$ implies $\sum_{x \in C} \mathfrak{D}(x) = \sum_{x \in C} \mathfrak{T}(x)$ contradicting our assumption. \blacktriangleleft

In the following we write $\mathfrak{D}|_{\rho}$ for $\{s_i|_{\rho} \triangleright tr_{Q'}(E_i(\rho \boxtimes \mathbb{I}_{Q \setminus Q'}))\}_{i \in I}$ with $\mathfrak{D} = \{s_i \triangleright E_i\}_{i \in I}$.

► **Theorem 28.** *If $s \sim_{ls} t$ then $s|_{\rho} \sim_{ls} t|_{\rho}$.*

Proof. We prove $\mathcal{R} = \{(s|_{\rho}, t|_{\rho}) \mid s \sim_{ls} t\}$ to be a ls-bisimulation. Take $(s|_{\rho}, t|_{\rho}) \in \mathcal{R}$, and assume $s|_{\rho}$ performs a reduction, then, by PEVAL it must be that $s \xrightarrow{\mu} \mathfrak{D}$. Since $s \sim_{ls} t$, there exists \mathfrak{T} such that $t \xrightarrow{\mu} \mathfrak{T}$ and $\forall C \in S / \sim_{ls}. \sum_{x \in C} \mathfrak{D}(x) = \sum_{x \in C} \mathfrak{T}(x)$. Moreover, $t|_{\rho} \xrightarrow{\mu} \mathfrak{T}|_{\rho}$ by PEVAL. We are left with proving that $\forall C \in S / \mathcal{R}. \sum_{x \in C} \mathfrak{D}(x)|_{\rho} = \sum_{x \in C} \mathfrak{T}(x)|_{\rho}$. Note that, by definition of $\mathcal{R} \subseteq S / \sim_{ls}$ if and only if $\{x|_{\rho} \mid x \in C\} \in S / \mathcal{R}$. Therefore, we can rewrite our condition as $\forall C \in S / \sim_{ls}. \sum_{x \in C} \mathfrak{D}(x)|_{\rho} = \sum_{x \in C} \mathfrak{T}(x)|_{\rho}$, which clearly derives from $\forall C \in S / \sim_{ls}. \sum_{x \in C} \mathfrak{D}(x) = \sum_{x \in C} \mathfrak{T}(x)$, by definition of $\mathfrak{D}|_{\rho}$. \blacktriangleleft

► **Lemma 43.** *For any eLTS (S, Act, \rightarrow) over Ef_Q and state $\rho \in DM_Q$, given a relation $\mathcal{R} \subseteq S \times S$ we have that \mathcal{R} is a ρ -bisimulation if and only if $\mathcal{R}|_{\rho}$ is a bisimulation, where $\mathcal{R}|_{\rho}$ is defined as $s|_{\rho} \mathcal{R}|_{\rho} t|_{\rho}$ if and only if $s \mathcal{R} t$.*

Proof. Note that for any two distribution $\mathfrak{D}, \mathfrak{T}$, it holds $\mathfrak{D} \downarrow_{\rho} \overset{\circ}{\mathcal{R}} \mathfrak{T} \downarrow_{\rho}$ iff $\mathfrak{D}|_{\rho} \overset{\circ}{\mathcal{R}}|_{\rho} \mathfrak{T}|_{\rho}$ since $\mathfrak{D} \downarrow_{\rho}$ and $\mathfrak{D}|_{\rho}$ assign the same probability to same elements, modulo the $|_{\rho}$ renaming.

We must now prove that $\mathcal{R}|_{\rho}$ is a bisimulation. Suppose $s|_{\rho} \mathcal{R}|_{\rho} t|_{\rho}$, then if $s|_{\rho} \xrightarrow{\mu} \mathfrak{D}|_{\rho}$ it must be $s \xrightarrow{\mu} \mathfrak{D}$. As t is ρ -bisimilar, we know that $t \xrightarrow{\mu} \mathfrak{T}$ and $\mathfrak{D} \downarrow_{\rho} \overset{\circ}{\mathcal{R}} \mathfrak{T} \downarrow_{\rho}$, because, since they are probability distributions, the equivalence class condition of ρ bisimilarity is equivalent to the relational lifting. Thus we get $\mathfrak{D}|_{\rho} \overset{\circ}{\mathcal{R}}|_{\rho} \mathfrak{T}|_{\rho}$, showing that $\mathcal{R}|_{\rho}$ is a bisimulation. \blacktriangleleft

► **Corollary 29.** *Given a finite eLTS (S, Act, \rightarrow) over Ef_Q and $s, t \in S$, if for any $\rho \in DM_Q$ we have $s|_\rho \sim_{ls} t|_\rho$, then $s \sim_{ls} t$.*

Proof. Take $\mathcal{R} = \{(x, y) \mid x|_\rho \sim_{ls} y|_\rho\}$. The relation $\mathcal{R}|_\rho$ is a bisimulation since if $x|_\rho \xrightarrow{\mu} \mathfrak{D}|_\rho$ we have $y|_\rho \xrightarrow{\mu} \mathfrak{F}|_\rho$, and $\mathfrak{D}|_\rho, \mathfrak{F}|_\rho$ are not only in $\overset{\circ}{\sim}_{ls}$, but also in $\overset{\circ}{\mathcal{R}}|_\rho$. By Lemma 43 \mathcal{R} is a ρ -bisimulation, and s, t are ρ -bisimilar for any ρ . Thus by Theorem 24 they are bisimilar. ◀

B Proofs of Section 4

Recall that we write \sim for the LS-bisimilarity \sim_{ls} .

► **Lemma 44.** *Consider a Ef_Q eLTS and $\mathcal{E} : DM_Q \rightarrow DM_{Q'}$ with $Q' \subseteq Q$. For any s , if $\langle \mathcal{E}, s \rangle \xrightarrow{\mu} \mathfrak{D}$, then there are states s_i and superoperators $\mathcal{E}_i : DM_Q \rightarrow DM_{Q''}$ with $Q'' \subseteq Q'$ such that $\mathfrak{D} = \{\langle \mathcal{E}_i, s_i \rangle \triangleright \mathfrak{Z}_i(\mathbb{I}_{Q''})\}$. Moreover, for all $\rho \in DM_Q$, $\langle \mathcal{E}(\rho), s \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i(\rho), s_i \rangle \triangleright tr(\mathcal{E}_i(\rho))\}$.*

Proof. By induction on the rules of the Heisenberg semantics.

(case $HPRE$) Consider the transition $\langle \mathcal{E}, \mu.([E_i]s_i)_{i \in I} \rangle \xrightarrow{\mu} \{\langle \mathcal{M}_{E_i} \circ \mathcal{E}, s_i \rangle \triangleright \mathfrak{Z}(E_i \boxtimes \mathbb{I}_{Q \setminus Q'})\}_{i \in I}$.

Let $\mathcal{E}_i = \mathcal{M}_{E_i} \circ \mathcal{E}$. The first point follows from duality, \mathcal{E}_i is $\mathfrak{Z}_i(F) = \mathfrak{Z} \circ (E_i \boxtimes F)$.

For the second point, take any ρ , and apply the $SPRE$ rule: $\langle \mathcal{E}(\rho), \mu.([E_i]s_i)_{i \in I} \rangle \xrightarrow{\mu} \{\langle \mathcal{M}_{E_i}(\mathcal{E}(\rho)), s_i \rangle \triangleright tr(\mathcal{M}_{E_i}(\mathcal{E}(\rho)))\}_{i \in I}$. The result holds by definition since $\mathcal{E}_i = \mathcal{M}_{E_i} \circ \mathcal{E}$.

(case HU) Consider the transition $\langle \mathcal{E}, U; s \rangle \xrightarrow{\mu} \mathfrak{D}$. By induction hypothesis, $\langle \mathcal{U} \circ \mathcal{E}, s \rangle \xrightarrow{\mu} \mathfrak{D}$ and $\mathfrak{D} = \{\langle \mathcal{E}_i, s_i \rangle \triangleright \mathfrak{Z}_i(\mathbb{I}_{Q''})\}$, trivially proving the first point. By induction hypothesis, it also holds that for any ρ , $\langle (\mathcal{U} \circ \mathcal{E})(\rho), s \rangle \xrightarrow{\mu} \mathfrak{F} = \{\langle \mathcal{E}_i(\rho), s_i \rangle \triangleright tr(\mathcal{E}_i(\rho))\}$. Then, the result holds by considering the rule SU : $\langle \mathcal{E}(\rho), U; s \rangle \xrightarrow{\mu} \mathfrak{F}$.

(case $HSYNCL$) Consider the transition $\langle \mathcal{E}, r \parallel t \rangle \xrightarrow{\tau} \{\langle \mathcal{E}_j, r_k \parallel t_j \rangle \triangleright E_j\}_{(k,j) \in K \times J_k}$. By induction hypothesis, we know that

$$\begin{aligned} \langle \mathcal{E}, r \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_k, r_k \rangle \triangleright \mathfrak{Z}_k(\mathbb{I})\}_{k \in K} &\Rightarrow \forall \rho. \langle \mathcal{E}(\rho), r \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_k(\rho), r_k \rangle \triangleright p_k\}_{k \in K} \\ \langle \mathcal{E}_k, t \rangle \xrightarrow{\bar{\mu}} \{\langle \mathcal{E}_j, t_j \rangle \triangleright \mathfrak{Z}_j(\mathbb{I})\}_{j \in J_k} &\Rightarrow \forall \rho. \langle \mathcal{E}_k(\rho), t \rangle \xrightarrow{\bar{\mu}} \{\langle \mathcal{E}_j(\rho), t_j \rangle \triangleright tr(\mathcal{E}_j(\rho))\}_{j \in J_k} \end{aligned}$$

since $E_j = \mathfrak{Z}_j(\mathbb{I})$, $I = K \times J_k$, $s_i = r_k \parallel t_j$, $\mathcal{E}_i = \mathcal{E}_j$, the first point holds by definition. Take any ρ , and apply $SSYNCL$. Then $\langle \mathcal{E}(\rho), r \parallel t \rangle \xrightarrow{\tau} \{\langle \mathcal{E}_j(\rho), r_k \parallel t_j \rangle \triangleright tr(\mathcal{E}_j(\rho))\}_{(k,j) \in K \times J_k}$, thus proving the second point. All the other cases follow by induction. ◀

► **Lemma 45.** *Let $\mathcal{E} : DM_Q \rightarrow DM_{Q'}$ with $Q' \subseteq Q$. For any s and any $\rho \in DM_Q$, if $\langle \mathcal{E}(\rho), s \rangle \xrightarrow{\mu} \mathfrak{D}$, then there exists states s_i and superoperators $\mathcal{E}_i : DM_Q \rightarrow DM_{Q''}$ with $Q'' \subseteq Q'$ such that $\mathfrak{D} = \{\langle \mathcal{E}_i(\rho), s_i \rangle \triangleright tr(\mathcal{E}_i(\rho))\}$. Moreover, there exists a transition $\langle \mathcal{E}, s \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i, s_i \rangle \triangleright \mathfrak{Z}_i(\mathbb{I}_{Q''})\}$.*

Proof. We proceed by induction on the rules of the Schrödinger semantics.

(case $SPRE$) Consider $\langle \mathcal{E}(\rho), \mu.([E_i]s_i)_{i \in I} \rangle \xrightarrow{\mu} \{\langle \mathcal{M}_{E_i}(\mathcal{E}(\rho)), s_i \rangle \triangleright tr(\mathcal{M}_{E_i}(\mathcal{E}(\rho)))\}_{i \in I}$. Then $\mathcal{E}_i = \mathcal{M}_{E_i} \circ \mathcal{E}$. Furthermore, by rule $HPRE$ $\langle \mathcal{E}, \mu.([E_i]s_i)_{i \in I} \rangle \xrightarrow{\mu} \{\langle \mathcal{M}_{E_i} \circ \mathcal{E}, s_i \rangle \triangleright \mathfrak{Z}(E_i \boxtimes \mathbb{I}_{Q \setminus Q'})\}_{i \in I}$, with $\mathfrak{Z}_i(F) = \mathfrak{Z} \circ (E_i \boxtimes F)$ being the dual of \mathcal{E}_i .

(case SU) Consider the transition $\langle \mathcal{E}(\rho), U; s \rangle \xrightarrow{\mu} \mathfrak{F}$. Then, by induction hypothesis, $\langle \mathcal{U}(\mathcal{E}(\rho)), s \rangle \xrightarrow{\mu} \mathfrak{F} = \{\langle \mathcal{E}_i(\rho), s_i \rangle \triangleright tr(\mathcal{E}_i(\rho))\}$, and $\langle \mathcal{U} \circ \mathcal{E}, s \rangle \xrightarrow{\mu} \mathfrak{D} = \{\langle \mathcal{E}_i, s_i \rangle \triangleright \mathfrak{Z}_i(\mathbb{I}_{Q''})\}$. Then, the result holds by considering the rule HU , granting that $\langle \mathcal{E}, U; s \rangle \xrightarrow{\mu} \mathfrak{D}$.

16:22 Effect Semantics for Quantum Process Calculi

(case ssyncL) Consider the transition $\langle \mathcal{E}(\rho), r \parallel t \rangle \xrightarrow{\tau} \{\langle \rho_j, r_k \parallel t_j \rangle \triangleright p_j\}_{(k,j) \in K \times J_k}$. By induction hypothesis, the required premises hold, and they have the following form and that

$$\begin{aligned} \langle \mathcal{E}(\rho), r \rangle &\xrightarrow{\mu} \{\langle \mathcal{E}_k(\rho), r_k \rangle \triangleright p_k\}_{k \in K} \Rightarrow \langle \mathcal{E}, r \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_k, r_k \rangle \triangleright \mathcal{Z}_k(\mathbb{I})\}_{k \in K} \\ \langle \mathcal{E}_k(\rho), t \rangle &\xrightarrow{\bar{\mu}} \{\langle \mathcal{E}_j(\rho), t_j \rangle \triangleright \text{tr}(\mathcal{E}_j(\rho))\}_{j \in J_k} \Rightarrow \langle \mathcal{E}_k, t \rangle \xrightarrow{\bar{\mu}} \{\langle \mathcal{E}_j, t_j \rangle \triangleright \mathcal{Z}_j(\mathbb{I})\}_{j \in J_k} \end{aligned}$$

where $\rho_j = \mathcal{E}_j(\rho)$ and $p_j = \text{tr}(\mathcal{E}_j(\rho))$. The first point holds by definition. Take HSyncL . Then $\langle \mathcal{E}, r \parallel t \rangle \xrightarrow{\tau} \{\langle \mathcal{E}_j, r_k \parallel t_j \rangle \triangleright \mathcal{Z}_j(\mathbb{I})\}_{(k,j) \in K \times J_k}$, thus proving the second point. All the other cases follows by induction. \blacktriangleleft



► **Theorem 36.** *For any process P and $\rho \in \text{DM}_{Q_P}$, $\langle \mathcal{I}, P \rangle|_{\rho} \sim \langle \rho, P \rangle$.*

Proof. Take $\mathcal{R} = \{(\langle \mathcal{E}, R \rangle|_{\rho}, \langle \mathcal{E}(\rho), R \rangle) \mid Q \text{ is a process, and } \mathcal{E} : \text{Ef}_{Q_P} \rightarrow \text{Ef}_Q, Q \supseteq Q_R\}$. Take a pair $(\langle \mathcal{E}, R \rangle|_{\rho}, \langle \mathcal{E}(\rho), R \rangle)$ and assume that $\langle \mathcal{E}, R \rangle|_{\rho} \xrightarrow{\mu} \mathfrak{D}$. Then, by definition of $\cdot|_{\rho}$ and Lemma 44, we have $\langle \mathcal{E}, R \rangle \xrightarrow{\mu} \{\langle \mathcal{E}_i, s_i \rangle \triangleright \mathcal{Z}_i(\mathbb{I})\}_{i \in I}$ and $\mathfrak{D} = \{\langle \mathcal{E}_i, s_i \rangle|_{\rho} \triangleright \text{tr}((\mathcal{Z}_i(\mathbb{I}))\rho)\}_{i \in I}$. Moreover, by Lemma 44, $\langle \mathcal{E}(\rho), R \rangle \xrightarrow{\mu} \mathfrak{T} = \{\langle \mathcal{E}_i(\rho), s_i \rangle \triangleright \text{tr}(\mathcal{E}_i(\rho))\}_{i \in I}$. Note that $\mathfrak{D} \overset{\circ}{\mathcal{R}} \mathfrak{T}$ since $\langle \mathcal{E}_i, s_i \rangle|_{\rho} \mathcal{R} \langle \mathcal{E}_i(\rho), s_i \rangle$ and $\text{tr}((\mathcal{Z}_i(\mathbb{I}))\rho) = \text{tr}(\mathbb{I}(\mathcal{E}_i(\rho))) = \text{tr}(\mathcal{E}_i(\rho))$. The other direction is symmetric thanks to Lemma 45. \blacktriangleleft

► **Corollary 37.** *Given two processes P and R , $\langle \mathcal{I}, P \rangle \sim \langle \mathcal{I}, R \rangle$ if and only if $\langle \rho, P \rangle \sim \langle \rho, R \rangle$ for any $\rho \in \text{DM}_{Q_P}$.*

Proof. By Corollary 29 and Theorem 36. \blacktriangleleft



Invariants for One-Counter Automata with Disequality Tests

Dmitry Chistikov¹  

Centre for Discrete Mathematics and its Applications (DIMAP) &
Department of Computer Science, University of Warwick, Coventry, UK

Jérôme Leroux 

LaBRI, CNRS, Univ. Bordeaux, France

Henry Sinclair-Banks  

Centre for Discrete Mathematics and its Applications (DIMAP) &
Department of Computer Science, University of Warwick, Coventry, UK

Nicolas Waldburger  

IRISA, Université de Rennes, France

Abstract

We study the reachability problem for one-counter automata in which transitions can carry disequality tests. A disequality test is a guard that prohibits a specified counter value. This reachability problem has been known to be NP-hard and in PSPACE, and characterising its computational complexity has been left as a challenging open question by Almagor, Cohen, Pérez, Shirmohammadi, and Worrell [1]. We reduce the complexity gap, placing the problem into the second level of the polynomial hierarchy, namely into the class coNP^{NP} . In the presence of both equality and disequality tests, our upper bound is at the third level, $\text{P}^{\text{NP}^{\text{NP}}}$.

To prove this result, we show that non-reachability can be witnessed by a pair of invariants (forward and backward). These invariants are almost inductive. They aim to over-approximate only a “core” of the reachability set instead of the entire set. The invariants are also leaky: it is possible to escape the set. We complement this with separate checks as the leaks can only occur in a controlled way.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Inductive invariant, Vector addition system, One-counter automaton

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.17

Related Version *Full Version*: <https://arxiv.org/abs/2408.11908>

Funding *Dmitry Chistikov*: Supported in part by the Engineering and Physical Sciences Research Council [EP/X03027X/1].

Henry Sinclair-Banks: Supported by EPSRC Standard Research Studentship (DTP), grant number EP/T51794X/1. Also supported in part by the International Emerging Actions grant (IEA’22) and by the ANR grant VeSyAM (ANR-22-CE48-0005).

Nicolas Waldburger: Supported in part by the International Emerging Actions grant (IEA’22) and by the ANR grant VeSyAM (ANR-22-CE48-0005).

Acknowledgements We would like to thank Mahsa Shirmohammadi, without whom this work would not have been possible, for many ideas and encouragement. We are also grateful to Thomas Colcombet, Rayna Dimitrova, and James Worrell for useful discussions.

¹ During the work on this paper, DC was a visitor to the Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern and Saarbrücken, Germany, a visiting fellow at St Catherine’s College and a visitor to the Department of Computer Science at the University of Oxford, United Kingdom.



1 Introduction

It is well known that the computational complexity of problems is often sensitive to seemingly minor adjustments in the problem setting. Consider, for example, vector addition systems with states (VASS). Perhaps more commonly presented as Petri nets, VASS are a very simple yet powerful model of concurrency. Many important computational problems in logic, language theory, and formal verification reduce to or are even equivalent to the reachability problem in VASS (see, e.g., [29, 45]). However, a classical result due to Minsky shows that adding the capability to test counters for zero makes the problem undecidable [42].

At the same time, while reachability in VASS is known to be decidable [41], its computational complexity was recently shown to be extremely high, namely Ackermann-complete (see [39] for the upper bound and [19, 20, 38] for the lower bound), so from the practical point of view one might question the significance of the complexity jump arising from zero tests.

More recent, “down-to-earth”, and perhaps more striking is the following result on 1-dimensional VASS, which can be thought of as finite-state automata equipped with one counter (capable of storing a nonnegative integer). Reachability in these systems can be decided in NP [28] and is in fact NP-complete. It is not difficult to show, using the standard hill-cutting technique [48], that reachability can also always be witnessed by executions in which all values assumed by the counter are bounded from above by an exponential function in the size of the system and the bit length of counter values of the source and target configurations. Because of this, one might expect that placing an exponential bound on the counter values *upfront* does not change the problem much. But, in fact, the complexity jumps: the problem – which is equivalent to reachability in two-clock timed automata [15] – becomes PSPACE-complete [23]. One may say that, in this case, formal verification toolkit available for the reachability problem is not robust to this change in the problem setting.

In this paper we study a different seemingly benign variation of the standard reachability question. Consider one-counter automata in which transitions may test the value of the counter for *disequality* against a given integer (which depends on the transition). In other words, executions of the system can be blocked by disequality guards, which *prevent* the transition from being fired if the counter value is equal to a specified number. The initial motivation for studying this question comes from a model checking problem for flat Freeze LTL; see Demri and Sangnier [21] and Lechner, Mayr, Ouaknine, Pouly, and Worrell [35]. Additionally, recall that automata can be used for the modeling of imperative code; see, e.g., Hague and Lin [30, 31], as well as discussion in Section 2. Classical Minsky machines encode *if-then* conditionals with equality comparisons to constants, $x = k$. Simulating an *if-then-else* conditional of this type on a Minsky machine seems to require additional $O(k)$ states. If k is large, this growth in size may be exponential, even though *then* branches as well as increments $x += k$ and decrements $x -= k$ can be encoded directly. (If the machine model is extended with $x \leq k$ comparisons, then the asymmetry between *then* and *else* disappears, but reachability becomes PSPACE-complete [23].)

One might expect that, since only a small number of configurations are forbidden (by the disequality guards) in the infinitely large configuration space, the complexity of the problem should not change significantly and existing techniques should be applicable. This conclusion, however, has remained elusive. For the problem of reachability in one-counter automata with disequality tests, the exponential upper bound on counter values necessary for witnessing reachability carries over. But, despite progress on related problems [9, 11] (including the settling of the complexity of the above-mentioned flat Freeze LTL model checking problem [11]), it has not been possible to pin down the complexity of this problem,

which has been known to be NP-hard (even without disequality tests) and to belong to PSPACE (thanks to the exponential bound on the counter) [35]. The apparent simplicity of the problem contrasts with the lack of robustness of the available toolbox. It was recently shown by Almagor, Cohen, Pérez, Shirmohammadi, and Worrell [1] that the coverability (or state reachability) problem can be solved in polynomial time for this model, similarly to the standard 1-dimensional VASS without tests. The algorithm and its analysis, however, become sophisticated, and the complexity of reachability was left as a *challenging open problem*.

In the present paper, we make progress on this problem. Existing techniques need to be extended and developed significantly to handle seemingly benign disequality tests. We have been unable to find an easily verifiable witness for reachability, and instead show that *non-reachability* is witnessed by a form of invariants (or, more precisely, separators). The existence of counterexamples that violate such invariants can be checked in NP, thus placing the reachability problem for OCA with disequality tests into the second level of the polynomial hierarchy, namely in coNP^{NP} . This complexity class captures the complement of synthesis-type questions, which ask to find a single object (say, a circuit) that works correctly for all (exponentially many) inputs. In our problem, one configuration can reach another if and only if every potential invariant (of a form we describe) violates one of the invariance conditions; moreover, this violation can be checked in NP. In the presence of both equality and disequality tests, we need a slightly larger class $\text{P}^{\text{NP}^{\text{NP}}}$, at the third level of the hierarchy.

Our contributions. Traditionally, an invariant is an overapproximation of the set of reachable configurations which is inductive, i.e., closed under the transition relation. Our invariants are different in several ways:

1. We capture only some configurations within the reachability set, which form its *core*. Accordingly, we require a tailored notion of closure, namely closure under a restricted form of reachability relation.
2. Our invariants are leaky (*almost* inductive): an execution may escape the set. Allowing leaks is complemented by a separate controlling mechanism (check) that all leaks – which may occur at the interface between strongly connected components of the automaton – are safe.
3. To compose our local inductive invariants, i.e., those restricted to a single strongly connected component, the controlling mechanism for leaks relies on relaxed integer semantics for the execution. More precisely, we extend (to automata with disequality tests) a known technique [28] for lifting \mathbb{Z} -executions to actual executions.

Our notion of local invariants requires that we place a certain technical assumption at the interface (entry and exit points) of strongly connected components. To discharge this assumption, we use a combination of two invariants, one for the main (forward) VASS and another for the reverse VASS. Together, these two sets form a separator – a witness for non-reachability.

2 Related Work

Invariants. In formal verification, a forward exploration of countably infinite configuration spaces from the initial configuration, or a symmetrical backward exploration from the target, is a standard approach to reachability problems and targets bug finding. General heuristics can be used to improve such an exploration (see, for instance, the recent directed reachability algorithm [8]). However, in order to prove non-reachability, thus certifying the absence of bugs, an invariant-based approach is more popular.

Many techniques have been developed in the past for computing inductive invariants, depending on the structure of the underlying system based on counterexample-guided abstraction refinement [17], automata [32], property-directed reachability [3, 13, 14], and more generally in the abstract interpretation framework [18].

In vector addition systems, semilinear invariants [25] are sufficient for the general reachability problem [36]. Even if those invariants are intractable in general, for some instances, namely the control-state reachability problem, the implementation of efficient tools computing invariants (downward-closed sets in that case) is an active research area [7, 22] with implementation of tools [6, 24, 33].

In this paper we focus on 1-dimensional VASS in the presence of equality and disequality tests; we call them *one-counter automata (OCA) with tests*. The notion of local inductive invariants with leaks, which we propose, provides a way to reduce the search space of inductive invariants, by specifying the shape of the “core” of the invariant (a union of arithmetic progressions within “bounded chains”), as well as restricting the problem to each strongly connected component one by one. We view this as a compositional approach for computing inductive invariants. As a theoretical application, we prove that the reachability problem for one-counter automata with tests is between NP and $P^{NP^{NP}}$, and in fact in $coNP^{NP}$ if only disequality tests are present.

The previous work on OCA with disequality tests by Almagor et al. [1] enables us to focus (subject to a technical assumption) on configurations in a small number of bounded chains (see Section 4). The structure of the set of reachable configurations in these chains admits a short description. At the core of our invariants are exactly such sets, and we need an appropriate notion of “inductiveness”, a condition to control “leaks” that violate the assumption above, and a verification mechanism for all these conditions.

One-counter automata. OCA can be seen as an abstraction of pushdown automata, a widely used model of recursive systems. Conceptually very simple, OCA are at the heart of a number of results in formal verification; see, e.g., [2, 4, 34]. Multi-counter automata are used to model imperative code with numerical data types [30, 31]; roughly speaking, a reachability query is expressed in logic, as a formula in existential linear integer arithmetic. In these two references an additional pushdown stack is available, capturing recursive function calls. We refer the reader to [16] for a retrospective on underlying “pumping” results for OCA, crucial for many of the recent results. There is also a rich history of research on behavioural equivalences and model checking for a variety of one-counter processes and systems; see, e.g., [10, 27, 47, 48].

The above-mentioned result that reachability in OCA is NP-complete, by Haase, Kreutzer, Ouaknine, and Worrell [28], has recently been built upon to give a representation of the entire reachability relation in existential linear integer arithmetic, with an implementation available online, by Li, Chen, Wu, and Xia [40]. The idea of “lifting” candidate runs to actual runs, which is shown in [28] and which we develop further by adding support for disequality tests, has been used in other settings as well [5, 37, 41]. For example, a construction similar to our Lemma 7.1 is an element of the proof of a tight upper bound on the length of shortest runs in OCA without disequality tests [16]. In comparison to the latter paper, our construction need not consider divisibility properties of run lengths, but at the same time applies in a more general scenario: the updates of our OCA are specified in binary notation (that is, succinctly); and, naturally, our OCA may have disequality tests.

We already mentioned above that, despite appearing atypical at first glance, the disequality tests do in fact contribute to the modeling power: namely, when modeling code, these tests enable the simulation of the *else* branch in conditional statements comparing an integer variable for equality with some constant. The framework of Hague and Lin [30, 31] assumes that each counter variable can undergo at most k reversals (i.e., changes between “increasing” and “decreasing”), where k is fixed. This assumption is strong; without it, a reachability instance would require a logical formula of exponential size. Results of Haase et al. [28] and Li et al. [40] avoid this assumption, but, for the standard syntax of one-counter automata, *if-then-else* conditionals remain out of reach – or rather require an exponential expansion of the automaton. Our leaky invariants technique allows us to handle such conditionals with equality tests on counters, without assuming any bound on the number of reversals.

3 OCA with Equality and Disequality Tests

We denote by \mathbb{Z} and \mathbb{N} the set of all integers and all nonnegative integers, respectively.

A *constraint* is either an *equality test* of the form $x = k$ with $k \in \mathbb{N}$, a *disequality test* of the form $x \neq k$ with $k \in \mathbb{N}$, or simply **true**; x denotes here our counter, which is a nonnegative integer variable. Let \mathcal{C} denote the set of all possible constraints. A *one-counter automaton (OCA) with equality and disequality tests* is a triplet $\mathcal{A} = (Q, \Delta, \tau)$, where Q is a finite set of *states*, $\Delta \subseteq Q \times \mathbb{Z} \times Q$ is a finite set of *transitions* and $\tau: Q \rightarrow \mathcal{C}$ is the *constraint function*. The automaton \mathcal{A} is an *OCA with disequality tests* if the constraint function τ does not have any equality tests. We sometimes refer to the constraints as *guards*.²

Syntactically, \mathcal{A} can be seen as an integer-weighted graph with directed edges between states. Viewed this way, \mathcal{A} can be decomposed into a set of *strongly connected components (SCCs)*. The automaton \mathcal{A} is *strongly connected* when it has one strongly connected component only. A *path* π in \mathcal{A} is a sequence $\pi = (t_1, t_2, \dots, t_n)$ of transitions, where $t_i = (q_{i-1}, a_i, q_i)$ for each i and $n \geq 0$. We may refer to π as a q_0 - q_n path. The *length* of such a path is $\text{len}(\pi) \stackrel{\text{def}}{=} n$. The *effect* of π is $\text{eff}(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^n a_i$. A *cycle* is a path starting and ending at the same state; for $q \in Q$, a *q-cycle* is a q - q path. A path or cycle is *simple* if it contains no repetition of states, except that a *simple cycle* has the same starting and ending state. Every simple cycle has length less than or equal to $|Q|$.

The size of an OCA \mathcal{A} is the bit size of its encoding, where all numbers are written in binary. We write $\|\Delta\|$ and $\|\tau\|$ to refer to the maximum absolute value of a transition update and test, respectively.

Configurations and runs. The semantics of \mathcal{A} is defined based on the set of valid configurations and the reachability relation, as follows.

A *configuration* is a pair (q, z) comprising a state $q \in Q$ and a nonnegative integer $z \in \mathbb{N}$; we may refer to z as the *counter value*. We say that (q, z) is a *valid configuration* if it respects the constraint $\tau(q)$. Write $\text{Conf} \stackrel{\text{def}}{=} Q \times \mathbb{N}$ for the set of all configurations. Given two configurations $(q, z), (q', z')$ and $t \in \Delta$, we write $(q, z) \xrightarrow{t} (q', z')$ when $t = (q, z' - z, q')$; we denote by $(q, z) \rightarrow (q', z')$ the existence of such a transition. A *run* of \mathcal{A} is a sequence $(q_0, z_0), \dots, (q_n, z_n)$ of valid configurations, for $n \geq 0$, such that there exists a path (t_1, \dots, t_n) with $(q_{i-1}, z_{i-1}) \xrightarrow{t_i} (q_i, z_i)$. We say that (q_n, z_n) is *reachable* from (q_0, z_0)

² We use automata with constraints on states. Automata with constraints on transitions are, for our purposes, equivalent.

if there exists a run from (q_0, z_0) to (q_n, z_n) . We write $(q_0, z_0) \xrightarrow{*} (q_n, z_n)$ to denote the existence of such a run. Given a path π , we write $(q_0, z_0) \xrightarrow{\pi} (q_n, z_n)$ if π yields a run from (q_0, z_0) to (q_n, z_n) .

A path π has no hope to yield a run from (q, z) if $z + \text{eff}(\pi') < 0$ for some prefix π' of π . We denote by $\text{drop}(\pi)$ the maximum of $-\text{eff}(\pi')$ over all prefixes π' of π , and call it the *drop* of π . Intuitively, $\text{drop}(\pi)$ is the smallest counter value $z \in \mathbb{N}$ such that π , when applied from (q, z) , remains nonnegative; note that hitting a guard is not a consideration here.

We use the following standard operators: $\text{Post}(c) \stackrel{\text{def}}{=} \{c' \in \text{Conf} \mid c \rightarrow c'\}$ and $\text{Pre}(c) \stackrel{\text{def}}{=} \{c' \in \text{Conf} \mid c' \rightarrow c\}$. For $X \subseteq \text{Conf}$, we write $\text{Post}(X) \stackrel{\text{def}}{=} \bigcup_{c \in X} \text{Post}(c)$ and $\text{Pre}(X) \stackrel{\text{def}}{=} \bigcup_{c \in X} \text{Pre}(c)$. Also, $\text{Post}^*(X) \stackrel{\text{def}}{=} \{d \mid \exists c \in X : c \xrightarrow{*} d\}$ and $\text{Pre}^*(X) \stackrel{\text{def}}{=} \{c \mid \exists d \in X : c \xrightarrow{*} d\}$.

For an OCA $\mathcal{A} = (Q, \Delta, \tau)$, we define the *reverse* of \mathcal{A} as $\mathcal{A}^R \stackrel{\text{def}}{=} (Q, \Delta^R, \tau)$ where $(q, a, q') \in \Delta^R$ if and only if $(q', -a, q) \in \Delta$. Given configurations c and d and a path π in \mathcal{A} , we have $c \xrightarrow{*} d$ in \mathcal{A} if and only if $d \xrightarrow{*} c$ in \mathcal{A}^R .

The reachability problem. We consider the following decision problem.

REACHABILITY

Input: An OCA \mathcal{A} with equality and disequality tests, a valid initial configuration `src`, and a valid target configuration `trg`.

Output: Does `src` $\xrightarrow{*}$ `trg` hold?

The model of OCA with disequality tests has been studied in [35] and [1]. The latter paper provides polynomial-time algorithms for the *coverability problem*: “given `src` and a state q , does there exist z such that `src` $\xrightarrow{*} (q, z)$?” and the related *unboundedness problem*: “is the set of configurations reachable from `src` infinite?”. The reachability problem, however, is NP-hard even without tests, see Figure 1 (Left).

Equality tests. In the reachability problem in OCA with equality and disequality tests, the main technical challenge stems from disequality tests. Indeed, a state with an equality test only has one valid configuration hence need not be visited more than once.

We now work with OCA with disequality tests only. We will discuss in Section 7.5 how our techniques are affected by the addition of equality tests.

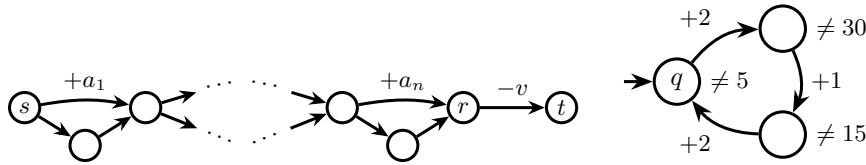
4 Getting Familiar with Disequality Tests

In this section, we fix an OCA $\mathcal{A} = (Q, \Delta, \tau)$ with disequality tests and two valid configurations `src` and `trg`.

A configuration c is *bounded* when $\text{Post}^*(c)$ is finite, and *unbounded* otherwise. It is known, although far from trivial, that one can decide boundedness in polynomial time.

► **Lemma 4.1** (see [1, Theorem 19]). *Given an OCA with disequality tests and a configuration c , it is decidable in polynomial time whether c is bounded or unbounded.*

A *candidate run* is a run except that neither the nonnegativity condition nor the disequality tests are necessarily respected. Formally, a candidate run is simply a sequence $(q_0, z_0), \dots, (q_n, z_n)$ where all $(q_i, z_i) \in Q \times \mathbb{Z}$, $n \geq 0$, and such that there exist transitions $(q_{i-1}, a_i, q_i) \in \Delta$ with $z_{i+1} = z_i + a_i$ for all $i \in \{1, \dots, n\}$. We write $(q_0, z_0) \xrightarrow[\mathbb{Z}]{*} (q_n, z_n)$.



■ **Figure 1 Left.** This OCA without tests is constructed from an instance of the subset sum problem (a_1, \dots, a_n, v) ; this is in fact how the reachability problem in OCA without tests is proved to be NP-hard in [28]. Configuration $(t, 0)$ can be reached from configuration $(s, 0)$ whenever there exists a subset of $\{a_1, \dots, a_n\}$ whose elements sum up to v . Note that all unlabelled transitions have update zero. The set of configurations reachable from $(s, 0)$ can have size exponential in n , and its structure is unwieldy.

Right. (For Section 4.) The named state q belongs to Q_+ since there is a simple q -cycle with positive effect. There are six bounded chains of configurations at q . The disequality test $\neq 5$ bounds the counter values with residue 0 modulo 5, so $\{(q, 0)\}$ and $\{(q, 5)\}$ are bounded chains. The disequality test $\neq 30$ bounds the counter values with residue 3 modulo 5, so $\{(q, 3), (q, 8), (q, 13), (q, 18), (q, 23), (q, 28)\}$ is a bounded chain. The disequality test $\neq 15$ bounds the counter values with residue 2 modulo 5, so $\{(q, 2), (q, 7), (q, 12)\}$ is a bounded chain.

An ingredient of the NP upper bound for reachability in OCA *without* disequality tests [28] is establishing conditions under which a candidate run can be *lifted* to a run. We adapt the argument to OCA with disequality tests.

► **Lemma 4.2.** *Let \mathcal{A} be a strongly connected OCA with disequality tests. If src is unbounded in \mathcal{A} and trg is unbounded in \mathcal{A}^R , then there is a run from src to trg in \mathcal{A} ($\text{src} \xrightarrow{*} \text{trg}$) if and only if there is a candidate run from src to trg in \mathcal{A} ($\text{src} \xrightarrow{*}_{\mathbb{Z}} \text{trg}$).*

The hypothesis that \mathcal{A} is strongly connected is crucial. Indeed, if \mathcal{A} is strongly connected and $\text{src} = (s, v)$ is unbounded in \mathcal{A} , then there is a cycle of positive effect that, from src , can be applied infinitely often to reach (s, z) with z arbitrarily large. If \mathcal{A} is not strongly connected, it could be that the positive cycles that make src unbounded are in another SCC and that the set $\{z \mid (s, v) \xrightarrow{*} (s, z)\}$ is finite.

Let $Q_+ \subseteq Q$ be the set of states $q \in Q$ such that there exists a q -cycle γ with $\text{len}(\gamma) \leq |Q|$ and with $\text{eff}(\gamma) > 0$. For each $q \in Q_+$, let γ_q be such a q -cycle with minimal drop. **We fix this choice for the remainder of the paper.** We define

$$\text{Conf}_+ \stackrel{\text{def}}{=} \{(q, z) : q \in Q_+ \text{ and } z \geq \text{drop}(\gamma_q)\}.$$

► **Lemma 4.3.** *There is a polynomial-time algorithm to identify Q_+ and to choose cycles γ_q for all $q \in Q_+$. Moreover, membership in Conf_+ can be decided in polynomial time.*

The proof of Lemma 4.3 can be found in Appendix A.

► **Remark 4.4.** Our choice of Q_+ differs slightly from the definition found in [1]: we use short cycles ($\text{len}(\gamma) \leq |Q|$) rather than simple cycles. For simple cycles, the ability to compute, in polynomial time, the minimal drop of a positive-effect simple q -cycle (for each $q \in Q$) is not justified in [1]. In fact, in Appendix B, we prove that deciding, for a given OCA without tests \mathcal{A} and a given state q , whether there exists a positive-effect simple q -cycle in \mathcal{A} is an NP-complete problem. However, all constructions and arguments of [1] appear to be insensitive to the replacement of “simple cycles” by “short cycles”. As a result, we can still use polynomial-time algorithms for coverability and for unboundedness in OCA with disequality tests (1-VASS with disequality tests).

The set of all $(q, z) \in \text{Conf}_+$ can be partitioned into q -chains. For each $q \in Q_+$, let $\text{Conf}_+(q) = (\{q\} \times \mathbb{N}) \cap \text{Conf}_+$. A q -chain C is a maximal non-empty subset $C \subseteq \text{Conf}_+(q)$ such that, for every two distinct $c, c' \in C$, either c is reachable from c' by iterating γ_q , or vice versa. In other words, C is a non-empty minimal subset of $\text{Conf}_+(q)$ (with respect to set inclusion) such that, for all $c \in C$ and all $c' \in \text{Conf}$, if $c \xrightarrow{\gamma_q} c'$ or $c' \xrightarrow{\gamma_q} c$ then $c' \in C$.

A q -chain is *bounded* if it is a finite set, otherwise it is *unbounded*. Note that configurations in unbounded chains are all themselves unbounded, but configurations in bounded chains need not be bounded (they may be unbounded). Because the number of disequality guards that a cycle γ_q may encounter is small, so is the total number of bounded chains.

► **Lemma 4.5** (see [1, Remark 6]). *There are at most $2|Q|^2$ bounded chains.*

Given a chain C , the counter values z of every $(q, z) \in C$ have the same remainder modulo $\text{eff}(\gamma_q)$. Henceforth, a bounded q -chain can be described as $[\ell, u] \cap (r + \text{eff}(\gamma_q) \cdot \mathbb{N})$ where $[\ell, u]$ is an interval of nonnegative integers and $r + \text{eff}(\gamma_q) \cdot \mathbb{N}$ is an arithmetic progression with initial term r and difference $\text{eff}(\gamma_q)$. Since the OCA \mathcal{A} is encoded in binary, the values of l, u, r , and $|\gamma_q|$ may be exponential in the size of \mathcal{A} . See Figure 1 (Left) for an example.

5 Pessimistic Reachability

In this section, we exhibit a family run of runs, namely pessimistic runs, that are guaranteed to admit an NP certificate. This will already enable us to prove, in Section 6, that the reachability problem is in coNP^{NP} in the special case where the OCA is strongly connected.

Let \mathcal{A} be an OCA with disequality tests. We call a run of \mathcal{A} *pessimistic* if none of its configurations are in Conf_+ , except possibly the first one. Of course, some pessimistic runs may be exponentially long relative to the size of \mathcal{A} ; however, we provide a way to handle them. For $S \subseteq \text{Conf}_+$, we write $\text{Post}_*^*(S)$ for the set of configurations reachable from S using only pessimistic runs. In particular, $S \subseteq \text{Post}_*^*(S)$.

Consider the following decision problem:

PESSIMISTIC REACHABILITY

Input: An OCA \mathcal{A} with disequality tests, and two configurations `src` and `trg`.

Output: Is there a pessimistic run from `src` to `trg` in \mathcal{A} ?

Pessimistic runs turn out to be very handy, not least because we can adapt an existing “flow” technique [28] to decide pessimistic reachability.

► **Lemma 5.1.** *The pessimistic reachability problem is in NP.*

In a nutshell, the idea [28] is to guess how many times the run traverses each transition. The guessed numbers are subject to polynomial-time checkable balance and connectivity conditions, akin to, e.g., [46]. However, we cannot check whether the (possibly very long) run constructed from the flow violates disequality constraints, so the technique cannot be applied directly.

Our solution uses the pessimism of the run. Let $x \neq g$ be a guard on state q . We split the run in two: in the first part, all visits to q are above g ; then the run *jumps* the guard so that, in the second part, all visits are below g . This way, with at most $|Q|$ splits, we can reduce the problem to the case in which the run does not jump *any* disequality guard (always staying above or below each of them).

6 Reachability in Strongly Connected OCA

In this section, for pedagogical purposes, we study the particular case where the OCA is strongly connected. The case with multiple SCCs presented in Section 7 is more technical but relies on the same key idea.

► **Theorem 6.1.** *The reachability problem for strongly connected OCA with disequality tests belongs to the complexity class coNP^{NP} .*

We sketch the proof of Theorem 6.1 below. Throughout the section, we fix a strongly connected OCA with disequality tests \mathcal{A} and two configurations src and trg , and we are interested in whether $\text{src} \xrightarrow{*} \text{trg}$.

6.1 Ruling Out the Unbounded Case

By Lemma 4.1, given an instance of reachability, we can check in polynomial time whether src is unbounded in \mathcal{A} and trg is unbounded in \mathcal{A}^R . If both are true, then, by Lemma 4.2, it suffices to determine whether there exists a candidate run from src to trg . The existence of a candidate run can be decided in NP (e.g., using integer linear programming, see [12]). This case will not affect our complexity result because $\text{NP} \subseteq \text{coNP}^{\text{NP}}$. Thus, without loss of generality, we may assume that src is bounded in \mathcal{A} or trg is bounded in \mathcal{A}^R . Moreover, if trg is bounded in \mathcal{A}^R , we symmetrically work with \mathcal{A}^R instead of \mathcal{A} .

In the remainder of this section, we assume that src is bounded in \mathcal{A} .

6.2 Inductive Invariants in the Bounded Case

We will show that $\text{src} \not\xrightarrow{*} \text{trg}$ if and only if there exists a certificate of a particular shape witnessing this non-reachability. This certificate takes the form of an inductive invariant separating src and trg . The exact set of configurations comprising this inductive invariant is unwieldy, so we concentrate on its *core* instead. This set of core configurations admits a short representation, as follows.

We call an *arithmetic progression* on state $q \in Q$ a set of configurations $\{(q, v) \mid \ell \leq v \leq L \wedge \exists k \in \mathbb{N}, v = kp + s\}$ with $p, s, \ell, L \in \mathbb{N}$. An arithmetic progression can be specified by writing q and the numbers p, s, ℓ, L . A set of configurations *has a concise description* if it is a union of at most $2|Q|^2 + 1$ arithmetic progressions whose configurations have counter value bounded by $2|Q| \cdot \|\Delta\| \cdot \|\tau\|$. Such a set can be described in polynomial space.

The set of all configurations in bounded chains has a concise description. This also holds for the set R of all *reachable* configurations in bounded chains: indeed, if a configuration of a chain can be reached, the same is true for all configurations above in the same chain. Because src is bounded, unbounded chains cannot be reached, and this R is in fact the set of reachable configurations in all chains. (Observe that runs that reach configurations from R may well visit the complement of Conf_+ .)

► **Lemma 6.2.** *The set R of reachable configurations in Conf_+ has a concise description.*

Intuitively, R is our desired “core invariant”, and the desired invariant is the set $\text{Post}_-(R)$. However, when given a set I , it is not easy to check whether I is actually equal to R . Instead, the following theorem defines possible invariant cores by 3 conditions.

Conditions involving src and trg are self-explanatory. Set inclusion $\text{Post}(\text{Post}_*^*(I)) \subseteq \text{Post}_*^*(I)$ would express inductiveness (closure of the set under $\text{Post}(\cdot)$). However, verifying this condition is computationally expensive, and we replace it with a version that “focuses” on the core only, and thus has I rather than $\text{Post}_*^*(I)$ on the right-hand side.

► **Theorem 6.3.** *Suppose src is bounded in \mathcal{A} . Then $\text{src} \xrightarrow{*} \text{trg}$ if and only if there exists a set $I \subseteq \text{Conf}_+ \cup \{\text{src}\}$ with concise description such that:*

- (Cond1) $\text{src} \in I$,
- (Cond2) $\text{trg} \notin \text{Post}_*^*(I)$, and
- (Cond3) $\text{Post}(\text{Post}_*^*(I)) \cap \text{Conf}_+ \subseteq I$.

Proof. First, assume that there is such a set I . Because $\text{trg} \notin \text{Post}_*^*(I)$ by (Cond2), it suffices to prove that $\text{Post}_*^*(\text{src}) \subseteq \text{Post}_*^*(I)$. We proceed by induction on the length of the run from src to $c \in \text{Post}_*^*(\text{src})$. The base of induction is (Cond1). Assume that we have $d \in \text{Post}_*^*(I)$ and $d \rightarrow c$. If $c \notin \text{Conf}_+$ then we have a pessimistic run from I to c , so $c \in \text{Post}_*^*(I)$. If $c \in \text{Conf}_+$ then $c \in \text{Post}(\text{Post}_*^*(I)) \cap \text{Conf}_+$, hence $c \in I$ by (Cond3). For the other direction, assume that trg is not reachable from src . Let $I \stackrel{\text{def}}{=} R \cup \{\text{src}\}$; by Lemma 6.2, I has a concise description. (Cond1) and (Cond2) are trivially satisfied. Moreover, $\text{Post}(\text{Post}_*^*(I)) \cap \text{Conf}_+ \subseteq \text{Post}_*^*(\text{src}) \cap \text{Conf}_+ \subseteq I$, hence (Cond3) is satisfied. ◀

6.3 The Complexity of Reachability in Strongly Connected OCA

We now prove that reachability is in coNP^{NP} by, equivalently, proving that *non*-reachability is in NP^{NP} . Roughly speaking, a problem is in NP^{NP} whenever this problem is solvable in non-deterministic polynomial time by a Turing machine which has access to an oracle for some NP-complete problem. The oracle is a black box that may provide the answer to any problem in NP (and therefore to any problem in coNP).

As argued in Section 6.1, we assume with no loss of generality that src is bounded. By Theorem 6.3, we have $\text{src} \xrightarrow{*} \text{trg}$ if and only if there exists I satisfying the three conditions (Cond1), (Cond2), and (Cond3). Moreover, by the same theorem, I can be assumed to have a concise description. Thus, we can guess such a set I in non-deterministic polynomial time. It remains to prove that the verification that a set I satisfies the three conditions can be performed using an NP oracle. To this end, we prove that this verification is a coNP problem. Indeed, I does *not* satisfy the three conditions when:

- either $\text{src} \notin I$ (which can be checked efficiently),
- or $\text{trg} \in \text{Post}_*^*(I)$ (this is when there is a *small* configuration c such that $c \in I$ and $\text{trg} \in \text{Post}_*^*(c)$),
- or there are some *small* configurations c and d such that $c \in I$, $d \in \text{Post}_*^*(c)$, and some successor of d belongs to Conf_+ but not to I .

The adjective *small* should here be understood as “bounded by an exponential in the size of \mathcal{A} , src , and trg ”. In fact, it is fairly easy to obtain an exponential bound on configurations to consider. Thanks to Lemma 5.1, verification of both whether there is a $c \in I$ such that $\text{trg} \in \text{Post}_*^*(c)$ and whether there exist $c \in I$, $d \in \text{Post}_*^*(c)$, and $e \in \text{Post}(d)$ such that $e \notin I$ are in NP. Since membership in Conf_+ can be checked in polynomial time by Lemma 4.3, the entire third condition is also an NP condition. This completes the proof of Theorem 6.1.

7 Combining Strongly Connected Components

In this section, we extend the techniques from Section 6 to the general case in which the OCA is not assumed to be strongly connected. We fix an OCA \mathcal{A} with disequality tests and two configurations src and trg .

We first highlight why the techniques developed above do not apply to this general case. In Section 6, the hypothesis that \mathcal{A} is strongly connected was necessary for the application of Lemma 4.2. When \mathcal{A} is not strongly connected, knowing that src is unbounded is no longer satisfactory. Indeed, it no longer implies the existence of a positive cycle involving its state, as the positive cycle allowing us to pump up could be in another SCC. We need to be able to specify whether a configuration is unbounded *within its own SCC* or not.

7.1 Locally Bounded Configurations and Runs

Locally bounded configurations. Given a SCC S of \mathcal{A} , we denote by \mathcal{A}_S the automaton obtained when restricting \mathcal{A} to states and transitions within S . A configuration c is *locally bounded* if c is bounded in \mathcal{A}_S where S is the SCC of c . We denote by L the set of all locally bounded configurations (and by L^R in \mathcal{A}^R). Configurations that are not locally bounded are referred to as *locally unbounded*. We generalise the lifting technique from Lemma 4.2.

► **Lemma 7.1** (Lifting). *For all $c \notin L$ and $d \notin L^R$, we have $c \xrightarrow{*} d$ if and only if $c \xrightarrow{L^*} d$.*

Locally bounded runs. A run $c \xrightarrow{\pi} d$ is said to be *locally bounded* if all configurations visited by the run are locally bounded. We denote such a run by $c \xrightarrow{L} d$, and denote its existence by $c \xrightarrow{L^*} d$. Notice that a locally bounded run may go through several SCCs. Moreover, a run starting from a locally bounded configuration is not always locally bounded: once it goes to a new SCC, it may visit configurations that are not locally bounded. We define the locally bounded counterpart LPost_-^* of the pessimistic post-star operator: $d \in \text{LPost}_-^*(c)$ if there is a pessimistic and locally bounded run from c to d . We extend this definition to sets of configurations X in the usual way. Dually, we also define, for every set of configurations X , the set $\text{LPre}_+^*(X)$ as the same notion but in the reverse OCA \mathcal{A}^R . We extend Lemma 5.1 to these new operators.

► **Lemma 7.2.** *Given $c, d \in \text{Conf}$, deciding whether $d \in \text{LPost}_-^*(c)$ is in NP.*

Proof sketch. We split the run on its transitions between SCCs. We apply Lemma 5.1 on the portions remaining in one SCC. Since the run is pessimistic, we can bound all the counter values in it. The run is locally bounded when the first configuration visited in each SCC is locally bounded, which is checked using Lemma 4.1. ◀

7.2 Leaky Invariants

Unlike in the strongly-connected case, a single invariant construction is not sufficient for our needs. Indeed, if src is locally bounded but unbounded, then one could imagine the invariant technique from Theorem 6.3 applied to the SCC S of src , but then this invariant would not apply to other SCCs. For example, there could be runs that are locally bounded in the SCC S_{src} of src but not in the SCC S_{trg} of trg , making the invariant inapplicable. Instead, assuming that trg is locally bounded in \mathcal{A}^R , one may consider in the SCC of trg an invariant constructed in the reverse automaton \mathcal{A}^R . We therefore employ a pair of invariants, one for \mathcal{A} (the *forward invariant*) and another one for its reverse \mathcal{A}^R (the *backward invariant*). The two invariants will induce two sets of configurations that, in a negative instance of the reachability problem, separate the source and target.

The following lemma will allow us to avoid treating src and trg separately. The set Conf_+^R is defined as the counterpart of Conf_+ in \mathcal{A}^R .

17:12 Invariants for OCA with Disequality Tests

► **Lemma 7.3.** *We may assume that $\text{src} \in \text{Conf}_+ \cap L$ and $\text{trg} \in \text{Conf}_+^R \cap L^R$.*

We now define our notion of a leaky invariant. As in Section 6, we represent the invariants using *core* sets of configurations that can be succinctly described, denoted by I and J . Our invariants must be inductive in the following weak sense:

► **Condition 7.4.** *Let $I \subseteq \text{Conf}_+ \cap L$ and $J \subseteq \text{Conf}_+^R \cap L^R$ be sets of configurations. The pair (I, J) is **inductive** if*

$$\begin{aligned} \text{(Ind)} \quad & \text{Post}(\text{LPost}_*(I)) \cap \text{Conf}_+ \cap L \subseteq I \quad \text{and} \\ & \text{Pre}(\text{LPre}_+(J)) \cap \text{Conf}_+^R \cap L^R \subseteq J. \end{aligned}$$

Notice that I and J play symmetric roles in \mathcal{A} and \mathcal{A}^R . We now provide some intuition for the (forward) inductive condition for I . The set I only contains configurations from $\text{Conf}_+ \cap L$, because the set $\text{Conf}_+ \cap L$ has a regular structure thanks to bounded chains. The set I is, again, only the *core* of the invariant. The full invariant³ is $\text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I))$, but this set is not easily described (see Remark 7.8). This explains why we use the composition $\text{Post}(\text{LPost}_*(\cdot))$ instead of the single-step $\text{Post}(\cdot)$ operator traditionally used to define inductiveness.

► **Remark 7.5.** We refer to our invariants as *leaky*, because they are not inductive in the traditional sense. Indeed, our invariants are “focused” on locally bounded configurations, and can be escaped by transitions to locally unbounded configurations. This leak may, however, only happen with transitions going from one SCC to another.

► **Condition 7.6.** *Let $\mathcal{I}, \mathcal{J} \subseteq \text{Conf}$ be sets of configurations.*

*The pair $(\mathcal{I}, \mathcal{J})$ is a **separator** if, for all $c \in \mathcal{I}$ and $d \in \mathcal{J}$,*

$$\begin{aligned} \text{(Sep1)} \quad & c \not\rightarrow d; \text{ and} \\ \text{(Sep2)} \quad & \text{if } c \notin L \text{ and } d \notin L^R, \text{ then } c \not\stackrel{*}{\rightarrow}_{\mathbb{Z}} d. \end{aligned}$$

Firstly, **(Sep1)** will forbid \mathcal{I} and \mathcal{J} from being connected by a single step. Secondly, **(Sep2)** will forbid connection between \mathcal{I} and \mathcal{J} using the lifting technique of Lemma 7.1. This does not, in general, prevent the existence of runs from \mathcal{I} to \mathcal{J} ; it will do so, however, for our leaky invariants that combine Conditions 7.4 and 7.6.

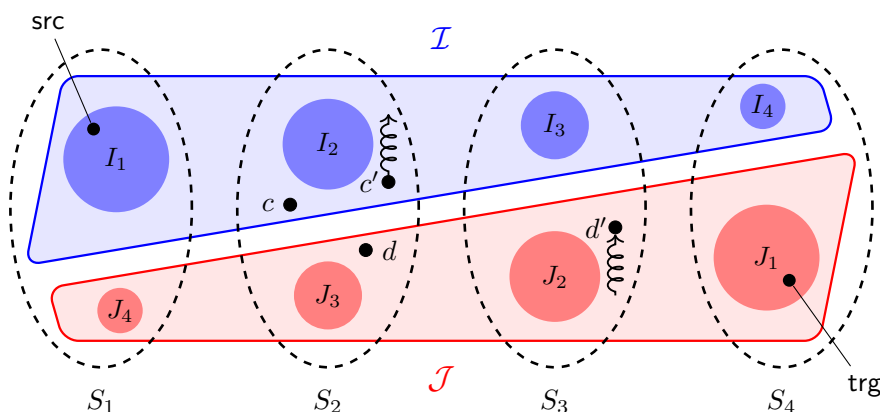
► **Definition 7.7.** *Let $I \subseteq \text{Conf}_+ \cap L$ and $J \subseteq \text{Conf}_+^R \cap L^R$. Consider the sets*

$$\begin{aligned} \mathcal{I} & := \text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I)) \text{ and} \\ \mathcal{J} & := \text{Pre}_+(J) \cup \text{Pre}(\text{Pre}_+(J)). \end{aligned}$$

We call the pair (I, J) a non-reachability witness for src and trg if (I, J) is inductive, $(\mathcal{I}, \mathcal{J})$ is a separator, $\text{src} \in \mathcal{I}$, and $\text{trg} \in \mathcal{J}$.

The pair of sets (I, J) forms the *core* of the invariant, namely I represents the *forward leaky invariant* and J represents the *backward leaky invariant*. In this case we also say that (I, J) *induces* the separator $(\mathcal{I}, \mathcal{J})$. A visualisation of a pair (I, J) and its induced separator $(\mathcal{I}, \mathcal{J})$ can be seen in Figure 2. A helpful intuition is that \mathcal{I} is approximately $\text{Post}_*(I)$ (and similarly \mathcal{J} is approximately $\text{Pre}_+(J)$). One additional step of $\text{Post}(\cdot)$ (and $\text{Pre}(\cdot)$, respectively) ensures that the “outer boundary” of this closure should also be included in the set.

³ Our invariant is $\text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I))$ but the operator that appears in Condition 7.4 is $\text{LPost}_*(\cdot)$. In Appendix C, we discuss the issues encountered if $\text{Post}_*(\cdot) \cup \text{Post}(\text{Post}_*(\cdot))$ was used in Condition 7.4.



■ **Figure 2** Core inductive sets and the separator they induce. The core of the forward leaky invariant is $I = I_1 \cup I_2 \cup I_3 \cup I_4$ (the blue circular sets) and the core of the backward leaky invariant is $J = J_1 \cup J_2 \cup J_3 \cup J_4$ (the red circular sets). The induced separator $(\mathcal{I}, \mathcal{J})$ is shown as blue and red rounded quadrilaterals containing the core sets. Notice that $\text{src} \in I$ and $\text{trg} \in J$. The upwards coiled arrow from c' represents that c' is locally unbounded in the SCC S_2 and the downwards coiled arrow from d' represents that d' is locally unbounded in the SCC S_3^R . Note also the separator conditions: Condition 7.6 (**Sep1**) means that configurations $c \in \mathcal{I}$ and $d \in \mathcal{J}$ cannot reach one another by one transition, so $c \not\rightarrow d$; Condition 7.6 (**Sep2**) means that unbounded configurations $c' \in \mathcal{I}$ and $d' \in \mathcal{J}$ cannot reach one another via a candidate run, so $c' \not\stackrel{*}{\underset{Z}{\rightarrow}} d'$.

► **Remark 7.8.** As in Section 6, our representation of invariants refers to their core only, i.e., the pair (I, J) . The example in Fig. 1 (left) demonstrates that the set $\text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I))$ does not always have a tractable description. The set of all possible sums of subsets has no convenient description, therefore we want it to be captured by $\text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I))$ only and not by I itself.

► **Theorem 7.9.** *In an OCA \mathcal{A} with disequality tests, trg is not reachable from src if and only if there exists a non-reachability witness. Moreover, in this case, there is always a non-reachability witness with a concise description.*

In Section 7.3, we define “perfect cores”, which we use in Section 7.4 to sketch a proof of Theorem 7.9 (details can be found in the full version).

7.3 Perfect Cores

Condition (**Ind**) on the core of leaky invariants captures a weak inductiveness property, which is central to our approach. We will now discuss two features of this condition that are used in the proof of Theorem 7.9.

Our conditions capture a specific invariant, which we now define. Consider the set

$$B \stackrel{\text{def}}{=} \{c \in \text{Conf} : \text{src} \xrightarrow[L]{*} c\}$$

In words, B contains all configurations reachable from src using locally bounded runs. In line with ideas from Section 6, we do not want to store B entirely, so we will restrict the core to configurations in bounded chains. We call *perfect core* the set $B \cap \text{Conf}_+$; the term *perfect* is motivated by the fact that Definition 7.7 aims to capture this set exactly. Similarly, let $B^R := \{c \in \text{Conf} : c \xrightarrow[L^R]{*} \text{trg}\}$. The *perfect core* in the reverse automaton is $B^R \cap \text{Conf}_+^R$.

The two features of **(Ind)** are summarised in the following two lemmas. We use the words “sound” and “complete” to characterise the relationship between Condition 7.4 (as part of Definition 7.7) and the perfect cores defined above. Completeness expresses that in *every instance* of non-reachability, the perfect cores defined above induce a non-reachability invariant. Conversely, soundness states that *every invariant* must contain all configurations from the perfect cores. (Thus, the perfect core are the smallest possible invariants.)

► **Lemma 7.10** (Soundness). *For all $I \subseteq \text{Conf}_+ \cap L$ and $J \subseteq \text{Conf}_+^R \cap L^R$ such that $\text{src} \in I$ and $\text{trg} \in J$, if (I, J) is inductive (Condition 7.4), then $B \cap \text{Conf}_+ \subseteq I$ and $B^R \cap \text{Conf}_+^R \subseteq J$.*

Proof sketch. Condition 7.4 for I gives $\text{Post}(\text{LPost}_*(I)) \cap \text{Conf}_+ \cap L \subseteq I$. Let $c \in B \cap \text{Conf}_+$ be a configuration of the perfect core. Thus, src reaches c by a locally bounded run. It is not always true that $c \in \text{Post}(\text{LPost}_*(\text{src}))$ because this run does not have to be pessimistic: it may observe configurations in Conf_+ . We prove by induction that all configurations in Conf_+ along the run are in I , using the property that $\text{Post}(\text{LPost}_*(I)) \cap \text{Conf}_+ \cap L \subseteq I$ once for each such configuration; this eventually proves that $c \in I$. The proof is analogous for J . ◀

► **Lemma 7.11** (Completeness). *If $I = B \cap \text{Conf}_+$ and $J = B^R \cap \text{Conf}_+^R$, then (I, J) is inductive (Condition 7.4).*

Proof sketch. We prove that $\text{Post}(\text{LPost}_*(B \cap \text{Conf}_+)) \cap \text{Conf}_+ \cap L \subseteq B \cap \text{Conf}_+$. Let $c \in \text{Conf}_+$ be locally bounded and belong to $\text{Post}(\text{LPost}_*(B \cap \text{Conf}_+))$. All configurations in $\text{LPost}_*(B \cap \text{Conf}_+)$ are in B by the definition of B , so c can be reached in one step from a configuration $d \in B$. By definition, d is reachable from src with a locally bounded run; since c is itself locally bounded, this is also true for c , and so $c \in B$. The case of J is similar. ◀

7.4 Non-reachability Witnesses and Their Complexity

► **Theorem 7.9.** *In an OCA \mathcal{A} with disequality tests, trg is not reachable from src if and only if there exists a non-reachability witness. Moreover, in this case, there is always a non-reachability witness with a concise description.*

Proof sketch. First, if $\text{src} \not\rightarrow^* \text{trg}$ then the perfect cores $I = B \cap \text{Conf}_+$ and $J = B^R \cap \text{Conf}_+^R$ form a non-reachability witness. Indeed, by Lemma 7.11, (I, J) is inductive. Moreover, the induced $\mathcal{I} = \text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I))$ and $\mathcal{J} = \text{Pre}_*(J) \cup \text{Pre}(\text{Pre}_*(J))$ form a separator. We have $\mathcal{I} \subseteq \text{Post}^*(\text{src})$ and $\mathcal{J} \subseteq \text{Pre}^*(\text{trg})$, proving Condition 7.6 (**Sep1**). If Condition 7.6 (**Sep2**) fails, Lemma 7.1 yields a contradiction. Moreover, I and J have a concise description thanks to bounded chains.

Conversely, suppose there is a non-reachability witness (I, J) . Assume for the sake of contradiction that $\text{src} \xrightarrow{*} \text{trg}$. By Lemma 7.10, since (I, J) is inductive, $B \cap \text{Conf}_+ \subseteq I$ and $B^R \cap \text{Conf}_+^R \subseteq J$. Consider a run from src to trg . It must leave $\mathcal{I} = \text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I))$ therefore it visits locally unbounded configurations. Let c be the first such configuration visited. Similarly, let d be the last visited configuration that is locally unbounded in \mathcal{A}^R . First, if c occurs before d , then Condition 7.6 (**Sep2**) is violated. Second, if c occurs after d , then there is an overlap in the runs from src to c and from d to trg . The overlap must be in $\mathcal{I} \cap \mathcal{J}$, leading to a violation of Condition 7.6 (**Sep1**). ◀

► **Theorem 7.12.** *The reachability problem for OCA with disequality tests belongs to the complexity class coNP^{NP} .*

Proof sketch. We use Theorem 7.9, deciding the existence of a non-reachability witness in NP^{NP} . Recall that NP^{NP} is introduced in Section 6.3. Let a pair (I, J) be given; we show that a violation of the conditions for being a non-reachability witness can be checked in NP.

- One can check in polynomial time whether $\text{src} \in I$ and $\text{trg} \in J$.
- Violation of Condition 7.4 (**Ind**) is an NP property. Indeed, this follows because membership in $\text{LPost}_*(\cdot)$ is in NP (by Lemma 7.2) and membership in Conf_+ and L is polynomial-time checkable (by Lemma 4.3 and Lemma 4.1, respectively).
- Violation of Condition 7.6 (**Sep1**) is in NP, because $\mathcal{I} := \text{Post}_*(I) \cup \text{Post}(\text{Post}_*(I))$, $\mathcal{J} := \text{Pre}_*(J) \cup \text{Pre}(\text{Pre}_*(J))$, and membership of given configurations in the pessimistic post-star (optimistic pre-star, respectively) is in NP by Lemma 5.1. This assumes that we have an exponential bound on relevant configurations.
- To check violation of Condition 7.6 (**Sep2**) in NP, we again use Lemma 4.1 for L , as well as the fact that the existence of a candidate run is in NP (by integer programming). ◀

7.5 Adding Equality Tests

The previous techniques have been developed for OCA with disequality tests only. In particular, the lifting argument of Lemma 7.1 does not hold in the presence of equality tests: candidate runs that visit a state with an equality test cannot be lifted to greater counter values. However, at the cost of increasing the complexity, one can handle equality tests.

Complexity class $\text{P}^{\text{NP}^{\text{NP}}}$ consists of decision problems solvable in polynomial time with access to an NP^{NP} oracle (which can solve NP^{NP} problems in one step).

► **Corollary 7.13.** *The reachability problem for OCA with equality and disequality tests belongs to the complexity class $\text{P}^{\text{NP}^{\text{NP}}}$.*

Proof. Let \mathcal{A} be such an OCA with tests, and src and trg two configurations. Denote by $\text{Conf}_=$ the set of valid configurations at states with equality tests; $|\text{Conf}_=|$ does not exceed the number of states in \mathcal{A} . Consider the OCA with disequality tests \mathcal{A}' that is obtained by deleting all states with equality tests (and incident transitions) from \mathcal{A} . By Theorem 7.12, with an NP^{NP} oracle we can build a graph with vertex set $\text{Conf}_= \cup \{\text{src}, \text{trg}\}$ and edge set $\{(c, d) \mid c \xrightarrow{*} d \text{ in } \mathcal{A}'\}$. Depth-first search in this graph for a path from src to trg takes polynomial time. ◀

8 Conclusions

We have looked at the reachability problem for one-counter automata with equality and disequality tests. We have proposed the idea of local inductive invariants and combined them with the notion of unboundedness within an SCC. Our construction circumvents the lack of computationally tractable descriptions: indeed, in the subset sum example (Fig. 1 (left) and Remark 7.8) the reachability set has exponential size, depending on a_1, \dots, a_n . There is no obvious means of compression available, and guessing/storing a traditional invariant is prohibitively expensive even for moderate n .

An outstanding theoretical question is characterisation of complexity of reachability in OCA with disequality tests. We have placed the problem in coNP^{NP} and, in the presence of equality tests, in $\text{P}^{\text{NP}^{\text{NP}}}$. Both problems have already been known to be NP-hard. Are they NP-complete or coNP -hard too? We also leave it open whether our technique can be extended to other systems and settings, e.g., to parameter synthesis questions (see, e.g., [26, 35, 43]).

In a more practical direction, while the general invariant-based effective procedure for (non-)reachability in vector addition systems [36] has not, to the best of our knowledge, been implemented, our work identifies these potentially practical ways to reduce the search space for invariants in VASS. The idea of restricting invariant sets to just a small “core” (in our case: a union of arithmetic progressions), combined with the compositionality of invariants, can help to direct an exploration of the search space, or assist a learning algorithm.

References

- 1 Shaull Almagor, Nathann Cohen, Guillermo A. Pérez, Mahsa Shirmohammadi, and James Worrell. Coverability in 1-VASS with disequality tests. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPICs*, pages 38:1–38:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 2 Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 599–610, 2011.
- 3 Nicolas Amat, Silvano Dal-Zilio, and Thomas Hujsa. Property directed reachability for generalized Petri nets. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 505–523. Springer, 2022.
- 4 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. Context-bounded verification of context-free specifications. *Proc. ACM Program. Lang.*, 7(POPL):2141–2170, 2023.
- 5 Michael Blondin, Matthias Englert, Alain Finkel, Stefan Göller, Christoph Haase, Ranko Lazic, Pierre McKenzie, and Patrick Totzke. The reachability problem for two-dimensional vector addition systems with states. *J. ACM*, 68(5):34:1–34:43, 2021.
- 6 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 480–496. Springer, 2016.
- 7 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. The logical view on continuous Petri nets. *ACM Trans. Comput. Log.*, 18(3):24:1–24:28, 2017.
- 8 Michael Blondin, Christoph Haase, and Philip Offtermatt. Directed reachability for infinite-state systems. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021.
- 9 Michael Blondin, Tim Leys, Filip Mazowiecki, Philip Offtermatt, and Guillermo A. Pérez. Continuous one-counter automata. *ACM Trans. Comput. Log.*, 24(1):3:1–3:31, 2023.
- 10 Stanislav Böhm, Stefan Göller, and Petr Jancar. Equivalence of deterministic one-counter automata is NL-complete. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 131–140. ACM, 2013.
- 11 Benedikt Bollig, Karin Quaas, and Arnaud Sangnier. The complexity of flat freeze LTL. *Log. Methods Comput. Sci.*, 15(3), 2019.

- 12 Itshak Borosh and Leon Bruce Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976.
- 13 Aaron R. Bradley. SAT-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.
- 14 Aaron R. Bradley. Understanding IC3. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2012.
- 15 Daniel Bundala and Joël Ouaknine. On parametric timed automata and one-counter machines. *Inf. Comput.*, 253:272–303, 2017.
- 16 Dmitry Chistikov, Wojciech Czerwinski, Piotr Hofman, Michal Pilipczuk, and Michael Wehar. Shortest paths in one-counter systems. *Log. Methods Comput. Sci.*, 15(1), 2019.
- 17 Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
- 18 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977.
- 19 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. *J. ACM*, 68(1):7:1–7:28, 2021.
- 20 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021.
- 21 Stéphane Demri and Arnaud Sangnier. When model-checking freeze LTL over counter machines becomes decidable. In C.-H. Luke Ong, editor, *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6014 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2010.
- 22 Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Nksic. An SMT-based approach to coverability analysis. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 603–619. Springer, 2014.
- 23 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015.
- 24 Alain Finkel, Serge Haddad, and Igor Khmelnitsky. Minimal coverability tree construction made complete and efficient. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 237–256. Springer, 2020.
- 25 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. doi:10.2140/pjm.1966.16.285.

- 26 Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model checking succinct and parametric one-counter automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2010.
- 27 Stefan Göller and Markus Lohrey. Branching-time model checking of one-counter processes and timed automata. *SIAM J. Comput.*, 42(3):884–923, 2013.
- 28 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009.
- 29 Michel Hack. *Decidability questions for Petri nets*. PhD thesis, MIT, 1975. URL: <http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-161.pdf>.
- 30 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 743–759. Springer, 2011.
- 31 Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2012.
- 32 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2013. doi:10.1007/978-3-642-39799-8_2.
- 33 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Efficient coverability analysis by proof minimization. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 500–515. Springer, 2012.
- 34 Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2005.
- 35 Antonia Lechner, Richard Mayr, Joël Ouaknine, Amaury Pouly, and James Worrell. Model checking flat freeze LTL on one-counter automata. *Log. Methods Comput. Sci.*, 14(4), 2018.
- 36 Jérôme Leroux. The general vector addition system reachability problem by Presburger inductive invariants. *Log. Methods Comput. Sci.*, 6(3), 2010.
- 37 Jérôme Leroux. Distance between mutually reachable Petri net configurations. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPICs*, pages 47:1–47:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 38 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021.

- 39 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019.
- 40 Xie Li, Taolue Chen, Zhilin Wu, and Mingji Xia. Computing linear arithmetic representation of reachability relation of one-counter automata. In Jun Pang and Lijun Zhang, editors, *Dependable Software Engineering. Theories, Tools, and Applications - 6th International Symposium, SETTA 2020, Guangzhou, China, November 24-27, 2020, Proceedings*, volume 12153 of *Lecture Notes in Computer Science*, pages 89–107. Springer, 2020.
- 41 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
- 42 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., USA, 1967.
- 43 Guillermo A. Pérez and Ritam Raha. Revisiting parameter synthesis for one-counter automata. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 33:1–33:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 44 Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *J. Comput. Syst. Sci.*, 32(1):105–135, 1986. doi:10.1016/0022-0000(86)90006-1.
- 45 Sylvain Schmitz. The complexity of reachability in vector addition systems. *SIGLOG News*, 3(1):4–21, 2016.
- 46 Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.
- 47 Alistair Stewart, Kousha Etessami, and Mihalis Yannakakis. Upper bounds for Newton’s method on monotone polynomial systems, and P-time model checking of probabilistic one-counter automata. *J. ACM*, 62(4):30:1–30:33, 2015.
- 48 Leslie G. Valiant and Mike Paterson. Deterministic one-counter automata. *J. Comput. Syst. Sci.*, 10(3):340–350, 1975.

A Proof of Lemma 4.3

The proof will use 1-dimensional vector addition systems with states (1-VASS). These are one-counter automata (as defined in this paper) without any tests:

- Syntactically, a 1-VASS is a pair (Q, T) , where Q is the set of states and $T \subseteq Q \times \mathbb{Z} \times Q$ is the set of transitions.
- The semantics is the same as that of an OCA with tests (Q, T, true) , where the map **true** assigns true to all states in Q .

We will consider an auxiliary problem which takes as input a 1-VASS (Q, T) , a state $q \in Q$, and a natural number d (encoded in binary); the problem asks whether there is a positive-effect q -cycle γ of length at most $|Q|$ such that $\text{drop}(\gamma) \leq d$. We show that this problem can be solved in polynomial time, given that coverability in 1-VASS can be decided in polynomial time. Coverability is placed in P by a standard reduction from coverability to unboundedness (see, e.g., [1, Lemma 1]) and a polynomial-time algorithm for unboundedness by Rosier and Yen [44, Theorem 3.4]. A stronger result from [1] is that coverability in 1-VASS is in fact in $\text{NC}^2 \subseteq \text{P}$.

Reduction of the auxiliary problem to coverability. Let $n = |Q|$. We can construct an instance of coverability as follows. Consider the unfolding (Q', T') where $Q' = \{q^{(i)} : q \in Q \text{ and } i \in [0, n]\}$ and $T' = \{(p^{(i-1)}, a, q^{(i)}) : (p, a, q) \in T \text{ and } i \in [1, n]\}$. Observe that there exists a positive-effect q -cycle whose length is at most $|Q|$ and with a drop bounded by d in (Q, T) if and only if $(q^{(i)}, d+1)$ can be covered from $(q^{(i)}, d)$ in (Q', T') for some $i \in [1, n]$. Moreover, in that case, such a cycle is obtained directly from a path in the unfolded 1-VASS that witnesses coverability.

Polynomial-time algorithms for minimum drop and membership in Conf_+ . We complete the proof of Lemma 4.3:

- First, observe that the minimum drop can be computed by a binary search for d . Let $m = \max\{|a| : (p, a, q) \in T\}$. By starting from an upper bound of $n(m+1)$, d can be computed using a polynomial number (at most $\lceil \log(n(m+1)) \rceil$) of coverability queries.
- Second, to decide membership of a configuration (q, v) in Conf_+ , it suffices to check that $q \in Q_+$, to compute $\text{drop}(\gamma_q)$, and to check that $v \geq \text{drop}(\gamma_q)$. ◀

B Finding Positive-Effect Simple Cycles is NP-hard

► **Proposition B.1.** *Deciding, for a given OCA without tests \mathcal{A} and a given state q , whether there exists a positive-effect simple q -cycle in \mathcal{A} is an NP-complete problem.*

Proof. Membership in NP is obtained by using the q -cycle itself as a certificate. To prove NP-hardness, we provide a reduction from the Hamiltonian path problem. Let $G = (V, E)$ be a directed graph and let $s, t \in V$ be two distinct vertices. A path from s to t is *Hamiltonian* if it is simple and visits every vertex in the graph. The Hamiltonian path problem takes as input a directed graph $G = (V, E)$ and two vertices $s, t \in V$ and asks whether there is a Hamiltonian path from s to t in G .

For the remainder of this proof, we fix an instance of this problem formed by $G = (V, E)$ and $s, t \in V$. Let $n = |V|$. We will now construct an OCA without tests (a 1-VASS) $\mathcal{A} = (Q, \Delta)$. Define $Q := V \cup \{q\}$, where $q \notin V$ is a new state, and

$$\Delta := \{(u, 1, v) : (u, v) \in E\} \cup \{(q, 0, s), (t, -(n-2), q)\}.$$

The construction of \mathcal{A} takes polynomial time. We claim that there exists a Hamiltonian path from s to t in G if and only if there exists a positive-effect simple q -cycle in \mathcal{A} .

Suppose there exists a Hamiltonian path π from s to t in G . Since π visits every vertex in G , we have $\text{len}(\pi) = n-1$. Consider the path σ in \mathcal{A} that is obtained from π by replacing each edge $(u, v) \in E$ with the corresponding transition $(u, 1, v) \in \Delta$ as well as prepending the transition $(q, 0, s)$ and appending the transition $(t, -(n-2), q)$. Given that π is a simple path in G , we know that σ is a simple q -cycle in \mathcal{A} . Furthermore, given that $\text{len}(\pi) = n-1$, we know that $\text{eff}(\sigma) = 0 + n-1 - (n-2) = 1$, so σ has positive effect.

Conversely, suppose there exists a positive-effect simple q -cycle σ in \mathcal{A} . This σ must begin with $(q, 0, s)$, the only outgoing transition from q , and end with $(t, -(n-2), q)$, the only transition leading back to q . Let $\sigma = (q, 0, s) \sigma' (t, -(n-2), q)$ for some σ' . Given that $\text{eff}(\sigma) \geq 1$ and all other transitions in \mathcal{A} have effect 1, we know that $\text{len}(\sigma') \geq n-1$. Since the cycle σ is simple and $|Q \setminus \{q, s, t\}| = n-2$, we conclude that σ' visits each of these $n-2$ states exactly once. So the path π obtained from σ' by replacing each transition $(u, 1, v) \in \Delta$ with the corresponding edge $(u, v) \in E$ is a Hamiltonian path from s to t in G . ◀

C Discussion of the Choice of Operators

In Condition 7.4, we made the choice of using operator $L\text{Post}_*^*(\cdot)$, and not $\text{Post}_*^*(\cdot) \cup \text{Post}(\text{Post}_*^*(\cdot))$ as in Section 6. Indeed, if we had used $\text{Post}_*^*(\cdot) \cup \text{Post}(\text{Post}_*^*(\cdot))$ instead, then in order to obtain completeness (Lemma 7.11), one would have to change the perfect core. We want the perfect core to be contained in $L \cap \text{Conf}_+$ so that it has a short representation; the natural candidate would be to take $\text{Post}_*^*(\text{src}) \cap L \cap \text{Conf}_+$ (and symmetrically in \mathcal{A}^R). This perfect core would satisfy the inductive property; however, this choice would break soundness (Lemma 7.10). Indeed, this invariant could contain a locally bounded configuration c that is reached from src using a run that visits many locally unbounded configurations in $\text{Conf} \setminus \text{Conf}_+$ before coming back to L . In this case, it could be that c is not captured by the inductive property, so one could find an inductive invariant I that does not contain c .

Weighted Basic Parallel Processes and Combinatorial Enumeration

Lorenzo Clemente   

Department of Mathematics, Mechanics, and Computer Science, University of Warsaw, Poland

Abstract

We study *weighted basic parallel processes* (WBPP), a nonlinear recursive generalisation of weighted finite automata inspired from process algebra and Petri net theory. Our main result is an algorithm of 2-EXPSpace complexity for the WBPP equivalence problem. While (unweighted) BPP language equivalence is undecidable, we can use this algorithm to decide multiplicity equivalence of BPP and language equivalence of unambiguous BPP, with the same complexity. These are long-standing open problems for the related model of weighted context-free grammars.

Our second contribution is a connection between WBPP, power series solutions of systems of polynomial differential equations, and combinatorial enumeration. To this end we consider *constructible differentially finite* power series (CDF), a class of multivariate differentially algebraic series introduced by Bergeron and Reutenauer in order to provide a combinatorial interpretation to differential equations. CDF series generalise rational, algebraic, and a large class of D-finite (holonomic) series, for which no complexity upper bound for equivalence was known. We show that CDF series correspond to *commutative* WBPP series. As a consequence of our result on WBPP and commutativity, we show that equivalence of CDF power series can be decided with 2-EXPTIME complexity.

In order to showcase the CDF equivalence algorithm, we show that CDF power series naturally arise from combinatorial enumeration, namely as the exponential generating series of *constructible species of structures*. Examples of such species include sequences, binary trees, ordered trees, Cayley trees, set partitions, series-parallel graphs, and many others. As a consequence of this connection, we obtain an algorithm to decide multiplicity equivalence of constructible species, decidability of which was not known before.

The complexity analysis is based on effective bounds from algebraic geometry, namely on the length of chains of polynomial ideals constructed by repeated application of finitely many, not necessarily commuting derivations of a multivariate polynomial ring. This is obtained by generalising a result of Novikov and Yakovenko in the case of a single derivation, which is noteworthy since generic bounds on ideal chains are non-primitive recursive in general. On the way, we develop the theory of WBPP series and CDF power series, exposing several of their appealing properties.

2012 ACM Subject Classification Theory of computation → Quantitative automata; Theory of computation → Concurrency; Mathematics of computing → Combinatorics

Keywords and phrases weighted automata, combinatorial enumeration, shuffle, algebraic differential equations, process algebra, basic parallel processes, species of structures

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.18

Related Version *Full Version*: <https://arxiv.org/abs/2407.03638> [24]

Funding Supported by the ERC grant INFSYS, agreement no. 950398.

Acknowledgements We warmly thank Mikołaj Bojańczyk, Arka Ghosh, Filip Mazowiecki, and Paweł Parys for their comments and support at the various stages of this work.



© Lorenzo Clemente;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 18; pp. 18:1–18:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We study the equivalence problem for a class of finitely presented seriesoriginating in weighted automata, process algebra, and combinatorics. We begin with some background.

1.1 Motivation and context

Weighted automata. Classical models of computation arising in the seminal work of Turing from the 1930's [75] have a Boolean-valued semantics (“is an input accepted?”) and naturally recognise languages of finite words $L \subseteq \Sigma^*$. In the 1950's a finite-memory restriction was imposed on Turing machines, leading to an elegant and robust theory of *finite automata* [64], with fruitful connections with logic [18, 28, 74] and regular expressions [47]. *Weighted finite automata* over a field \mathbb{F} (WFA) [68] were introduced in the 1960's by Schützenberger as a generalisation of finite automata to a quantitative *series* semantics $\Sigma^* \rightarrow \mathbb{F}$ (“in how many ways can an input be accepted?”). This has been followed by the development of a rich theory of weighted automata and logics [27]. While the general theory can be developed over arbitrary semirings, the methods that we develop in this work are specific to fields, and in particular for effectiveness we assume the field of rational numbers $\mathbb{F} = \mathbb{Q}$.

The central algorithmic question that we study is the *equivalence problem*: Given two (finitely presented) series $f, g : \Sigma^* \rightarrow \mathbb{Q}$, is it the case that $f = g$? (In algorithmic group theory this is known as the *word problem*.) A mathematical characterisation of equivalence yields a deeper understanding of the interplay between syntax and semantics, and a decidability result means that this understanding is even encoded as an algorithm. Equivalence of weighted models generalises *multiplicity equivalence* of their unweighted counterparts (“do two models accept each input in the same number of ways?”), in turn generalising language equivalence of *unambiguous models* (each input is accepted with multiplicity 0 or 1). Since equivalence $f = g$ reduces to *zeroness* $f - g = 0$, from now on we will focus on the latter problem.

While nonemptiness of WFA is undecidable [58, Theorem 21] (later reported in Paz' book [60, Theorem 6.17]), zeroness is decidable, even in polynomial time [68] – a fact often rediscovered, e.g., [73, 76]. This has motivated the search for generalisations of WFA with decidable zeroness. However, many of them are either known to be undecidable (e.g., *weighted Petri nets* [42, Theorem 3]), or beyond the reach of current techniques (e.g., *weighted one counter automata*, *weighted context-free grammars*, and *weighted Parikh automata*). One notable exception is *polynomial weighted automata*, although zeroness has very high complexity (Ackermann-complete) [3]. In the restricted case of a unary input alphabet, decidability and complexity results can be obtained with algebraic [1] and D-finite techniques [15].

Process algebra. On a parallel line of research, the process algebra community has developed a variety of formalisms modelling different aspects of concurrency and nondeterminism. We focus on *basic parallel processes* (BPP) [21], a subset of the *calculus of communicating systems* without sequential composition [55]. BPP are also known as *communication-free Petri nets* (every transition consumes exactly one token) and *commutative context-free grammars* (nonterminals in sentential forms are allowed to commute with each other). While language equality for BPP is undecidable [40, 41], bisimulation equivalence is decidable [22] (even PSPACE-complete [70, 43]). *Multiplicity equivalence*, finer than language equality and incomparable with bisimulation, does not seem to have been studied for BPP.

Combinatorial enumeration and power series. We shall make a connection between BPP, power series, and combinatorial enumeration. For this purpose, let us recall that the study of multivariate power series in commuting variables has a long tradition at the border

of combinatorics, algebra, and analysis of algorithms [72, 30]. We focus on *constructible differentially finite* power series (CDF) [6, 7], a class of differentially algebraic power series arising in combinatorial enumeration [49, 4]. Their study was initiated in the *univariate* context in [6], later extended to multivariate [7]. They generalise rational and algebraic power series, and are incomparable with D-finite power series [71, 51]. For instance, the exponential generating series $\sum_{n \in \mathbb{N}} n^{n-1} \cdot x^n/n!$ of Cayley trees is CDF, but it is neither algebraic/D-finite [16, Theorem 1], nor polynomial recursive [20, Theorem 5.3].

The theory of *combinatorial species* [45, 5] is a formalism describing families of finite structures. It arises as a categorification of power series, by noticing how primitives used to build structures – sum, combinatorial product, composition, differentiation, resolution of implicit equations – are in a one-to-one correspondence with corresponding primitives on series. Using these primitives, a rich class of *constructible species* can be defined [61]. For instance the species $C[\mathcal{X}]$ of Cayley trees (rooted unordered trees) is constructible since it satisfies $C[\mathcal{X}] = \mathcal{X} \cdot \text{SET}[C[\mathcal{X}]]$. Two species are *multiplicity equivalent* (*equipotent* [61]) if for every $n \in \mathbb{N}$ they have the same number of structures of size n . Multiplicity equivalence of species has not been studied from an algorithmic point of view.

1.2 Contributions

We study *weighted basic parallel processes* over the field of rational numbers (WBPP), a weighted extension of BPP generalising WFA. The following is our main contribution.

► **Theorem 1.** *The zeroness problem for WBPP is in 2-EXPSpace.*

This elementary complexity should be contrasted with Ackermann-hardness of zeroness of polynomial automata [3], another incomparable extension of WFA. Since WBPP can model the multiplicity semantics of BPP, as an application we get the following corollary.

► **Corollary 2.** *Multiplicity equivalence of BPP and language equivalence of unambiguous BPP are decidable in 2-EXPSpace.*

On a technical level, Theorem 1 is obtained by extending an ideal construction and complexity analysis from [59] from the case of a single polynomial derivation to the case of a finite set of not necessarily commuting polynomial derivations. It is remarkable that such ideal chains have elementary length, since generic bounds without further structural restrictions are only general recursive [69]. This shows that the BPP semantics is adequately captured by differential algebra. These results are presented in § 2. In § 3 we observe that *commutative* WBPP series coincide with CDF power series, thus establishing a novel connection between automata theory, polynomial differential equations, and combinatorics. This allows us to obtain a zeroness algorithm for CDF, which is our second main contribution.

► **Theorem 3.** *The zeroness problem for multivariate CDF power series is in 2-EXPTIME.*

The complexity improvement from 2-EXPSpace to 2-EXPTIME is due to commutativity. In the special *univariate* case, decidability was observed in [6] with no complexity analysis, while [7] did not discuss decidability in the multivariate case. In § 4 we apply Theorem 3 to multiplicity equivalence of a class of constructible species. This follows from the observation that their exponential generating series (EGS) are effectively CDF, proved by an inductive argument based on the closure properties of CDF series. For instance, the EGS of Cayley trees satisfies $C = x \cdot e^C$; by introducing auxiliary series $D := e^C$, $E := (1 - C)^{-1}$ and by differentiating we obtain CDF equations $\partial_x C = D \cdot E$, $\partial_x D = D^2 \cdot E$, $\partial_x E = D \cdot E^3$.

► **Theorem 4.** *Multiplicity equivalence of strongly constructible species is decidable.*

1.3 Related works

There have recently been many decidability results for models incomparable with WBPP, such as multiplicity equivalence of *boundedly-ambiguous* Petri nets [26, Theorem 3]; zeroness for weighted one-counter automata with *deterministic counter updates* [52]; zeroness of *P-finite automata*, a model intermediate between WFA and polynomial automata (even in PTIME [19]); and zeroness of *orbit-finite weighted automata* in sets with atoms [9].

Regarding power series, there is a rich literature on dynamical systems satisfying differential equations in the CDF format, that is polynomial ordinary differential equations (ODE; cf. [62] and references therein). While many algorithms have been proposed for their analysis (e.g., invariant checking [63]), the complexity of the zeroness problem has not been addressed before. A decision procedure for zeroness of multivariate CDF can be obtained from first principles as a consequence of Hilbert's *finite basis theorem* [25, Theorem 4, §5, Ch. 2]. For instance, decidability follows from the algorithm of [12] computing pre- and post-conditions for restricted systems of partial differential equations (covering CDF), and also from the *Rosenfeld-Gröbner algorithm* [17], which can be used to test membership in the radical differential ideal generated by the system of CDF equations. In both cases, no complexity-theoretic analysis is provided and only decidability can be deduced. In the univariate CDF case, decidability can also be deduced from [10, 11]. Univariate CDF also arise in the coalgebraic treatment of stream equations with the shuffle product [14, 13], where an equivalence algorithm based on Hilbert's theorem is provided.

The work [34] studies *Noetherian functions*, which are analytic functions satisfying CDF equations. In fact, Noetherian functions which are analytic around the origin coincide with multivariate CDF power series. The work [77] discusses a subclass of Noetherian functions obtained by iteratively applying certain extensions to the ring of multivariate polynomials and presents a zeroness algorithm running in doubly exponential time. Theorem 3 is more general since it applies to all Noetherian power series.

In the context of the realisability problem in control theory, Fliess has introduced the class of *differentially producible series* [32] (cf. also the exposition of Reutenauer [66]), a generalisation of WBPP series where the state and transitions are given by arbitrary power series (instead of polynomials). Such series are characterised by a notion of *finite Lie rank* and it is shown that differentially producible series of minimal Lie rank exist and are unique. Such series are not finitely presented and thus algorithmic problems, such as equivalence, cannot even be formulated.

Full proofs can be found in the technical report [24].

Preliminaries. Let $\Sigma = \{a_1, \dots, a_d\}$ be a finite alphabet. We denote by Σ^* the set of *finite words* over Σ , a monoid under the operation of concatenation, with neutral element the empty word ε . The *Parikh image* of a word $w \in \Sigma^*$ is $\#(w) := (\#(w)_{a_1}, \dots, \#(w)_{a_d}) \in \mathbb{N}^d$, where $\#(w)_{a_j}$ is the number of occurrences of a_j in w . Let \mathbb{Q} be the field of rational numbers. Most results in the paper hold for any field, however for computability considerations we restrict our presentation to \mathbb{Q} . For a tuple of commuting indeterminates $x = (x_1, \dots, x_k)$, denote by $\mathbb{Q}[x]$ the ring of *multivariate polynomials* ($\mathbb{Q}[k]$ when the name of variables does not matter) and by $\mathbb{Q}(x)$ its fraction field of *rational functions* (that is, ratios of polynomials $p(x)/q(x)$). The *one norm* $|z|_1$ of a vector $z = (z_1, \dots, z_k) \in \mathbb{Q}^k$ is $|z_1| + \dots + |z_k|$, and the *infinity norm* is $|z|_\infty = \max_{1 \leq i \leq k} |z_i|$. Similarly, the *infinity norm* (also called *height*) of a polynomial $p \in \mathbb{Q}[k]$, written $|p|_\infty$, is the maximal absolute value of any of its coefficients.

A *derivation* of a ring R is a linear function $\delta : R \rightarrow R$ satisfying

$$\delta(a \cdot b) = \delta(a) \cdot b + a \cdot \delta(b). \quad (\text{Leibniz rule})$$

A derivation δ of a polynomial ring $R[x]$ is uniquely defined once we fix $\delta(x) \in R[x]$. For instance, $\partial_x : R[x] \rightarrow R[x]$ is the unique derivation δ of the polynomial ring s.t. $\delta(x) = 1$. Other technical notions will be recalled when necessary. For a general introduction to algebraic geometry we refer to [25].

2 Weighted extension of basic parallel processes

2.1 Basic parallel processes

In this section we recall the notion of basic parallel process (BPP) together with its language semantics. Let $\{X_1, X_2, \dots\}$ be a countable set of *nonterminals* (process variables) and let Σ be a finite alphabet of *terminals* (actions). A *BPP expression* is generated by the following abstract grammar (cf. [29, Sec. 5]): $E, F ::= \perp \mid X_i \mid a.E \mid E + F \mid E \parallel F$. Intuitively, \perp is a constant representing the *terminated process*, $a.E$ (*action prefix*), is the process that performs action a and becomes E , $E + F$ (*choice*) is the process that behaves like E or F , and $E \parallel F$ (*merge*) is the *parallel* execution of E and F . We say that an expression E is *guarded* if every occurrence of a nonterminal X_i is under the scope of an action prefix. A *BPP* consists of a distinguished *starting nonterminal* X_1 and rules

$$X_1 \rightarrow E_1 \quad \dots \quad X_k \rightarrow E_k, \quad (1)$$

where the r.h.s. expressions E_1, \dots, E_k are guarded and contain only nonterminals X_1, \dots, X_k .

$$\begin{array}{c} a.E \xrightarrow{a} E \\ \frac{E_i \xrightarrow{a} E'}{X_i \xrightarrow{a} E'} \\ \frac{E \xrightarrow{a} E'}{E \parallel F \xrightarrow{a} E' \parallel F} \end{array} \quad \frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'} \quad \frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'} \quad \frac{F \xrightarrow{a} F'}{E \parallel F \xrightarrow{a} E \parallel F'}$$

Figure 1 BPP transition rules.

A BPP induces an infinite labelled transition system where states are expressions and the labelled transition relations \xrightarrow{a} are the least family of relations closed under the rules from Fig. 1. The transition relation is extended naturally to words \xrightarrow{w} , $w \in \Sigma^*$. An expression E is *final* if there are no a, E' s.t. $E \xrightarrow{a} E'$ (e.g., $\perp \parallel \perp$); it *accepts* a word $w \in \Sigma^*$ if there is a final expression F s.t. $E \xrightarrow{w} F$. The *language* $L(E)$ recognised by an expression E is the set of words it accepts, and the language of a BPP is $L(X_1)$.

An expression E is in (*full*) *standard form* if it is a sum of products $a_1.\alpha_1 + \dots + a_n.\alpha_n$, where each α_i is a merge of nonterminals; a BPP (1) is in standard form if every E_1, \dots, E_k is in standard form. The standard form for BPP is analogous to the Greibach normal form for context-free grammars [35]. Every BPP can be effectively transformed to one in standard form preserving bisimilarity [21, Proposition 2.31], and thus the language it recognises.

► **Example 5.** Consider two input symbols $\Sigma = \{a, b\}$ and two nonterminals $N = \{S, X\}$. The following is a BPP in standard form: $S \rightarrow a.X, X \rightarrow a.(X \parallel X) + b.\perp$. An example execution is $S \xrightarrow{a} X \xrightarrow{a} X \parallel X \xrightarrow{b} \perp \parallel X \xrightarrow{b} \perp \parallel \perp$, and thus $a^2b^2 \in L(S)$.

While language equivalence is undecidable for BPP [36, Sec. 5], the finer bisimulation equivalence is decidable [22], and in fact PSPACE-complete [70, 43]. These initial results have motivated a rich line of research investigating decidability and complexity for variants of bisimulation equivalence. We consider another classical variation on language equivalence,

namely multiplicity equivalence, and apply it to decide language equivalence of unambiguous BPP. We show in Corollary 2 that both problems are decidable and in 2-EXPSpace. This is obtained by considering a more general model, introduced next.

2.2 Weighted basic parallel processes

Preliminaries. Let $\Sigma^* \rightarrow \mathbb{Q}$ be the set of (*non-commutative*) series with coefficients in \mathbb{Q} , also known as *weighted languages*. An alternative notation is $\mathbb{Q}\langle\langle\Sigma\rangle\rangle$. We write a series as $f = \sum_{w \in \Sigma^*} f_w \cdot w$, where the value of f at w is $f_w \in \mathbb{Q}$. Thus, $3aba - \frac{5}{2}bc$ and $1 + a + a^2 + \dots$ are series. The set of series carries the structure of a vector space over \mathbb{Q} , with element-wise scalar product $c \cdot f$ ($c \in \mathbb{Q}$) and sum $f + g$. The *support* of a series f is the subset of its domain $\text{supp}(f) \subseteq \Sigma^*$ where it evaluates to a nonzero value. *Polynomials* $\mathbb{Q}\langle\Sigma\rangle$ are series with finite support. The *characteristic series* of a language $L \subseteq \Sigma^*$ is the series that maps words in L to 1 and all the other words to 0.

For two words $u \in \Sigma^m$ and $v \in \Sigma^n$, let $u \sqcup v$ be the multiset of all words $w = a_1 \dots a_{m+n}$ s.t. the set of indices $\{1, \dots, m+n\}$ can be partitioned into two subsequences $i_1 < \dots < i_m$ and $j_1 < \dots < j_n$ s.t. $u = a_{i_1} \dots a_{i_m}$ and $v = a_{j_1} \dots a_{j_n}$. The multiset semantics preserves multiplicities, e.g., $ab \sqcup a = \{\{aab, aab, aba\}\}$. The *shuffle* of two series f, g is the series $f \sqcup g$ defined as $(f \sqcup g)_w := \sum_{w \in u \sqcup v} f_u \cdot g_v$, for every $w \in \Sigma^*$, where the sum is taken with multiplicities. Shuffle product (called *Hurwitz product* in [31]) leads to the commutative *ring of shuffle series* $(\mathbb{Q}\langle\langle\Sigma\rangle\rangle; +, \sqcup, 0, 1)$, whose *shuffle identity* 1 is the series mapping ε to 1 and all other words to 0. A series f has a *shuffle inverse* g , i.e., $f \sqcup g = 1$, iff $f_\varepsilon \neq 0$. The n -th *shuffle power* $f^{\sqcup n}$ of a series f is inductively defined by $f^{\sqcup 0} := 1$ and $f^{\sqcup(n+1)} := f \sqcup f^{\sqcup n}$.

Consider the mapping $\delta : \Sigma^* \rightarrow \mathbb{Q}\langle\langle\Sigma\rangle\rangle \rightarrow \mathbb{Q}\langle\langle\Sigma\rangle\rangle$ s.t. for every $u \in \Sigma^*$ and $f \in \mathbb{Q}\langle\langle\Sigma\rangle\rangle$, $\delta_u f \in \mathbb{Q}\langle\langle\Sigma\rangle\rangle$ is the series defined as $(\delta_u f)_w = f_{uw}$, for every $w \in \Sigma^*$. We call $\delta_u f$ the *u-derivative* of f (a.k.a. *shift* or *left-quotient*). For example, $\delta_a(ab + c) = b$. The derivative operation δ_u is linear, for every $u \in \Sigma^*$. The one-letter derivatives δ_a 's are (noncommuting) derivations of the shuffle ring since they satisfy (Leibniz rule),

$$\delta_a(f \sqcup g) = \delta_a f \sqcup g + f \sqcup \delta_a g, \quad \text{for all } a \in \Sigma, f, g \in \mathbb{Q}\langle\langle\Sigma\rangle\rangle, \quad (2)$$

Syntax and semantics. A *weighted basic parallel process* (WBPP) is a tuple $P = (\Sigma, N, S, F, \Delta)$ where Σ is a finite input alphabet of *terminal symbols/actions*, N is a finite set of *nonterminal symbols/processes*, $S \in N$ is the *initial nonterminal*, $F : N \rightarrow \mathbb{Q}$ assigns a *final weight* $FX \in \mathbb{Q}$ to each nonterminal $X \in N$, and $\Delta : \Sigma \times N \rightarrow \mathbb{Q}[N]$ is a *transition function* mapping a nonterminal $X \in N$ and an input symbol $a \in \Sigma$ to a polynomial $\Delta_a X \in \mathbb{Q}[N]$.

► **Example 6.** A BPP in standard form is readily converted to a WBPP with 0, 1 weights: The BPP from Example 5 yields the WBPP with output function $FS = FX = 0$ and transitions $\Delta_a S = X, \Delta_a X = X^2, \Delta_b S = 0, \Delta_b X = 1$. Configurations reachable from S, X are of the form cX^n ($c \in \mathbb{N}$). Action “ a ” acts as an increment $X^n \xrightarrow{a} nX^{n+1}$ and “ b ” as a decrement $X^n \xrightarrow{b} nX^{n-1}$. The constant coefficient $c \in \mathbb{N}$ in a reachable configuration cX^n keeps track of the “multiplicity” of reaching this configuration, i.e., the number of distinct runs leading to it. For instance, $\llbracket S \rrbracket_{a^2 b^2} = 2$ since $S \xrightarrow{a} X \xrightarrow{a} X^2 \xrightarrow{b} 2X \xrightarrow{b} 2$. In the underlying BPP,

$$S \xrightarrow{a} X \xrightarrow{a} X \parallel X \begin{array}{l} \xrightarrow{b} \perp \parallel X \xrightarrow{b} \perp \parallel \perp \\ \xrightarrow{b} X \parallel \perp \xrightarrow{b} \perp \parallel \perp \end{array}$$

where the branching upon reading the first symbol “ b ” depends on whether the first or second occurrence of X reads this symbol.

A *configuration* of a WBPP is a polynomial $\alpha \in \mathbb{Q}[N]$. The transition function extends uniquely to a derivation of the polynomial ring $\mathbb{Q}[N]$ via linearity and (Leibniz rule):

$$\begin{aligned} \Delta &: \Sigma \times \mathbb{Q}[N] \rightarrow \mathbb{Q}[N] \\ \Delta_a(c \cdot \alpha) &= c \cdot \Delta_a(\alpha), & \forall a \in \Sigma, c \in \mathbb{Q}, \\ \Delta_a(\alpha + \beta) &= \Delta_a(\alpha) + \Delta_a(\beta), & \forall a \in \Sigma, \alpha, \beta \in \mathbb{Q}[N], \\ \Delta_a(\alpha \cdot \beta) &= \Delta_a(\alpha) \cdot \beta + \alpha \cdot \Delta_a(\beta), & \forall a \in \Sigma, \alpha, \beta \in \mathbb{Q}[N]. \end{aligned} \quad (3)$$

For example, from configuration $X \cdot Y$ we can read a and go to $\Delta_a(X \cdot Y) = \Delta_a(X) \cdot Y + X \cdot \Delta_a(Y)$; this models the fact that either X reads a and Y is unchanged, or vice versa. The transition function is then extended homomorphically to words:

$$\begin{aligned} \Delta &: \Sigma^* \times \mathbb{Q}[N] \rightarrow \mathbb{Q}[N] \\ \Delta_\varepsilon \alpha &:= \alpha, \quad \Delta_{a \cdot w} \alpha := \Delta_w(\Delta_a \alpha), \quad \forall (a \cdot w) \in \Sigma^*, \alpha \in \mathbb{Q}[N]. \end{aligned} \quad (4)$$

Sometimes we write $\alpha \xrightarrow{w} \beta$ when $\beta = \Delta_w(\alpha)$. For instance, from configuration α we can read $ab \in \Sigma^*$ visiting configurations $\alpha \xrightarrow{a} \Delta_a(\alpha) \xrightarrow{b} \Delta_b(\Delta_a(\alpha))$. The order of reading symbols matters: For the transition function $\Delta_a(X) = 0$, $\Delta_b(X) = Y$, and $\Delta_a(Y) = \Delta_b(Y) = 1$, we have $X \xrightarrow{a} 0 \xrightarrow{b} 0$ but $X \xrightarrow{b} Y \xrightarrow{a} 1$. The *semantics* of a WBPP is the mapping

$$\begin{aligned} \llbracket _ \rrbracket &: \mathbb{Q}[N] \rightarrow \mathbb{Q}\langle\langle \Sigma \rangle\rangle \\ \llbracket \alpha \rrbracket_w &:= F(\Delta_w \alpha), \quad \forall \alpha \in \mathbb{Q}[N], w \in \Sigma^*. \end{aligned} \quad (5)$$

Here F is extended homomorphically from nonterminals to configurations: $F(\alpha + \beta) = F(\alpha) + F(\beta)$ and $F(\alpha \cdot \beta) = F(\alpha) \cdot F(\beta)$. We say that configuration α *recognises* the series $\llbracket \alpha \rrbracket$. The series recognised by a WBPP is the series recognised by its initial nonterminal. A *WBPP series* is a series which is recognised by some WBPP.

► **Example 7.** We show a WBPP series which is not a WFA series. In particular, its support is nonregular support since WFA supports include the regular languages. Consider the WBPP from Example 6. The language $L := \text{supp}(\llbracket S \rrbracket) \cap a^*b^*$ is the set of words of the form $a^n b^n$, which is not regular, and thus $\text{supp}(\llbracket S \rrbracket)$ is not regular either. Moreover, $\llbracket S \rrbracket$ is not a WFA series: 1) the set M of words of the form $a^m b^n$ with $m \neq n$ is a WFA support, 2) if a language and its complement are WFA supports, then they are regular by a result of Restivo and Reutenauer [65, Theorem 3.1], and 3) since M is not regular, it follows that its complement is not a WFA support, and thus $L = (\Sigma^* \setminus M) \cap a^*b^*$ is not a WFA support either.

2.3 Basic properties

We present some basic properties of the semantics of WBPP. First of all, applying the derivative δ_w to the semantics corresponds to applying Δ_w to the configuration.

► **Lemma 8 (Exchange).** *For every $\alpha \in \mathbb{Q}[N]$ and $w \in \Sigma^*$, $\delta_w \llbracket \alpha \rrbracket = \llbracket \Delta_w \alpha \rrbracket$.*

As a consequence, the semantics is a homomorphism from configurations to series.

► **Lemma 9 (Homomorphism).** *The semantics function $\llbracket _ \rrbracket$ is a homomorphism from the polynomial to the shuffle series ring:*

$$\begin{aligned} \llbracket _ \rrbracket &: (\mathbb{Q}[N]; +, \cdot) \rightarrow (\mathbb{Q}\langle\langle N \rangle\rangle; +, \sqcup) \\ \llbracket c \cdot \alpha \rrbracket &= c \cdot \llbracket \alpha \rrbracket, \quad \llbracket \alpha + \beta \rrbracket = \llbracket \alpha \rrbracket + \llbracket \beta \rrbracket, \quad \llbracket \alpha \cdot \beta \rrbracket = \llbracket \alpha \rrbracket \sqcup \llbracket \beta \rrbracket. \end{aligned}$$

Lemmas 8 and 9 illustrate the interplay between the syntax and semantics of WBPP, and they can be applied to obtain some basic closure properties for the class of WBPP series.

► **Lemma 10** (Closure properties). *Let $f, g \in \mathbb{Q}\langle\langle\Sigma\rangle\rangle$ be WBPP series. The following series are also WBPP: $c \cdot f$, $f + g$, $f \sqcup g$, $\delta_a f$, the shuffle inverse of f (when defined).*

WBPP series generalise the rational series (i.e., recognised by finite weighted automata [8]), which in fact correspond to WBPP with a linear transition relation.

► **Example 11.** The shuffle of two WBPP series with context-free support can yield a WBPP series with non-context-free support. Consider the WBPP from Example 6 over $\Sigma = \{a, b\}$. Make a copy of this WBPP over a disjoint alphabet $\Gamma = \{c, d\}$ with nonterminals $\{T, Y\}$. Now consider the shuffle $f := \llbracket S \rrbracket \sqcup \llbracket T \rrbracket \in \mathbb{Q}\langle\langle\Sigma \cup \Gamma\rangle\rangle$. It is WBPP recognisable by Lemma 10. (For instance we can add a new initial nonterminal U with rules $\Delta_a U = X \cdot T$, $\Delta_c U = S \cdot Y$, and $\Delta_b U = \Delta_d U = 0$.) $\text{supp}(f)$ is not context free, since intersecting it with the regular language $a^*c^*b^*d^*$ yields $\{a^m c^n b^m d^n \mid m, n \in \mathbb{N}\}$, which is not context-free by the pumping lemma for context-free languages [37, Theorem 7.18] (cf. [57, Problem 101]).

2.4 Differential algebra of shuffle-finite series

Differential algebra allows us to provide an elegant characterisation of WBPP series. An *algebra* (over \mathbb{Q}) is a vector space equipped with a bilinear product. Shuffle series are a commutative algebra, called *shuffle series algebra*. A subset of $\mathbb{Q}\langle\langle\Sigma\rangle\rangle$ is a *subalgebra* if it contains \mathbb{Q} and is closed under scalar product, addition, and shuffle product. It is *differential* if it is closed under derivations δ_a ($a \in \Sigma$). By Lemma 10, WBPP series are a differential subalgebra. Let $\mathbb{Q}[f^{(1)}, \dots, f^{(k)}] \subseteq \mathbb{Q}\langle\langle\Sigma\rangle\rangle$ be the smallest subalgebra containing $f^{(1)}, \dots, f^{(k)} \in \mathbb{Q}\langle\langle\Sigma\rangle\rangle$. Algebras of this form are called *finitely generated*. A series is *shuffle finite* if it belongs to a finitely generated differential subalgebra of shuffle series.

► **Theorem 12.** *A series is shuffle finite iff it is WBPP.*

The characterisation above provides an insight into the algebraic structure of WBPP series. Other classes of series can be characterised in a similar style. For instance, a series is accepted by a WFA iff it belongs to a finitely generated differential vector space over \mathbb{Q} [8, Proposition 5.1]; by a weighted context-free grammar iff it belongs to a δ_a -closed, finitely generated subalgebra of the algebra of series with (noncommutative) *Cauchy product* ($(f * g)_w := \sum_{w=uv} f_u \cdot f_v$); and by a polynomial automaton [3] iff its reversal ($f_{a_1 \dots a_n}^R := f_{a_n \dots a_1}$) belongs to a δ_a -closed, finitely generated subalgebra of the algebra of series with *Hadamard product* ($(f \odot g)_w := f_w \cdot g_w$). Considering other products yields novel classes of series, too. For instance, the *infiltration product* [2] yields the class of series that belong to a δ_a -closed, finitely generated subalgebra of the algebra of series with infiltration product.

2.5 Equivalence and zeroness problems

The *WBPP equivalence problem* takes in input two WBPP P, Q and amounts to determine whether $\llbracket P \rrbracket = \llbracket Q \rrbracket$. In the special case where $\llbracket Q \rrbracket = 0$, we have an instance of the *zeroness problem*. Since WBPP series form an effective vector space, equivalence reduces to zeroness, and thus we concentrate on the latter.

Evaluation and word-zeroneess problems. We first discuss a simpler problem, which will be a building block in our zeroness algorithm. The *evaluation problem* takes in input a WBPP with initial configuration α and a word $w \in \Sigma^*$, and it amounts to compute $\llbracket \alpha \rrbracket_w$. The *word-zeroneess problem* takes the same input, and it amounts to decide whether $\llbracket \alpha \rrbracket_w = 0$.

► **Theorem 13.** *The evaluation and word-zeroneess problems for WBPP are in PSPACE.*

The proof follows from the following three ingredients: The construction of an algebraic circuit of exponential size computing the polynomial $\Delta_w \alpha$ (Lemma 14), the fact that this polynomial has polynomial degree (Lemma 15), and the fact that circuits computing multivariate polynomials of polynomial degree can be evaluated in NC [44, Theorem 2.4.5].

► **Lemma 14.** *Fix a word $w \in \Sigma$ and an initial configuration $\alpha \in \mathbb{Q}[N]$ of a WBPP, where $\alpha, \Delta_a X_i \in \mathbb{Q}[N]$ are the outputs of an algebraic circuit of size n . We can construct an algebraic circuit computing $\Delta_w \alpha$ of size $\leq 4^{|w|} \cdot n$. The construction can be done in space polynomial in $|w|$ and logarithmic in n .*

► **Lemma 15.** *Let $D \in \mathbb{N}$ be the maximum of the degree of the transition relation Δ and the initial configuration α . The configuration $\Delta_w \alpha \in \mathbb{Q}[N]$ reached by reading a word $w \in \Sigma^n$ of length n has total degree $O(n \cdot D)$.*

Decidability of the zeroness problem. Fix a WBPP and a configuration $\alpha \in \mathbb{Q}[N]$. Suppose we want to decide whether $\llbracket \alpha \rrbracket$ is zero. An algorithm for this problem follows from first principles. Recall that an *ideal* $I \subseteq \mathbb{Q}[N]$ is a subset closed under addition, and multiplication by arbitrary polynomials [25, §4, Ch. 1]. Let $\langle S \rangle$ be the smallest ideal including $S \subseteq \mathbb{Q}[N]$. Intuitively, this is the set of “logical consequences” of the vanishing of polynomials in S . Build a chain of polynomial ideals

$$I_0 \subseteq I_1 \subseteq \dots \subseteq \mathbb{Q}[N], \text{ with } I_n := \langle \Delta_w \alpha \mid w \in \Sigma^{\leq n} \rangle, n \in \mathbb{N}. \quad (6)$$

Intuitively, I_n is the set of polynomials that vanish as a consequence of the vanishing of $\Delta_w \alpha$ for all words w of length $\leq n$. The chain above has some important structural properties, essentially relying on the fact that the Δ_a ’s are derivations of the polynomial ring.

► **Lemma 16.** *1. $\Delta_a I_n \subseteq I_{n+1}$. 2. $I_{n+1} = I_n + \langle \bigcup_{a \in \Sigma} \Delta_a I_n \rangle$. 3. $I_n = I_{n+1}$ implies $I_n = I_{n+1} = I_{n+2} = \dots$.*

By Hilbert’s finite basis theorem [25, Theorem 4, §5, Ch. 2], there is $M \in \mathbb{N}$ s.t. $I_M = I_{M+1} = \dots$. By Lemma 16 (3) and decidability of ideal inclusion [53], M can be computed. This suffices to decide WBPP zeroness. Indeed, let $\Delta_{w_1} \alpha, \dots, \Delta_{w_m} \alpha$ be the generators of I_M . For every input word $w \in \Sigma^*$ there are $\beta_1, \dots, \beta_m \in \mathbb{Q}[N]$ s.t. $\Delta_w \alpha = \beta_1 \cdot \Delta_{w_1} \alpha + \dots + \beta_m \cdot \Delta_{w_m} \alpha$. By applying the output function F on both sides, we have $\llbracket \alpha \rrbracket_w = F(\Delta_w \alpha) = F\beta_1 \cdot \llbracket \alpha \rrbracket_{w_1} + \dots + F\beta_m \cdot \llbracket \alpha \rrbracket_{w_m}$. It follows that if $\llbracket \alpha \rrbracket_w = 0$ for all words of length $\leq M$, then $\llbracket \alpha \rrbracket = 0$. One can thus enumerate all words w of length $\leq M$ and check $\llbracket \alpha \rrbracket_w = 0$ with Theorem 13. So far we only know that M is computable. In the next section we show that in fact M is an elementary function of the input WBPP.

Elementary upper bound for the zeroness problem. We present an elementary upper bound on the length of the chain of polynomial ideals (6). This is obtained by generalising the case of a single derivation from Novikov and Yakovenko [59, Theorem 4] to the situation of several, not necessarily commuting derivations $\Delta_a, a \in \Sigma$. The two main ingredients in the proof of [59,

18:10 Weighted Basic Parallel Processes and Combinatorial Enumeration

Theorem 4] are 1) a structural property of the chain (6) called *convexity*, and 2) a degree bound on the generators of the n -th ideal I_n (which we have already established in Lemma 15). For two sets $I, J \subseteq \mathbb{Q}[N]$ consider the *colon set* $I : J := \{f \in \mathbb{Q}[N] \mid \forall g \in J, f \cdot g \in I\}$ [25, Def. 5, §4, Ch. 4]. If I, J are ideals of $\mathbb{Q}[N]$ then $I : J$ is also an ideal. An ideal chain $I_0 \subseteq I_1 \subseteq \dots$ is *convex* if the colon ideals $I_n : I_{n+1}$ form themselves a chain $I_0 : I_1 \subseteq I_1 : I_2 \subseteq \dots$. Chain of ideals obtained by iterated application of a single derivation are convex by [59, Lemma 7]. We extend this observation to a finite set of derivations.

► **Lemma 17** (generalisation of [59, Lemma 7]). *The ideal chain (6) is convex.*

Proof. We extend the argument from [59] to the case of many derivations. Assume $f \in I_{n-1} : I_n$ and let $h \in I_{n+1}$ be arbitrary. We have to show $f \cdot h \in I_n$.

▷ **Claim.** $f \cdot \Delta_a g \in I_n$, for all $a \in \Sigma$ and $g \in I_n$.

Proof of the claim. Since Δ_a is a derivation (4), $\Delta_a(f \cdot g) = \Delta_a f \cdot g + f \cdot \Delta_a g$, and by solving for $f \cdot \Delta_a g$ we can write $f \cdot \Delta_a g = \underbrace{\Delta_a(\overbrace{f \cdot g}^{(a) I_{n-1}})}_{(b) I_n} - \underbrace{\Delta_a f \cdot g}_{(c) I_n}$. Condition (a) follows from the definition of colon ideal, (b) from point (1) of Lemma 16, and (c) from I_n being an ideal. ◁

Since $h \in I_{n+1}$, by point (2) of Lemma 16, we can write $h = h_0 + h_1$ with $h_0 \in I_n$ and $h_1 \in \langle \bigcup_{a \in \Sigma} \Delta_a I_n \rangle$. In particular, $h_1 = \sum_i p_i \cdot \Delta_{a_i} g_i$ with $g_i \in I_n$. By the claim, $f \cdot h_1 = \sum_i p_i \cdot f \cdot \Delta_{a_i} g_i \in I_n$. Consequently, $f \cdot h = f \cdot h_0 + f \cdot h_1 \in I_n$ as well. ◀

Thanks to Lemma 17 we can generalise the whole proof of [59, Theorem 4], eventually arriving at the following elementary bound. The *order* of a WBPP is the number of nonterminals and its *degree* is the maximal degree of the polynomials $\Delta_a X$ ($a \in \Sigma, X \in N$).

► **Theorem 18.** *Consider a WBPP of order $\leq k$ and degree $\leq D$. The length of the ideal chain (6) is at most $D^{k^{O(k^2)}}$.*

The elementary bound above may be of independent interest. Already in the case of a single derivation, it is not known whether the bound from [59] is tight, albeit it is expected not to be so. We provide a proof sketch of Theorem 18 in order to illustrate the main notions from algebraic geometry which are required.

Proof sketch. We recall some basic facts from algebraic geometry. The *radical* \sqrt{I} of an ideal I is the set of elements r s.t. $r^m \in I$ for some $m \in \mathbb{N}$; note that \sqrt{I} is itself an ideal. An ideal I is *primary* if $p \cdot q \in I$ and $p \notin I$ implies $q \in \sqrt{I}$. A *primary decomposition* of an ideal I is a collection of primary ideals $\{Q_1, \dots, Q_s\}$, called *primary components*, s.t. $I = Q_1 \cap \dots \cap Q_s$. The *dimension* $\dim I$ of a polynomial ideal $I \subseteq \mathbb{Q}[k]$ is the dimension of its associated variety $V(I) = \{x \in \mathbb{C}^k \mid \forall p \in I, p(x) = 0\}$. Since the operation of taking the variety of an ideal is inclusion-reversing, ideal inclusion is dimension-reversing: $I \subseteq J$ implies $\dim I \geq \dim J$. Consider a convex chain of polynomial ideals as in (6). By convexity, the colon ideals also form a chain $I_0 : I_1 \subseteq I_1 : I_2 \subseteq \dots \subseteq \mathbb{Q}[k]$. The colon dimensions are at most k and non-increasing, $k \geq \dim(I_0 : I_1) \geq \dim(I_1 : I_2) \geq \dots$. Divide the original ideal chain (6) into segments, where in the i -th segment the colon dimension is a constant m_i :

$$\underbrace{I_0 \subseteq \dots \subseteq I_{n_0-1}}_{\dim(I_n : I_{n+1})=m_0} \subseteq \underbrace{I_{n_0} \subseteq \dots \subseteq I_{n_1-1}}_{\dim(I_n : I_{n+1})=m_1} \subseteq \dots \subseteq \underbrace{I_{n_i} \subseteq \dots \subseteq I_{n_{i+1}-1}}_{\dim(I_n : I_{n+1})=m_i} \subseteq \dots \quad (7)$$

Since the colon dimension can strictly decrease at most k times, there are at most k segments. In the following claim we show that the length of a convex ideal chain with equidimensional colon ideal chain can be bounded by the number of primary components of the initial ideal.

▷ **Claim 19** ([59, Lemmas 8+9]). Consider a strictly ascending convex chain of ideals $I_0 \subsetneq I_1 \subsetneq \dots \subsetneq I_\ell$ of length ℓ where the colon ratios have the same dimension $m := \dim(I_0 : I_1) = \dots = \dim(I_{\ell-1} : I_\ell)$. Then ℓ is at most the number of primary components of any primary ideal decomposition of the initial ideal I_0 (counted with multiplicities¹).

We apply Claim 19 to the i -th segment (7) and obtain that its length $\ell_i := n_{i+1} - n_i$ is at most the number of primary components in any primary ideal decomposition of its starting ideal I_{n_i} . We now use a result from effective commutative algebra showing that we can compute primary ideal decompositions of size bounded by the degree of the generators.

▷ **Claim 20** (variant of [59, Corollary 2]). An ideal $I \subseteq \mathbb{C}[k]$ generated by polynomials of degree $\leq D$ admits a primary ideal decomposition of size $D^{k^{O(k)}}$ (counted with multiplicities).

By Claim 20, I_{n_i} admits some primary decomposition of size $d_i^{k^{O(k)}}$, where d_i is the maximal degree of the generators of I_{n_i} . By Lemma 15, d_i is at most $O(D \cdot n_i)$. All in all, the i -th segment has length $\ell_i = n_{i+1} - n_i \leq (D \cdot n_i)^{k^{O(k)}}$. We have $n_i \leq O(f_i)$ where f_i satisfies $f_{i+1} \leq a \cdot f_i^b$ with $a = D^b$ and $b = k^{O(k)}$. Thus $f_k \leq a \cdot a^b \dots a^{b^{k-1}} \leq a^{b^{O(k)}}$, yielding the required upper bound on the length of the ideal chain $n_k \leq D^{k^{O(k^2)}}$. ◀

Thanks to the bound from Theorem 18, we obtain the main contribution of the paper, which was announced in the introduction.

▶ **Theorem 1.** *The zeroness problem for WBPP is in 2-EXPSPACE.*

Proof. The bound on the length of the ideal chain (6) from Theorem 18 implies that if the WBPP is not zero, then there exists a witnessing input word of length at most doubly exponential. We can guess this word and verify its correctness in 2-EXPSPACE by Theorem 13. This is a nondeterministic algorithm, but by courtesy of Savitch's theorem [67] we obtain a *bona fide* deterministic 2-EXPSPACE algorithm. ◀

Application to BPP. The *multiplicity semantics* of a BPP is its series semantics as an N-WBPP. Intuitively, one counts all possible ways in which an input is accepted by the model. The *BPP multiplicity equivalence problem* takes as input two BPP P, Q and returns “yes” iff P, Q have the same multiplicity semantics. Decidability of BPP multiplicity equivalence readily follows from Theorem 1. We say that a BPP is *unambiguous* if its multiplicity semantics is $\{0, 1\}$ -valued. While BPP language equivalence is undecidable [36, Sec. 5], we obtain decidability for unambiguous BPP. We have thus proved Corollary 2. This generalises decidability for deterministic BPP, which follows from decidability of bisimulation equivalence [22]. Language equivalence of unambiguous context-free grammars, the sequential counterpart of BPP (sometimes called BPA in process algebra), is a long-standing open problem, as well as the more general multiplicity equivalence problem (cf. [33, 23, 1]).

¹ We refer to [59, Sec. 4.1] for the notion of *multiplicity* of a primary component.

3 Constructible differentially finite power series

In this section we study a class of multivariate power series in commuting variables called *constructible differentially finite* (CDF) [6, 7]. We show that CDF power series arise naturally as the commutative variant of WBPP series from § 2. Stated differently, the novel WBPP can be seen as the noncommutative variant of CDF, showing a connection between the theory of weighted automata and differential equations. As a consequence, by specialising to the commutative context the 2-EXPSPACE WBPP zeroness procedure, we obtain an algorithm to decide zeroness for CDF power series in 2-EXPTIME. This is the main result of the section, which was announced in the introduction (Theorem 3).

On the way, we recall and further develop the theory of CDF power series. In particular, we provide a novel closure under regular support restrictions (Lemma 24). In § 4 we illustrate a connection between CDF power series and combinatorics, by showing that the generating series of a class of constructible species of structures are CDF, which will broaden the applicability of the CDF zeroness algorithm to multiplicity equivalence of species.

Preliminaries. In the rest of the section, we consider commuting variables $x = (x_1, \dots, x_d)$, $y = (y_1, \dots, y_k)$. We denote by $\mathbb{Q}[[x]]$ the set of multivariate power series in x , endowed with the structure of a commutative ring $(\mathbb{Q}[[x]]; +, \cdot, 0, 1)$ with pointwise addition and (Cauchy) product. The partial derivatives ∂_{x_j} 's satisfy (Leibniz rule), and thus form a family of commuting derivations of this ring. To keep notations compact, we use vector notation: For a tuple of naturals $n = (n_1, \dots, n_d) \in \mathbb{N}^d$, define $n! := n_1! \cdots n_d!$, $x^n := x_1^{n_1} \cdots x_d^{n_d}$, and $\partial_x^n := \partial_{x_1}^{n_1} \cdots \partial_{x_d}^{n_d}$. We write a power series as $f = \sum_{n \in \mathbb{N}^d} f_n \cdot \frac{x^n}{n!} \in \mathbb{Q}[[x]]$, and define the (*exponential*) *coefficient extraction* operation $[x^n]f := f_n$, for every $n \in \mathbb{N}^d$. This is designed in order to have the following simple commuting rule with partial derivative:

$$[x^m](\partial_x^n f) = [x^{m+n}]f, \quad \text{for all } m, n \in \mathbb{N}^d. \quad (8)$$

Coefficient extraction is linear, and *constant term* extraction $[x^0]$ is even a homomorphism since $[x^0](f \cdot g) = [x^0]f \cdot [x^0]g$. The *Jacobian matrix* of a tuple of power series $f = (f^{(1)}, \dots, f^{(k)}) \in \mathbb{Q}[[x]]^k$ is the matrix $\partial_x f \in \mathbb{Q}[[x]]^{k \times d}$ where entry (i, j) is $\partial_{x_j} f^{(i)}$. Consider commuting variables $y = (y_1, \dots, y_k)$. For a set of indices $I \subseteq \{1, \dots, k\}$, by y_I we denote the tuple of variables y_i s.t. $i \in I$ and by $y_{\setminus I}$ we denote the tuple of variables y_i s.t. $i \notin I$. A power series $f \in \mathbb{Q}[[y]]$ is *locally polynomial w.r.t. y_I* if $f \in \mathbb{Q}[[y_I]][[y_{\setminus I}]]$ (f is a power series in $y_{\setminus I}$ with coefficients polynomial in y_I), and that it is *polynomial w.r.t. y_I* if $f \in \mathbb{Q}[[y_{\setminus I}]][[y_I]]$ (f is a polynomial in y_I with coefficients which are power series in $y_{\setminus I}$). For instance $\frac{1}{1-y_1 y_2} = 1 + y_1 y_2 + (y_1 y_2)^2 + \dots$ is not polynomial, but it is locally polynomial in $y_{\{1\}}$ (and $y_{\{2\}}$). A power series $f \in \mathbb{Q}[[x, y]]$ and a tuple $g = (g^{(1)}, \dots, g^{(k)}) \in \mathbb{Q}[[x]]^k$ are *y-composable* if f is locally polynomial w.r.t. y_I , where I is the set of indices i s.t. $g^{(i)}(0) \neq 0$; *strongly y-composable* is obtained by replacing “locally polynomial” with “polynomial”. As a corner case often arising in practice, f, g are always strongly y -composable when $g(0) = 0$. When f, g are y -composable, their *composition* $f \circ_y g \in \mathbb{Q}[[x]]$ obtained by replacing y_i in f with $g^{(i)}$, for every $1 \leq i \leq k$, exists. Composition extends component-wise to vectors and matrices.

3.1 Multivariate CDF power series

A power series $f^{(1)} \in \mathbb{Q}[[x]]$ is CDF [6, 7] if it is the first component of a solution $f = (f^{(1)}, \dots, f^{(k)}) \in \mathbb{Q}[[x]]^k$ of a system of polynomial partial differential equations

$$\partial_x f = P \circ_y f, \quad \text{where } P \in \mathbb{Q}[[x, y]]^{k \times d}. \quad (9)$$

We call k the *order* of the system and d its *dimension*; in the univariate case $d = 1$, (9) is a system of ordinary differential equations. The matrix P is called the *kernel* of the system. The *degree* the system is the maximum degree of polynomials in the kernel, and so it is its *height*. When the kernel does not contain x the system is called *autonomous*, otherwise *non-autonomous*. There is no loss of expressive power in considering only autonomous systems. Many analytic functions give rise to univariate CDF power series, such as polynomials, the exponential series $f := e^x = 1 + x + x^2/2! + \dots$ (since $\partial_x f = f$), the trigonometric series $\sin x$, $\cos x$, $\sec x := 1/\cos x$, \arcsin , \arccos , \arctan their hyperbolic variants \sinh , \cosh , \tanh , $\operatorname{sech} = 1/\cosh$, arsinh , artanh , the non-elementary error function $\operatorname{erf}(x) := \int_0^x e^{-t^2} dt$ (since $\partial_x \operatorname{erf} = e^{-x^2}$ and $\partial_x(e^{-x^2}) = -2x \cdot e^{-x^2}$). Multivariate CDF power series include polynomials, rational power series, constructible algebraic series (in the sense of [31, Sec. 2]; [6, Theorem 4], [7, Corollary 13]), and a large class of D-finite series ([7, Lemma 6]; but not all of them). Moreover, we demonstrate in Theorem 31 that the generating series of strongly constructible species are CDF. We recall some basic closure properties for the class of CDF power series.

► **Lemma 21** (Closure properties; [6, Theorem 2], [7, Theorem 11]). (1) If $f, g \in \mathbb{Q}[[x]]$ are CDF, then are also CDF: $c \cdot f$ for $c \in \mathbb{Q}$, $f + g$, $f \cdot g$, $\partial_{x_j} f$ for $1 \leq j \leq d$, $1/f$ (when defined). (2) If $\partial_{x_1} f, \dots, \partial_{x_d} f$ are CDF, then so is f . (3) Closure under strong composition: If $f \in \mathbb{Q}[[x, y]]$, $g \in \mathbb{Q}[[x]]^k$ are strongly y -composable and CDF, then $f \circ_y g$ is CDF.

► **Remark 22.** In the univariate case $d = 1$, [6, Theorem 2] proves closure under composition under the stronger assumption $g(0) = 0$. In the multivariate case, [7, Theorem 11] claims without proof closure under composition (when defined). We leave it open whether CDF power series are closed under composition.

Of the many pleasant closure properties above, especially composition is remarkable, since this does not hold for other important classes of power series, such as the algebraic and the D-finite power series. For instance, e^x and $e^x - 1$ are D-finite, but $e^{e^x - 1}$ is not [46, Problem 7.8]. On the other hand, CDF power series are not closed under *Hadamard product*, already in the univariate case [6, Sec. 4]. (The *Hadamard product* of $f = \sum_{n \in \mathbb{N}^d} f_n \cdot x^n$, $g = \sum_{n \in \mathbb{N}^d} g_n \cdot x^n \in \mathbb{Q}[[x]]$ is $f \odot g = \sum_{n \in \mathbb{N}^d} (f_n \cdot g_n) \cdot x^n$.) Another paramount closure property regards resolution of systems of power series equations. A system of equations of the constructible form $y = f$ with $f \in \mathbb{Q}[[x, y]]^k$ is *well posed* if $f(0, 0) = 0$ and the Jacobian matrix evaluated at the origin $\partial_y f(0, 0)$ is nilpotent. A *canonical solution* is a series $g \in \mathbb{Q}[[x]]^k$ solving the system for $y := g(x)$ s.t. $g(0) = 0$. The following is a slight generalisation of [7, Corollary 13].

► **Lemma 23** (Constructible power series theorem). A well-posed system of equations $y = f(x, y)$ has a unique canonical solution $y := g(x)$. Moreover, if f is CDF, then g is CDF.

For example, the unique canonical solution of the well-posed equation $y = f := x \cdot e^y$ is CDF.

3.2 Support restrictions

We discuss a novel closure property for CDF power series, which will be useful later in the context of combinatorial enumeration (§ 4). The *restriction* of $f \in \mathbb{Q}[[x]]$ by a *support constraint* $S \subseteq \mathbb{N}^d$ is the series $f|_S \in \mathbb{Q}[[x]]$ which agrees with f on the coefficient of x^n for every $n \in S$, and is zero otherwise. We introduce a small constraint language in order to express a class of support constraints. The set of *constraint expressions* of dimension $d \in \mathbb{N}$ is generated by the following abstract grammar,

$$\varphi, \psi ::= z_j = n \mid z_j \equiv n \pmod{m} \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \neg \varphi, \quad (10)$$

where $1 \leq j \leq d$ and $m, n \in \mathbb{N}$ with $m \geq 1$. Expressions $z_j \leq n$ and $z_j \geq n$ can be derived. The semantics of a constraint expressions φ of dimension d , written $\llbracket \varphi \rrbracket \subseteq \mathbb{N}^d$, is defined by structural induction in the expected way. For instance, the semantics of $z_1 \geq 2 \wedge z_2 \equiv 1 \pmod{2}$ is the set of pairs $(a, b) \in \mathbb{N}^2$ where $a \geq 2$ and b is odd. Call a set $S \subseteq \mathbb{N}^d$ *regular* if it is denoted by a constraint expression.

► **Lemma 24.** *CDF power series are closed under regular support restrictions.*

For instance, since e^x is CDF also $\sinh x = e^x|_{\llbracket z_1 \equiv 1 \pmod{2} \rrbracket}$ is CDF. CDF are not closed under more general *semilinear support restrictions*. E.g., restricting to the semilinear set $\{(m, \dots, m) \in \mathbb{N}^d \mid m \in \mathbb{N}\}$ amounts to taking the *diagonal*, which in turn can be used to express the Hadamard product of power series [50, remark (2) on pg. 377], and CDF are closed under none of these operations.

3.3 CDF = Commutative WBPP series

We demonstrate that CDF power series correspond to WBPP series satisfying a commutativity condition. In particular, they coincide in the univariate case $x = (x_1)$ and $\Sigma = \{a_1\}$. A series $f \in \mathbb{Q}\langle\langle \Sigma \rangle\rangle$ over a finite alphabet $\Sigma = \{a_1, \dots, a_d\}$ is *commutative* if $f_u = f_v$ whenever $\#(u) = \#(v)$; in this case we associate to it a power series $\text{s2p}(f) \in \mathbb{Q}\llbracket x \rrbracket$ in commuting variables $x = (x_1, \dots, x_d)$ by $\text{s2p}(f) := \sum_{n \in \mathbb{N}^d} f_n \cdot \frac{x^n}{n!}$ where $f_n := f_w$ for any $w \in \Sigma^*$ s.t. $\#(w) = n$. Conversely, to any power series $f \in \mathbb{Q}\llbracket x \rrbracket$ we associate a commutative series $\text{p2s}(f) \in \mathbb{Q}\langle\langle \Sigma \rangle\rangle$ by $\text{p2s}(f) := \sum_{w \in \Sigma^*} [x^{\#(w)}]f \cdot w$. These two mappings are mutual inverses and by the following lemma we can identify CDF power series with commutative WBPP series, thus providing a bridge between the theory of weighted automata and differential equations.

► **Lemma 25.** *If $f \in \mathbb{Q}\langle\langle \Sigma \rangle\rangle$ is a commutative WBPP series, then $\text{s2p}(f) \in \mathbb{Q}\llbracket x \rrbracket$ is a CDF power series. Conversely, if $f \in \mathbb{Q}\llbracket x \rrbracket$ is a CDF power series, then $\text{p2s}(f) \in \mathbb{Q}\langle\langle \Sigma \rangle\rangle$ is a commutative WBPP series.*

3.4 Zeroness of CDF power series

Coefficient computation. We provide an algorithm to compute CDF power series coefficients. While a PSPACE algorithm follows from Theorem 13, we are interested here in the precise complexity w.r.t. degree, height, and order. This will allow us obtain the improved 2-EXPTIME complexity for zeroness (Theorem 3).

► **Lemma 26.** *Given a tuple of d -variate CDF power series $f \in \mathbb{Z}\llbracket x \rrbracket^k$ satisfying an integer system of CDF equations (9) of degree D , order k , height H , and a bound N , we can compute all coefficients $[x^n]f \in \mathbb{Z}^k$ with total degree $|n|_1 \leq N$ in deterministic time $\leq (N + d \cdot D + k)^{O(d \cdot D + k)} \cdot (\log H)^{O(1)}$.*

The lemma is proved by a dynamic programming algorithm storing all required coefficients in a table, which is feasible since numerators and denominators are not too big. This rough estimation shows that the complexity is exponential in d, D, k and *polynomial* in N .

Zeroness. The *zeroness problem* for CDF power series takes as input a polynomial $p \in \mathbb{Q}[y]$ and a system of equations (9) with an initial condition $c \in \mathbb{Q}^k$ extending to a (unique) power series solution f s.t. $f(0) = c$, and asks whether $p \circ_y f = 0$.

► **Remark 27.** This is a *promise problem*: We do not decide solvability in power series. In our application in § 4 this is not an issue since power series solutions exist by construction. In the univariate case $d = 1$ the promise is always satisfied. We leave it as future work to investigate the problem of solvability in power series of CDF equations.

The following lemma gives short nonzeroness witnesses. It follows immediately from the WBPP ideal construction (6). Together with Lemma 26 it yields the announced Theorem 3.

► **Lemma 28.** *Consider a CDF $f \in \mathbb{Q}[x]^k$ and $p \in \mathbb{Q}[y]$, both of degree $\leq D$. The power series $g := p \circ_y f$ is zero iff $[x^n]g = 0$ for all monomials x^n of total degree $|n|_1 \leq D^{k \circ(k^2)}$.*

4 Constructible species of structures

The purpose of this section is to show how a rich combinatorial framework for building classes of finite structures (called *species*) gives rise in a principled way to a large class of CDF power series. The main result of this section is that multiplicity equivalence is decidable for a large class of species (Theorem 4). *Combinatorial species of structures* [45] are a formalisation of combinatorics based on category theory, designed in such a way as to expose a bridge between combinatorial operations on species and corresponding algebraic operations on power series. Formally, a d -sorted species is a d -ary endofunctor \mathcal{F} in the category of finite sets and bijections. In particular, \mathcal{F} defines a mapping from d -tuples of finite sets $U = (U_1, \dots, U_d)$ to a finite set $\mathcal{F}[U]$, satisfying certain naturality conditions which ensure that \mathcal{F} is independent of the names of the elements of U . In particular, the cardinality of the output $|\mathcal{F}[U_1, \dots, U_d]|$ depends only on the cardinality of the inputs $|U_1|, \dots, |U_d|$, which allows one to associate to \mathcal{F} the *exponential generating series* (EGS) $\text{EGS}[\mathcal{F}] := \sum_{n \in \mathbb{N}^d} \mathcal{F}_n \cdot \frac{x^n}{n!}$, where $\mathcal{F}_{n_1, \dots, n_d} := |\mathcal{F}[U_1, \dots, U_d]|$ for some (equivalently, all) finite sets of cardinalities $|U_1| = n_1, \dots, |U_d| = n_d$. We refer to [61, Sec. 1] for an introduction to species tailored towards combinatorial enumeration (cf. also the book [5]). Below we present the main ingredients relevant for our purposes by means of examples.

Species can be built from basic species by applying species operations and solving species equations. Examples of *basic species* are the *zero species* $\mathbf{0}$ with EGS 0, the *one species* $\mathbf{1}$ with EGS 1, the *singleton species* \mathcal{X}_j of sort j with EGS x_j , the *sets species* SET with EGS $e^x = 1 + x + x^2/2! + \dots$ (since there is only one set of size n for each n), and the *cycles species* CYC with EGS $-\log(1 - x)$. New species can be obtained by the operations of *sum* (disjoint union) $\mathcal{F} + \mathcal{G}$, *combinatorial product* $\mathcal{F} \cdot \mathcal{G}$ (generalising the Cauchy product for words), *derivative* $\partial_{X_j} \mathcal{F}$ (cf. [61, Sec. 1.2 and 1.4] for formal definitions), and *cardinality restriction* $\mathcal{F}|_S$ (for a *cardinality constraint* $S \subseteq \mathbb{N}^d$). Regarding the latter, $\mathcal{F}|_S$ equals \mathcal{F} on inputs (U_1, \dots, U_d) satisfying $(|U_1|, \dots, |U_d|) \in S$, and is \emptyset otherwise; we use the notation $\mathcal{F}_{\sim n}$ for the constraint $|U_1| + \dots + |U_d| \sim n$, for \sim a comparison operator such as $=$ or \geq .

Another important operation is that of composition of species [61, Sec. 1.5]. Consider sorts $\mathcal{X} = (\mathcal{X}_1, \dots, \mathcal{X}_d)$ and $\mathcal{Y} = (\mathcal{Y}_1, \dots, \mathcal{Y}_k)$. Let \mathcal{F} be a $(\mathcal{X}, \mathcal{Y})$ -sorted species and let $\mathcal{G} = (\mathcal{G}_1, \dots, \mathcal{G}_k)$ be a k -tuple of \mathcal{X} -sorted species. For a set of indices $I \subseteq \{1, \dots, k\}$, we write \mathcal{Y}_I for the tuple of those \mathcal{Y}_i 's s.t. $i \in I$. We say that \mathcal{F} is *polynomial* w.r.t. \mathcal{Y}_I if $\text{EGS}[\mathcal{F}]$ is polynomial w.r.t. y_I , and similarly for *locally polynomial*. We say that \mathcal{F}, \mathcal{G} are \mathcal{Y} -*composable* if \mathcal{F} is locally polynomial w.r.t. \mathcal{Y}_I , where I is the set of indices i s.t. $\mathcal{G}_i[\emptyset] \neq \emptyset$. The notion of *strongly \mathcal{Y} -composable* is obtained by replacing “locally polynomial” with “polynomial”. For two \mathcal{Y} -composable species \mathcal{F}, \mathcal{G} their *composition* $\mathcal{F} \circ_{\mathcal{Y}} \mathcal{G}$ is a well-defined \mathcal{X} -sorted species. Informally, it is obtained by replacing each \mathcal{Y}_i in \mathcal{F} by \mathcal{G}_i .

We will not need the formal definitions of these operations, but we will use the fact that each of these has a corresponding operation on power series [5, Ch. 1]: $\text{EGS}[\mathcal{F} + \mathcal{G}] = \text{EGS}[\mathcal{F}] + \text{EGS}[\mathcal{G}]$, $\text{EGS}[\mathcal{F} \cdot \mathcal{G}] = \text{EGS}[\mathcal{F}] \cdot \text{EGS}[\mathcal{G}]$, $\text{EGS}[\partial_{x_j} \mathcal{F}] = \partial_{x_j} \text{EGS}[\mathcal{F}]$, $\text{EGS}[\mathcal{F} \circ_y \mathcal{G}] = \text{EGS}[\mathcal{F}] \circ_y \text{EGS}[\mathcal{G}]$, and $\text{EGS}[\mathcal{F}|_S] = \text{EGS}[\mathcal{F}]|_S$. For instance, $\text{SET}[\mathcal{X}]_{\geq 1}$ is the species of nonempty sets, with $\text{EGS } e^x - 1$; $\text{SET}[\mathcal{X}] \cdot \text{SET}[\mathcal{X}]$ is the species of *subsets* with $\text{EGS } e^x \cdot e^x = \sum_{n \in \mathbb{N}} 2^n \cdot x^n / n!$ since subsets correspond to partitions of a set into two parts and there are 2^n ways to do this for a set of size n ; $\mathcal{X} \cdot \mathcal{X}$ is the species of pairs with $\text{EGS } 2! \cdot x^2 / 2!$ since there are two ways to organise a set of size 2 into a pair; $\text{SEQ}[\mathcal{X}] = 1 + \mathcal{X} + \mathcal{X} \cdot \mathcal{X} + \dots$ is the species of lists with $\text{EGS } (1 - x)^{-1} = 1 + x + x^2 + \dots$ since there are $n!$ ways to organise a set of size n into a tuple of n elements; $\text{SET}[\mathcal{Y}] \circ_y \text{SET}[\mathcal{X}]_{\geq 1}$ is the species of set partitions with $\text{EGS } e^{e^x - 1}$ since a set partition is a collection of nonempty sets which are pairwise disjoint and whose union is the whole set.

Finally, species can be defined as unique solutions of systems of species equations. E.g., the species of sequences $\text{SEQ}[\mathcal{X}]$ is the unique species satisfying $\mathcal{Y} = \mathbf{1} + \mathcal{X} \cdot \mathcal{Y}$ since a nonempty sequence decomposes uniquely into a first element together with the sequence of the remaining elements; *binary trees* is the unique species solution of $\mathcal{Y} = \mathbf{1} + \mathcal{X} \cdot \mathcal{Y}^2$; *ordered trees* is the unique species solution of $\mathcal{Y} = \mathbf{1} + \mathcal{X} \cdot \text{SEQ}[\mathcal{Y}]$; *Cayley trees* (rooted unordered trees) is the unique species satisfying $\mathcal{Y} = \mathcal{X} \cdot \text{SET}[\mathcal{Y}]$ since a Cayley tree uniquely decomposes into a root together with a set of Cayley subtrees. For a more elaborate example, the species of *series-parallel graphs* is the unique solution for \mathcal{Y}_1 of the following system [61, Sec. 0]:

$$\begin{cases} \mathcal{Y}_1 &= \mathcal{X} + \mathcal{Y}_2 + \mathcal{Y}_3, & (\text{sp graphs}) \\ \mathcal{Y}_2 &= \text{SEQ}[\mathcal{X} + \mathcal{Y}_3]_{\geq 2}, & (\text{series graphs}) \\ \mathcal{Y}_3 &= \text{SET}[\mathcal{X} + \mathcal{Y}_2]_{\geq 2}. & (\text{parallel graphs}) \end{cases} \quad (11)$$

Joyal's *implicit species theorem* [45] (cf. [61, Theorem 2.1], [5, Theorem 2 of Sec. 3.2]), which we now recall, provides conditions guaranteeing existence and uniqueness of solutions to species equations. Let a system of species equations $\mathcal{Y} = \mathcal{F}(\mathcal{X}, \mathcal{Y})$ (with \mathcal{F} a k -tuple of species) be *well posed* if $\mathcal{F}(\mathbf{0}, \mathbf{0}) = \mathbf{0}$ and the Jacobian matrix $\partial_{\mathcal{Y}} \mathcal{F}$ (defined as for power series [61, Sec. 1.6]) is nilpotent at $(\mathbf{0}, \mathbf{0})$. A *canonical solution* is a solution $\mathcal{Y} := \mathcal{G}(\mathcal{X})$ s.t. $\mathcal{G}(\mathbf{0}) = \mathbf{0}$.

► **Theorem 29** (Implicit species theorem [45]). *A well-posed system of species equations $\mathcal{Y} = \mathcal{F}(\mathcal{X}, \mathcal{Y})$ admits a unique canonical solution $\mathcal{Y} := \mathcal{G}(\mathcal{X})$.*

The implicit species theorem is a direct analogue of the implicit function theorem for power series. Furthermore, if $\mathcal{Y} = \mathcal{F}(\mathcal{X}, \mathcal{Y})$ is a well-posed system of species equations then $y = \text{EGS}[\mathcal{F}](x, y)$ is a well-posed system of power series equations; moreover the EGS of the canonical species solution of the former is the canonical power series solution of the latter. We now have enough ingredients to define a large class of combinatorial species. *Strongly constructible species* are the smallest class of species (1) containing the basic species $\mathbf{0}, \mathbf{1}, \mathcal{X}_j$ ($j \in \mathbb{N}$), SET, CYC ; (2) closed under sum, product, strong composition, regular cardinality restrictions; and (3) closed under canonical resolution of well-posed systems $\mathcal{Y} = \mathcal{F}(\mathcal{X}, \mathcal{Y})$ with \mathcal{F} a tuple of strongly constructible species. Note that the equation $\mathcal{Y} = \mathbf{1} + \mathcal{X} \cdot \mathcal{Y}$ for sequences is not well posed, nonetheless sequences are strongly constructible: Nonempty sequences $\text{SEQ}[\mathcal{X}]_{\geq 1}$ are the unique canonical solution of the well-posed species equation $\mathcal{Z} = \mathcal{X} + \mathcal{X} \cdot \mathcal{Z}$ and $\text{SEQ}[\mathcal{X}] = \mathbf{1} + \text{SEQ}[\mathcal{X}]_{\geq 1}$. Similar manipulations show that all the examples mentioned are strongly constructible.

► **Remark 30.** The class of strongly constructible species is incomparable with the class from [61, Definition 7.1]. On the one hand, [61] considers as cardinality restrictions only finite unions of intervals, while we allow general regular restrictions, e.g. periodic constraints such as “even size”; moreover, constraints in [61] are applied only to basic species, while we allow arbitrary strongly constructible species. On the other hand, we consider well-posed systems, while [61] considers more general *well-founded systems*. Finally, we consider strong composition, while [61] considers composition.

Since CDF power series include the the basic species EGS 0 , 1 , x_j ($j \in \mathbb{N}$), $(1-x)^{-1}$, e^x , and $-\log(1-x)$, from the CDF closure properties Lemmas 21, 23, and 24 and the discussion above, we have:

► **Theorem 31.** *The EGS of a strongly constructible species is effectively CDF.*

► **Remark 32.** *Constructible species* are obtained by considering composition instead of strong composition. We conjecture that even the EGS of constructible species are CDF, which would follow by generalising Lemma 21(3) from “strongly composable” to “composable”.

For instance, the well-posed species equation $\mathcal{Y} = \mathcal{X} \cdot \text{SET}[\mathcal{Y}]$ for Cayley trees translates to the well-posed power series equation $y = x \cdot e^y$ for its EGS. The well-posed species equations for series-parallel graphs (11) translate to the following well-posed power series equations for their EGS:

$$\begin{cases} y_1 = x + y_2 + y_3, \\ y_2 = \frac{1}{1-(x+y_3)} - 1 - (x + y_3), \\ y_3 = e^{x+y_2} - 1 - (x + y_2). \end{cases} \quad (12)$$

We conclude this section by deciding multiplicity equivalence of species. Two d -sorted species \mathcal{F}, \mathcal{G} are *multiplicity equivalent* (*equipotent* [61]) if $\mathcal{F}_n = \mathcal{G}_n$ for every $n \in \mathbb{N}^d$. Decidability of multiplicity equivalence of strongly constructible species, announced in Theorem 4, follows from Theorems 3 and 31.

5 Conclusions

We have presented two related computation models, WBPP series and CDF power series. We have provided decision procedures of elementary complexity for their zeroness problems (Theorems 1 and 3), which are based on a novel analysis on the length of chains of polynomial ideals obtained by iterating a finite set of possibly noncommuting derivations (Theorem 18). On the way, we have developed the theory of WBPP and CDF, showing in particular that the latter arises as the commutative variant of the former. Finally, we have applied WBPP to the multiplicity equivalence of BPP (Corollary 2), and CDF to the multiplicity equivalence of constructible species (Theorem 4). Many directions are left for further work. Some were already mentioned in the previous sections. We highlight here some more.

Invariant ideal. Fix a WBPP (or CDF). Consider the *invariant ideal* of all configurations evaluating to zero $Z := \{\alpha \in \mathbb{Q}[N] \mid \llbracket \alpha \rrbracket = 0\}$. Zeroness is just membership in Z . Since Z is a polynomial ideal, it has a finite basis. The most pressing open problem is whether we can compute one such finite basis, perhaps leveraging on differential algebra [48]. Z is computable in the special case of WFA [38, 39], however for polynomial automata it is not [56].

Regular support restrictions. BPP languages are not closed under intersection with regular languages [21, proof of Proposition 3.11], and thus it is not clear for instance whether we can decide BPP multiplicity equivalence within a given regular language. We do not know whether WBPP series are closed under regular support restriction, and thus also zeroness of WBPP series within a regular language is an open problem.

WBPP with edge multiplicities. One can consider a slightly more expressive BPP model where one transition can remove more than one token from the same place [54]. It is conceivable that zeroness stays decidable, however a new complexity analysis is required since the corresponding ideal chains may fail to be convex.

References

- 1 Nikhil Balaji, Lorenzo Clemente, Klara Nosan, Mahsa Shirmohammadi, and James Worrell. Multiplicity problems on algebraic series and context-free grammars. In *Proc. of LICS'23*, pages 1–12, 2023. doi:10.1109/LICS56636.2023.10175707.
- 2 Henning Basold, Helle Hvid Hansen, Jean-Éric Pin, and Jan Rutten. Newton series, coinductively: a comparative study of composition. *Mathematical Structures in Computer Science*, 29(1):38–66, June 2017. doi:10.1017/s0960129517000159.
- 3 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *Proc. of LICS'17*, pages 1–12, June 2017. doi:10.1109/LICS.2017.8005101.
- 4 François Bergeron, Philippe Flajolet, and Bruno Salvy. Varieties of increasing trees. In J. C. Raoult, editor, *CAAP'92*, pages 24–48, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
- 5 François Bergeron, Gilbert Labelle, Pierre Leroux, and Margaret Readdy. *Combinatorial Species and Tree-like Structures*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1998.
- 6 François Bergeron and Christophe Reutenauer. Combinatorial resolution of systems of differential equations iii: A special class of differentially algebraic series. *European Journal of Combinatorics*, 11(6):501–512, 1990.
- 7 François Bergeron and Ulrike Sattler. Constructible differentially finite algebraic series in several variables. *Theoretical Computer Science*, 144(1):59–65, 1995.
- 8 J. Berstel and C. Reutenauer. *Noncommutative rational series with applications*. CUP, 2010.
- 9 Mikołaj Bojańczyk, Bartek Klin, and Joshua Moerman. Orbit-finite-dimensional vector spaces and weighted register automata. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '21. IEEE Press, 2021. doi:10.1109/LICS52264.2021.9470634.
- 10 Michele Boreale. Algebra, coalgebra, and minimization in polynomial differential equations. *Logical Methods in Computer Science*, Volume 15, Issue 1, February 2019.
- 11 Michele Boreale. Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial odes. *Science of Computer Programming*, 193:102441, 2020.
- 12 Michele Boreale. Automatic pre- and postconditions for partial differential equations. *Information and Computation*, 285:104860, 2022.
- 13 Michele Boreale, Luisa Collodi, and Daniele Gorla. Products, polynomials and differential equations in the stream calculus. *ACM Trans. Comput. Logic*, 25(1), January 2024. doi:10.1145/3632747.
- 14 Michele Boreale and Daniele Gorla. Algebra and Coalgebra of Stream Products. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2021.19.


- 15 Alin Bostan, Arnaud Carayol, Florent Koechlin, and Cyril Nicaud. Weakly-Unambiguous Parikh Automata and Their Link to Holonomic Series. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. of ICALP'20*, volume 168 of *LIPICs*, pages 114:1–114:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ICALP.2020.114.
- 16 Alin Bostan and Antonio Jiménez-Pastor. On the exponential generating function of labelled trees. *Comptes Rendus. Mathématique*, 358(9-10):1005–1009, 2020. doi:10.5802/crmath.108.
- 17 François Boulier, Daniel Lazard, François Ollivier, and Michel Petitot. Computing representations for radicals of finitely generated differential ideals. *Applicable Algebra in Engineering, Communication and Computing*, 20(1):73, 2009. doi:10.1007/s00200-009-0091-7.
- 18 Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik und grundl. Math.*, 6:66–92, 1960. doi:10.1002/malq.19600060105.
- 19 Alex Buna-Marginean, Vincent Cheval, Mahsa Shirmohammadi, and James Worrell. On learning polynomial recursive programs. *Proceedings of the ACM on Programming Languages*, 8(POPL):1001–1027, January 2024. doi:10.1145/3632876.
- 20 Michaël Cadilhac, Filip Mazowiecki, Charles Paperman, Michał Pilipczuk, and Géraud Sénizergues. On polynomial recursive sequences. *Theory of Computing Systems*, 2021. doi:10.1007/s00224-021-10046-9.
- 21 Søren Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Department of Computer Science, University of Edinburgh, 1993.
- 22 Søren Christensen, Yoram Hirshfeld, and Faron Moller. Bisimulation equivalence is decidable for basic parallel processes. In *CONCUR'93*, pages 143–157. Springer Berlin Heidelberg, 1993. doi:10.1007/3-540-57208-2_11.
- 23 Lorenzo Clemente. On the complexity of the universality and inclusion problems for unambiguous context-free grammars. In Laurent Fribourg and Matthias Heizmann, editors, *Proceedings 8th International Workshop on Verification and Program Transformation and 7th Workshop on Horn Clauses for Verification and Synthesis*, Dublin, Ireland, 25-26th April 2020, volume 320 of *EPTCS*, pages 29–43. Open Publishing Association, 2020. doi:10.4204/EPTCS.320.2.
- 24 Lorenzo Clemente. Weighted basic parallel processes and combinatorial enumeration. *arXiv e-prints*, page arXiv:2407.03638, July 2024. doi:10.48550/arXiv.2407.03638.
- 25 David A. Cox, John Little, and Donal O’Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer International Publishing, 4 edition, 2015.
- 26 Wojciech Czerwiński and Piotr Hofman. Language Inclusion for Boundedly-Ambiguous Vector Addition Systems Is Decidable. In Bartek Klin, Sławomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.CONCUR.2022.16.
- 27 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, 2009.
- 28 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961. doi:10.1090/S0002-9947-1961-0139530-9.
- 29 Javier Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13–25, 1997. doi:10.3233/fi-1997-3112.
- 30 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
- 31 Michel Fliess. Sur divers produits de séries formelles. *Bulletin de la Société Mathématique de France*, 102:181–191, 1974. doi:10.24033/bsmf.1777.
- 32 Michel Fliess. Réalisation locale des systèmes non linéaires, algèbres de lie filtrées transitives et séries génératrices non commutatives. *Inventiones Mathematicae*, 71(3):521–537, March 1983. doi:10.1007/bf02095991.

- 33 Vojtěch Forejt, Petr Jančar, Stefan Kiefer, and James Worrell. Language equivalence of probabilistic pushdown automata. *Information and Computation*, 237:1–11, 2014. doi:10.1016/j.ic.2014.04.003.
- 34 Andrei Gabrielov and Nicolai Vorobjov. Complexity of computations with pfaffian and noetherian functions. In Y Ilyashenko and C Rousseau, editors, *Normal Forms, Bifurcations and Finiteness Problems in Differential Equations*, NATO Science Series II, page 211. Springer, January 2004.
- 35 Sheila A. Greibach. A new normal-form theorem for context-free phrase structure grammars. *Journal of the ACM*, 12(1):42–52, January 1965. doi:10.1145/321250.321254.
- 36 Yoram Hirshfeld. Petri nets and the equivalence problem. In Egon Börger, Yuri Gurevich, and Karl Meinke, editors, *Computer Science Logic*, pages 165–174, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- 37 John Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2000.
- 38 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. Polynomial invariants for affine programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 530–539, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209142.
- 39 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. On strongest algebraic program invariants. *J. ACM*, August 2023. Just Accepted. doi:10.1145/3614319.
- 40 Hans Hüttel. Undecidable equivalences for basic parallel processes. In *Theoretical Aspects of Computer Software. TACS 1994*, pages 454–464. Springer Berlin Heidelberg, 1994. doi:10.1007/3-540-57887-0_110.
- 41 Hans Hüttel, Naoki Kobayashi, and Takashi Suto. Undecidable equivalences for basic parallel processes. *Information and Computation*, 207(7):812–829, July 2009. doi:10.1016/j.ic.2008.12.011.
- 42 Petr Jančar. Nonprimitive recursive complexity and undecidability for petri net equivalences. *Theoretical Computer Science*, 256(1):23–30, 2001. ISS. doi:10.1016/S0304-3975(00)00100-6.
- 43 Petr Jančar. Strong bisimilarity on basic parallel processes in PSPACE-complete. In *Proc. of LICS'03*, pages 218–227, 2003. doi:10.1109/LICS.2003.1210061.
- 44 Johannes Mittmann. *Independence in Algebraic Complexity Theory*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, December 2013. URL: <https://hdl.handle.net/20.500.11811/5810>.
- 45 André Joyal. Une théorie combinatoire des séries formelles. *Advances in Mathematics*, 42(1):1–82, 1981.
- 46 Manuel Kauers and Peter Paule. *The Concrete Tetrahedron: Symbolic Sums, Recurrence Equations, Generating Functions, Asymptotic Estimates*. Texts and Monographs in Symbolic Computation. Springer-Verlag Wien, 1 edition, 2011.
- 47 S. C. Kleene. Representation of events in nerve nets and finite automata. In Shannon and McCarthy, editors, *Automata Studies*, pages 3–41. Princeton Univ. Press, 1956. URL: http://www.rand.org/pubs/research_memoranda/RM704.html.
- 48 E. R. Kolchin. *Differential Algebra and Algebraic Groups*. Pure and Applied Mathematics 54. Academic Press, Elsevier, 1973.
- 49 Pierre Leroux and Gérard X. Viennot. Combinatorial resolution of systems of differential equations, i. ordinary differential equations. In Gilbert Labelle and Pierre Leroux, editors, *Combinatoire énumérative*, pages 210–245, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg. doi:10.1007/BFb0072518.
- 50 L Lipshitz. The diagonal of a d-finite power series is d-finite. *Journal of Algebra*, 113(2):373–378, 1988. doi:10.1016/0021-8693(88)90166-4.
- 51 Leonard Lipshitz. D-finite power series. *Journal of Algebra*, 122(2):353–373, 1989. doi:10.1016/0021-8693(89)90222-6.

- 52 Prince Mathew, Vincent Penelle, Prakash Saivasan, and A.V. Sreejith. Weighted One-Deterministic-Counter Automata. In Patricia Bouyer and Srikanth Srinivasan, editors, *Proc. of FSTTCS'23*, volume 284 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2023.39.
- 53 Ernst Mayr. Membership in polynomial ideals over q is exponential space complete. In B. Monien and R. Cori, editors, *In Proc. of STACS'89*, pages 400–406, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. doi:10.1007/BFb0029002.
- 54 Ernst W. Mayr and Jeremias Weihmann. *Completeness Results for Generalized Communication-Free Petri Nets with Arbitrary Edge Multiplicities*, pages 209–221. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-41036-9_19.
- 55 Robin Milner. *A calculus of communicating systems*. Lecture Notes in Computer Science 92. Springer-Verlag Berlin Heidelberg, 1 edition, 1980.
- 56 Julian Müllner, Marcel Moosbrugger, and Laura Kovács. Strong Invariants Are Hard: On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs. *arXiv e-prints*, page arXiv:2307.10902, July 2023. doi:10.48550/arXiv.2307.10902.
- 57 Filip Murlak, Damian Niwiński, and Wojciech Rytter, editors. *200 Problems on Languages, Automata, and Computation*. Cambridge University Press, March 2023. doi:10.1017/9781009072632.
- 58 Masakazu Nasu and Namio Honda. Mappings induced by pgsm-mappings and some recursively unsolvable problems of finite probabilistic automata. *Information and Control*, 15(3):250–273, September 1969. doi:10.1016/s0019-9958(69)90449-5.
- 59 Dmitri Novikov and Sergei Yakovenko. Trajectories of polynomial vector fields and ascending chains of polynomial ideals. *Annales de l'Institut Fourier*, 49(2):563–609, 1999.
- 60 Azaria Paz. *Introduction to Probabilistic Automata*. Computer Science and Applied Mathematics. Elsevier Inc, Academic Press Inc, 1971.
- 61 Carine Pivoteau, Bruno Salvy, and Michèle Soria. Algorithms for combinatorial structures: Well-founded systems and Newton iterations. *Journal of Combinatorial Theory, Series A*, 119(8):1711–1773, 2012.
- 62 André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer International Publishing, 1st ed. edition, 2018.
- 63 André Platzer and Yong Kiam Tan. Differential equation invariance axiomatization. *J. ACM*, 67(1), April 2020.
- 64 Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, April 1959. doi:10.1147/rd.32.0114.
- 65 Antonio Restivo and Christophe Reutenauer. On cancellation properties of languages which are supports of rational power series. *J. Comput. Syst. Sci.*, 29(2):153–159, October 1984. doi:10.1016/0022-0000(84)90026-6.
- 66 Christophe Reutenauer. *The Local Realization of Generating Series of Finite Lie Rank*, pages 33–43. Springer Netherlands, 1986. doi:10.1007/978-94-009-4706-1_2.
- 67 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.
- 68 Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270, September 1961. doi:10.1016/s0019-9958(61)80020-x.
- 69 A. Seidenberg. Constructions in algebra. *Transactions of the American Mathematical Society*, 197:273–313, 1974. doi:10.2307/1996938.
- 70 Jiří Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In *STACS 2002*, pages 535–546. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45841-7_44.
- 71 R. P. Stanley. Differentiably finite power series. *European Journal of Combinatorics*, 1(2):175–188, 1980. doi:10.1016/S0195-6698(80)80051-5.

- 72 Richard Stanley. *Enumerative combinatorics*, volume 1 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2ed edition, 2011.
- 73 R. E. Stearns and H. B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, August 1985. doi:10.1137/0214044.
- 74 B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Siberian Math. J.*, 1962.
- 75 A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. doi:10.1112/plms/s2-42.1.230.
- 76 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, April 1992. doi:10.1137/0221017.
- 77 Joris van der Hoeven and John Shackell. Complexity bounds for zero-test algorithms. *Journal of Symbolic Computation*, 41(9):1004–1020, 2006.

Computing Inductive Invariants of Regular Abstraction Frameworks

Philipp Czerner 

Technical University of Munich, Germany

Javier Esparza 

Technical University of Munich, Germany

Valentin Krasotin 

Technical University of Munich, Germany

Christoph Welzel-Mohr 

Technical University of Munich, Germany

Abstract

Regular transition systems (RTS) are a popular formalism for modeling infinite-state systems in general, and parameterised systems in particular. In a CONCUR 22 paper, Esparza et al. introduce a novel approach to the verification of RTS, based on inductive invariants. The approach computes the intersection of all inductive invariants of a given RTS that can be expressed as CNF formulas with a bounded number of clauses, and uses it to construct an automaton recognising an overapproximation of the reachable configurations. The paper shows that the problem of deciding if the language of this automaton intersects a given regular set of unsafe configurations is in EXPSPACE and PSPACE-hard.

We introduce *regular abstraction frameworks*, a generalisation of the approach of Esparza et al., very similar to the regular abstractions of Hong and Lin. A framework consists of a regular language of *constraints*, and a transducer, called the *interpretation*, that assigns to each constraint the set of configurations of the RTS satisfying it. Examples of regular abstraction frameworks include the formulas of Esparza et al., octagons, bounded difference matrices, and views. We show that the generalisation of the decision problem above to regular abstraction frameworks remains in EXPSPACE, and prove a matching (non-trivial) EXPSPACE-hardness bound.

EXPSPACE-hardness implies that, in the worst case, the automaton recognising the overapproximation of the reachable configurations has a double-exponential number of states. We introduce a learning algorithm that computes this automaton in a lazy manner, stopping whenever the current hypothesis is already strong enough to prove safety. We report on an implementation and show that our experimental results improve on those of Esparza et al.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness; Theory of computation → Verification by model checking

Keywords and phrases Regular model checking, abstraction, inductive invariants

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.19

Related Version *Full Version*: <https://arxiv.org/abs/2404.10752> [16]

Supplementary Material *Software*: <https://doi.org/10.5281/zenodo.12734991> [33]
archived at [swh:1:snp:9789ae3f53de50f43367d1c0f108665ea96b619e](https://www.swh.io/snp:9789ae3f53de50f43367d1c0f108665ea96b619e)

1 Introduction

Regular transition systems (RTS) are a popular formalism for modelling infinite-state systems satisfying the following conditions: configurations can be encoded as words, the set of initial configurations is recognised by a finite automaton, and the transition relation is recognised by a transducer. Model checking RTS has been intensely studied under the name of *regular*



© Philipp Czerner, Javier Esparza, Valentin Krasotin, and Christoph Welzel-Mohr; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 19; pp. 19:1–19:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

model checking (see [23, 13, 24, 10] and the surveys [5, 1, 6, 2]). Most regular model checking algorithms address the *safety problem*: given a regular set of unsafe configurations, decide if its intersection with the set of reachable configurations is empty or not. They combine algorithms for the computation of increasingly larger regular subsets of the reachable configurations with acceleration, abstraction, and widening techniques [13, 23, 17, 4, 10, 12, 14, 11, 26, 15].

Recently, Esparza et al. have introduced a novel approach that, starting with the set of all configurations of the RTS, computes increasingly smaller inductive invariants, that is, inductive supersets of the reachable configurations. More precisely, [19] considers invariants given by Boolean formulas in conjunctive normal form with at most b clauses. The paper proves that, for every bound $b \geq 0$, the intersection of *all* inductive b -invariants of the system is recognised by a DFA of double exponential size in the RTS. As a corollary, they obtain that, for every $b \geq 0$, deciding if this intersection contains some unsafe configuration is in EXPSPACE. They also show that the problem is PSPACE-hard, and leave the question of closing the gap open.

In [20] (a revised version of [19]), the EXPSPACE proof is conducted in a more general setting than in [19]. Inspired by this, in our first contribution we show that the approach of [19] can be vastly generalised to arbitrary *regular abstraction frameworks*, consisting of a regular language of *constraints*, and an *interpretation*. Interpretations are functions, represented by transducers, that assign to each constraint a set of configurations, viewed as the set of configurations that *satisfy* the constraint. Examples of regular abstraction frameworks include the formulas of [19] for every $b \geq 0$, views [3], and families of Presburger arithmetic formulas like octagons [27] or bounded difference matrices [25, 8]. A framework induces an abstract interpretation, in which, loosely speaking, the word encoding a constraint is the abstraction of the set of configurations satisfying the constraint. Just as regular model checking started with the observation that different classes of *systems* could be uniformly modeled as RTSs [5, 1, 6, 2], we add the observation, also made in [21], that different classes of *abstractions* can be uniformly modeled as regular abstraction frameworks. We show that the generalisation of the verification problem of [19, 20] to arbitrary regular abstraction frameworks remains in EXPSPACE.

In our second contribution we show that our problem is also EXPSPACE-hard. The reduction (from the acceptance problem for exponentially bounded Turing machines) is surprisingly involved. Loosely speaking, it requires to characterise the set of prefixes of the run of a Turing machine on a given word as an intersection of inductive invariants of a very restrictive kind. We think that this construction can be of independent interest.

Our third and final contribution is motivated by the EXPSPACE-hardness result. A consequence of this lower bound is that the automaton recognising the overapproximation of the reachable configurations must necessarily have a double-exponential number of states in the worst case. We present an approach, based on automata learning, that constructs increasingly larger automata that recognise increasingly smaller overapproximations, and checks whether they are precise enough to prove safety. A key to the approach is solving the separability problem: given a pair (c, c') of configurations, is there an inductive constraint that *separates* c and c' , i.e. is satisfied by c but not by c' ? We show that the problem is PSPACE-complete and NP-complete for interpretations captured by length-preserving transducers. We provide an implementation on top of a SAT solver for the latter case (this is the only case considered in [19, 20]). An experimental comparison shows that this approach beats the one of [19, 20].

Related work. As mentioned above, our first contribution is a reformulation of results of [20] into a more ambitious formalism; it is a conceptual but not a technical novelty. The second and third contributions are new technical results.

Our regular abstraction frameworks are in the same spirit as the regular abstractions of Hong and Lin [21], which use regular languages as abstract objects. In this paper we concentrate on the inductive invariant approach of [19], and in particular on its complexity. This is unlike the approach of [21], which on the one hand is more general, since it also considers liveness properties, but on the other hand does not contain complexity results.

Automata learning has been explored for the verification of regular transition systems multiple times [28, 31, 15, 32, 29]. Roughly speaking, all these approaches formulate a learning process to obtain a *regular* inductive invariant of the system that proves a safety property. Since it is impossible to algorithmically identify the cases where such regular inductive invariant exists, timeouts [15] and resource limits [28] are used as heuristics. In contrast, our approach is designed to always terminate. In particular, we either provide a regular set of constraints that suffices to establish the safety property or a pair of configurations that cannot be separated by inductive constraints of the considered framework. This information can be used to design a more precise framework by adding a new type of constraints.

2 Preliminaries and regular transition systems

Automata. Let Σ be an alphabet. A *nondeterministic finite automaton (NFA)* over Σ is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where Q is a finite set of *states*, $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is the *transition function*, $Q_0 \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final states*. A *run* of A on a word $w = w_1 \cdots w_l \in \Sigma^l$ is a sequence $q_0 q_1 \cdots q_l$ of states where $q_0 \in Q_0$ and $\forall i \in [l] : q_i \in \delta(q_{i-1}, w_i)$. A run on w is *accepting* if $q_l \in F$, and A *accepts* w if there exists an accepting run of A on w . The language *recognised* by A , denoted $L(A)$ or L_A , is the set of words accepted by A . If $|Q_0| = 1$ and $|\delta(q, a)| = 1$ for every $q \in Q, a \in \Sigma$, $|Q_0| = 1$, then A is a *deterministic finite automaton (DFA)*. In this case, we write $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$ and have a single initial state q_0 instead of a set Q_0 .

Relations. Let $R \subseteq X \times Y$ be a relation. The *complement* of R is the relation $\bar{R} := \{(x, y) \in X \times Y \mid (x, y) \notin R\}$. The *inverse* of R is the relation $R^{-1} := \{(y, x) \in Y \times X \mid (x, y) \in R\}$. The *projections* of R onto its first and second components are the sets $R|_1 := \{x \in X \mid \exists y \in Y : (x, y) \in R\}$ and $R|_2 := \{y \in Y \mid \exists x \in X : (x, y) \in R\}$. The *join* of two relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ is the relation $R \circ S := \{(x, z) \in X \times Z \mid \exists y \in Y : (x, y) \in R, (y, z) \in S\}$. The *post-image* of a set $X' \subseteq X$ under a relation $R \subseteq X \times Y$, denoted $X' \circ R$ or $R(X')$, is the set $\{y \in Y \mid \exists x \in X' : (x, y) \in R\}$; the *pre-image*, denoted $R \circ Y$ or $R^{-1}(Y)$, is defined analogously. Throughout this paper, we only consider relations where $X = \Sigma^*$ and $Y = \Gamma^*$ for some alphabets Σ, Γ . We just call them relations. A relation $R \subseteq \Sigma^* \times \Gamma^*$ is *length-preserving* if $(u, w) \in R$ implies $|u| = |w|$.

Convolutions and transducers. Let Σ, Γ be alphabets, let $\# \notin \Sigma \cup \Gamma$ be a padding symbol, and let $\Sigma_{\#} := \Sigma \cup \{\#\}$ and $\Gamma_{\#} := \Gamma \cup \{\#\}$. The *convolution* of two words $u = a_1 \dots a_k \in \Sigma^*$ and $w = b_1 \dots b_l \in \Gamma^*$, denoted $\begin{bmatrix} u \\ w \end{bmatrix}$, is the word over the alphabet $\Sigma_{\#} \times \Gamma_{\#}$ defined as follows. Intuitively, $\begin{bmatrix} u \\ w \end{bmatrix}$ is the result of putting u on top of w , aligned left, and padding the shorter of u and w with $\#$. Formally, if $k \leq l$, then $\begin{bmatrix} u \\ w \end{bmatrix} = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \cdots \begin{bmatrix} a_k \\ b_k \end{bmatrix} \begin{bmatrix} \# \\ b_{k+1} \end{bmatrix} \cdots \begin{bmatrix} \# \\ b_l \end{bmatrix}$, and otherwise $\begin{bmatrix} u \\ w \end{bmatrix} = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \cdots \begin{bmatrix} a_l \\ b_l \end{bmatrix} \begin{bmatrix} a_{l+1} \\ \# \end{bmatrix} \cdots \begin{bmatrix} a_k \\ \# \end{bmatrix}$. The convolution of a tuple of words $u_1 \in \Sigma_1^*, \dots, u_k \in \Sigma_k^*$ is defined analogously, putting all k words on top of each other, aligned left, and padding the shorter words with $\#$.

A *transducer* over $\Sigma \times \Gamma$ is an NFA over $\Sigma_{\#} \times \Gamma_{\#}$. The binary relation recognised by a transducer T over $\Sigma \times \Gamma$, denoted $R(T)$, is the set of pairs $(u, w) \in \Sigma^* \times \Gamma^*$ such that T accepts $\begin{bmatrix} u \\ w \end{bmatrix}$. The definition is generalised to relations of higher arity in the obvious way. In the paper transducers recognise binary relations unless mentioned otherwise. A relation is *regular* if it is recognised by some transducer. A transducer is *length-preserving* if it recognises a length-preserving relation.

Complexity of operations on automata and transducers. Given NFAs A_1, A_2 over Σ with n_1 and n_2 states, DFAs B_1, B_2 over Σ with m_1 and m_2 states, and transducers T_1 over $\Sigma \times \Gamma$ and T_2 over $\Gamma \times \Sigma$ with l_1 and l_2 states, the following facts are well known (see e.g. chapters 3 and 5 of [18]):

- there exist NFAs for $L(A_1) \cup L(A_2)$, $L(A_1) \cap L(A_2)$, and $\overline{L(A_1)}$ with at most $n_1 + n_2$, $n_1 n_2$, and 2^{n_1} states, respectively;
- there exist DFAs for $L(B_1) \cup L(B_2)$, $L(B_1) \cap L(B_2)$, and $\overline{L(B_1)}$ with at most $m_1 m_2$, $m_1 m_2$, and m_1 states, respectively;
- there exist NFAs for $R(T_1)|_1$ and $R(T_1)|_2$ and a transducer for $R(T_1)^{-1}$ with at most l_1 states;
- there exists a transducer for $R(T_1) \circ R(T_2)$ with at most $l_1 l_2$ states; and
- there exist NFAs for $L(A_1) \circ R(T_1)$ and $R(T_1) \circ L(A_2)$ with at most $n_1 l_1$ and $l_1 n_2$ states, respectively.

Regular transition systems

We recall standard notions about regular transition systems and fix some notations. A *transition system* is a pair $\mathcal{S} = (\mathcal{C}, \Delta)$ where \mathcal{C} is the set of all possible *configurations* of the system, and $\Delta \subseteq \mathcal{C} \times \mathcal{C}$ is a *transition relation*. The *reachability relation* $Reach$ is the reflexive and transitive closure of Δ . Observe that, by our definition of post-set, $\Delta(C)$ and $Reach(C)$ are the sets of configurations reachable in one step and in arbitrarily many steps from C , respectively.

Regular transition systems are transition systems where Δ can be finitely represented by a transducer. Formally:

► **Definition 1.** A transition system $\mathcal{S} = (\mathcal{C}, \Delta)$ is regular if \mathcal{C} is a regular language over some alphabet Σ , and Δ is a regular relation. We abbreviate regular transition system to *RTS*.

RTSs are often used to model parameterised systems [5, 1, 6, 2]. In this case, Σ is the set of possible *states* of a process, the set of configurations is $\mathcal{C} = \Sigma^* \setminus \{\varepsilon\}$, and a configuration $a_1 \cdots a_n \in \Sigma^*$ describes the global state of an *array* consisting of n identical copies of the process, with the i -th process in state a_i for every $1 \leq i \leq n$. The transition relation Δ describes the possible transitions of all arrays, of any length.

► **Example 2 (Token passing [5]).** We use a version of the well-known token passing algorithm as running example. We have an array of processes of arbitrary length. At each moment in time, a process either has a token (t) or not (n). Initially, only the first process has a token. A process that has a token can pass it to the process to the right if that process does not have one. We set $\Sigma = \{t, n\}$, and so $\mathcal{C} = \{t, n\}^* \setminus \{\varepsilon\}$. We have $c_2 \in \Delta(c_1)$ iff the word $\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ belongs to the regular expression $(\begin{bmatrix} n \\ n \end{bmatrix} + \begin{bmatrix} t \\ t \end{bmatrix})^* (\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}) (\begin{bmatrix} n \\ n \end{bmatrix} + \begin{bmatrix} t \\ t \end{bmatrix})^*$. For the set of initial configurations $C_I := tn^*$ where only the first process has a token, the set of reachable configurations is $Reach(C_I) = n^* t n^*$.

3 Regular abstraction frameworks

In the same way that RTSs can model multiple classes of *systems* (e.g. parameterised systems with synchronous/asynchronous, binary/multiway/broadcast communication), regular abstraction frameworks are a formalism to model a wide range of *abstractions*.

► **Definition 3.** An abstraction framework is a triple $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$, where \mathcal{C} is a set of configurations, \mathcal{A} is a set of constraints, and $\mathcal{V} \subseteq \mathcal{A} \times \mathcal{C}$ is an interpretation. \mathcal{F} is regular if \mathcal{C} and \mathcal{A} are regular languages over alphabets Σ and Γ , respectively, and the interpretation \mathcal{V} is a regular relation over $\mathcal{A} \times \mathcal{C}$.

Intuitively, the constraints of an abstraction framework are the abstract objects of the abstraction, and $\mathcal{V}(A)$ is the set of configurations abstracted by A . The following remark formalises this.

► **Remark 4.** An abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ induces an abstract interpretation as follows. The concrete and abstract domains are $(2^{\mathcal{C}}, \leq_{\mathcal{C}})$ and $(2^{\mathcal{A}}, \leq_{\mathcal{A}})$, respectively, where $\leq_{\mathcal{C}} := \subseteq$ and $\leq_{\mathcal{A}} := \supseteq$. Both are complete lattices. The concretisation function $\gamma: 2^{\mathcal{A}} \rightarrow 2^{\mathcal{C}}$ and the abstraction function $\alpha: 2^{\mathcal{C}} \rightarrow 2^{\mathcal{A}}$ are given by:

- $\gamma(\mathcal{A}') := \bigcap_{A \in \mathcal{A}'} \mathcal{V}(A)$. Intuitively, $\gamma(\mathcal{A}')$ is the set of configurations that satisfy all constraints of \mathcal{A}' . In particular, $\gamma(\emptyset) = \mathcal{C}$.
- $\alpha(\mathcal{C}') := \{A \in \mathcal{A} \mid \mathcal{C}' \subseteq \mathcal{V}(A)\}$. Intuitively, $\alpha(\mathcal{C}')$ is the set of constraints satisfied by all configurations in \mathcal{C}' . In particular, $\alpha(\emptyset) = \mathcal{A}$.

It is easy to see that the functions α and γ form a Galois connection, that is, for all $C \subseteq \mathcal{C}$ and $A \subseteq \mathcal{A}$, we have $B \subseteq \alpha(C) \Leftrightarrow C \subseteq \gamma(B)$.

Regular abstractions can be combined to yield more precise ones. Given abstraction frameworks $\mathcal{F}_1 = (\mathcal{C}, \mathcal{A}_1, \mathcal{V}_1)$ and $\mathcal{F}_2 = (\mathcal{C}, \mathcal{A}_2, \mathcal{V}_2)$, we can define new frameworks $(\mathcal{C}, \mathcal{A}, \mathcal{V})$ by means of the following operations:

- **Union:** $\mathcal{A} := \mathcal{A}_1 \cup \mathcal{A}_2$, $\mathcal{V}(A) := \mathcal{V}_1(A)$ if $A \in \mathcal{A}_1$, else $\mathcal{V}_2(A)$.
A constraint of the union framework is either a constraint of the first framework, or a constraint of the second.
- **Convolution:** $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$, $\mathcal{V}(A_1, A_2) := \mathcal{V}_1(A_1) \cap \mathcal{V}_2(A_2)$.
A constraint of the convolution framework is the conjunction of two constraints, one of each framework. This operation is implicitly used in [19]: the constraint for a Boolean formula with b clauses is the convolution, applied b times, of the constraints for formulas with one clause.

The proof of the following lemma is in the full version of the paper [16].

► **Lemma 5.** Regular abstraction frameworks are closed under union and convolution. If the interpretations of the frameworks are recognised by transducers with n_1 and n_2 states, then the interpretations of the union and convolution frameworks are recognised by transducers with $O(n_1 + n_2)$ and $O(n_1 n_2)$ states, respectively.

Many abstractions used in the literature can be modeled as regular abstraction frameworks. We give some examples.

► **Example 6.** Consider a transition system where $\mathcal{C} = \mathbb{N}^d$ for some d , and Δ is given by a formula of Presburger arithmetic $\delta(\mathbf{x}, \mathbf{x}')$, that is, $(\mathbf{n}, \mathbf{n}') \in \Delta$ iff $\delta(\mathbf{n}, \mathbf{n}')$ holds. It is well-known that for any Presburger formula there is a transducer recognising the set of its solutions when numbers are encoded in binary, and so with this encoding (\mathcal{C}, Δ) is an RTS. Any Presburger formula $\varphi(\mathbf{x}, \mathbf{y})$, where \mathbf{x} has dimension d and \mathbf{y} has some arbitrary

dimension e , induces a regular abstraction framework as follows. The set of constraints is the set of all tuples $\mathbf{m} \in \mathbb{N}^e$; the interpretation assigns to \mathbf{m} all tuples \mathbf{n} such that $\varphi(\mathbf{n}, \mathbf{m})$ holds. Intuitively, the constraints are the formulas $\varphi_{\mathbf{m}}(\mathbf{x}) := \varphi(\mathbf{x}, \mathbf{m})$, but using \mathbf{m} as encoding of $\varphi_{\mathbf{m}}$.

Special cases of this setting are used in many different areas. For example, bounded difference matrices (see e.g. [25, 8]) and octagons [27] correspond to abstraction frameworks with constraints $\varphi(x_1, x_2, y)$ of the form $x_1 \pm x_2 \leq y$.

► **Example 7.** The approach to regular model checking of [19] is another instance of a regular abstraction framework. The paper encodes sets of configurations as positive Boolean formulas in conjunctive normal form with a bounded number b of clauses. We explain this by means of an example. Consider an RTS with $\Sigma = \{a, b, c\}$ and $\mathcal{C} = \Sigma^*$. Consider the formula $\varphi = (a_{1:5} \vee b_{1:5} \vee a_{3:5}) \wedge b_{4:5}$. We interpret φ on configurations. The intended meaning of a literal, say $a_{1:5}$, is “if the configuration has length 5, then its first letter is an a .” So the set of configurations satisfying the formula is $\Sigma^{\leq 4} + \Sigma^6 \Sigma^* + (a + b)\Sigma^2 b \Sigma + \Sigma^2 ab \Sigma$. In the formulas of [19] all literals have the same length, where the length of a literal $x_{i:j}$ is j .

Formulas with at most b clauses can be encoded as words over the alphabet $\Gamma = (2^\Sigma)^b$. Each clause is encoded as a word over 2^Σ . For example, the encodings of the clauses $(a_{1:5} \vee b_{1:5} \vee a_{3:5})$ and $b_{4:5}$ are $\{a, b\}^0 \{a\}^0 \emptyset^0$ and $\emptyset^0 \emptyset^0 \{b\}^0$, and the encoding of φ is the convolution of the encodings of the clauses. It is easy to see that the interpretation of [19] that assigns to a formula the set of configurations satisfying it is a regular relation recognised by a transducer with 2^b states [19]. In particular, for the case $b = 1$ we get the two-state transducer on the left of Figure 1.

► **Example 8.** In [3] Abdulla et al. introduce *view abstraction* for the verification of parameterised systems. Given a number $k \geq 1$, a *view* of a word $w \in \Sigma^*$ is a scattered subword of w . Loosely speaking, Abdulla et al. abstract a word by its set of views of length up to k . In our setting, a constraint is a set $F \subseteq \Sigma^{\leq k}$ of “forbidden views”, and $\mathcal{V}(F)$ is the set of all words that do not contain any view of F . Since k is fixed, this interpretation is regular.

3.1 The abstract safety problem

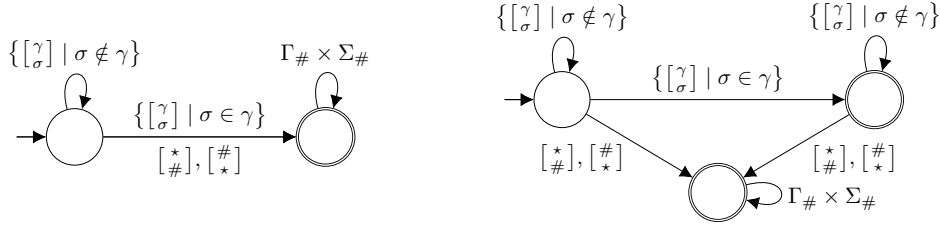
We apply regular abstraction frameworks to the problem of deciding whether an RTS avoids some regular set of unsafe configurations. For simplicity, we assume w.l.o.g. that the set of configurations of the RTS is Σ^{*1} . Let us first formalise the SAFETY problem:

Given: a nondeterministic transducer recognising a regular relation $\Delta \subseteq \Sigma^* \times \Sigma^*$, and two NFAs recognising regular sets $C_I, C_U \subseteq \Sigma^*$ of initial and unsafe configurations, respectively.

Decide: does $\text{Reach}(C_I) \cap C_U = \emptyset$ hold?

It is a folklore result that SAFETY is undecidable. Let us sketch the argument. The configurations of a given Turing machine can be encoded as words of the form $w_l q w_r$, where w_l, w_r encode the contents of the tape to the left and to the right of the head, and q encodes the current state. With this encoding, the successor relation between configurations of the Turing machine is regular, and so is the set of accepting configurations. Taking the latter as set of unsafe configurations, the Turing machine accepts a given initial configuration iff the RTS started at the initial configuration is unsafe.

¹ By interpreting Δ as a relation over $\Sigma \times \Sigma$, any RTS can be transformed into an equivalent one with the same transitions where the set of configurations is Σ^* .



■ **Figure 1** Transducers for the interpretations of Example 7 and 10. We have $\Gamma = 2^{\Sigma}$, and so the alphabet of the transducer is $(2^{\Sigma})_{\#} \times \Sigma_{\#}$. The symbols $\begin{bmatrix} * \\ \# \end{bmatrix}$ and $\begin{bmatrix} \# \\ * \end{bmatrix}$ stand for the sets of all letters of the form $\begin{bmatrix} \gamma \\ \# \end{bmatrix}$ and $\begin{bmatrix} \# \\ \sigma \end{bmatrix}$, respectively.

AbstractSafety. We show that regular abstraction framework induces an “abstract” version of the safety problem, in which we replace the reachability relation by an overapproximation derived from the abstraction framework. Fix an RTS $\mathcal{S} = (\mathcal{C}, \Delta)$ and a regular abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$. We introduce some definitions:

► **Definition 9.** A set $C \subseteq \mathcal{C}$ of configurations is inductive if $\Delta(C) \subseteq C$. A constraint A is inductive if $\mathcal{V}(A)$ is inductive. We let $Ind \subseteq \mathcal{A}$ denote the set of all inductive constraints of \mathcal{A} . Given two configurations c, c' and $A \in Ind$, we say that A separates c from c' if $c \in \mathcal{V}(A)$ and $c' \notin \mathcal{V}(A)$.

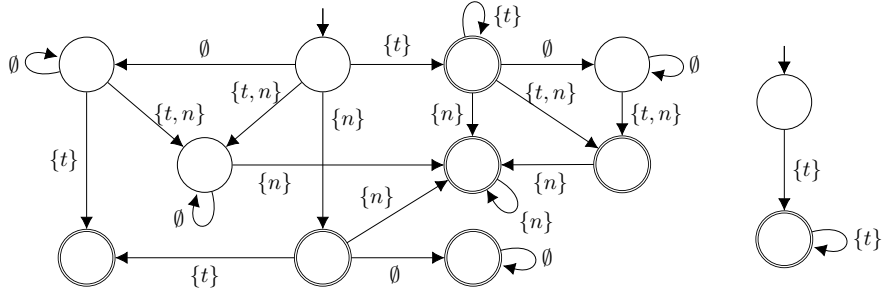
It is a folklore result that $Reach(C)$ is the smallest inductive set containing C , and that if some $A \in Ind$ separates c and c' , then $(c, c') \notin Reach$. Hence, an abstraction framework $(\mathcal{C}, \mathcal{A}, \mathcal{V})$ induces a *potential reachability* relation $PReach \subseteq \mathcal{C} \times \mathcal{C}$, defined as the set of all pairs of configurations that are not separated by any inductive constraint. Formally:

$$PReach := \{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \forall A \in Ind: c \in \mathcal{V}(A) \rightarrow c' \in \mathcal{V}(A)\}$$

We have $Reach(C) \subseteq PReach(C)$ for every set of configurations C . In particular, given sets $C_I, C_U \subseteq \mathcal{C}$ of initial and unsafe configurations, if $PReach(C_I) \cap C_U = \emptyset$, then the RTS is safe.

► **Example 10.** Consider the RTS of the token passing system of Example 2, where $\Sigma = \{t, n\}$. We give two examples of abstraction frameworks. The first one is the abstraction framework of [19], already presented in Example 7, with $b = 1$. We have $\Gamma = 2^{\Sigma} = \{\emptyset, \{t\}, \{n\}, \Sigma\}$. A constraint like $\varphi = \bigvee_{i=3}^5 t_{i:5}$ is encoded by the word $\emptyset\emptyset\{t\}\{t\}\{t\} \in \Gamma^*$, and interpreted as the set of all configurations of length 5 that have a token at positions 3, 4, or 5, plus the set of all configurations of length different from 5. The two-state transducer for this interpretation is on the left of Figure 1. For example, the left state has transitions leading to itself for the letters $\begin{bmatrix} \emptyset \\ n \end{bmatrix}, \begin{bmatrix} \emptyset \\ t \end{bmatrix}, \begin{bmatrix} \{t\} \\ n \end{bmatrix}, \begin{bmatrix} \{n\} \\ t \end{bmatrix}$. The constraint φ is inductive. In fact, the language of all non-trivial inductive constraints (a constraint is trivial if it is satisfied by all configurations or by none) is $\{n\}^+\emptyset^*\{t\}^* + \{n\}^*\emptyset^*\{t\}^+$. The set of configurations potentially reachable from $C_I = tn^*$ is $PReach(C_I) = (tn + nn^*t)(t + n)^*$. In particular, $PReach(C_I) \cap n^* = \emptyset$, but $tnt \in PReach(C_I)$. So this abstraction framework is strong enough to prove that every reachable configuration has at least one token, but not to prove that it has exactly one.

Consider now the framework in which, instead of a *disjunction* of literals, a constraint is an *exclusive disjunction* of literals, that is, a configuration satisfies the constraint if it satisfies *exactly one* of its literals. So, in particular, the interpretation of $\emptyset\emptyset\{t\}\{t\}\{t\}$ is now that exactly one of the positions 3, 4, and 5 has a token. The interpretation is also



■ **Figure 2** On the left, DFA recognising all non-trivial inductive constraints of Example 17. On the right, fragment with the same interpretation as the DFA on the left.

regular; it is given by the three-state transducer on the right of Figure 1. Examples of inductive constraints are $\{t\}\emptyset\{t,n\}\{n\}$ and all words of $\{t\}^*$. The language of non-trivial inductive constraints is given by the DFA on the left of Figure 2. Observe that the set of words satisfying all constraints of $\{t\}^*$ is the language n^*tn^* . In particular, we have $PReach(C_I) \subseteq n^*tn^* = Reach(C_I)$, and so $PReach(C_I) = Reach(C_I)$.

The **ABSTRACTSAFETY** problem is defined exactly as **SAFETY**, just replacing the reachability set $Reach(C_I)$ by the potential reachability set $PReach(C_I)$ implicitly defined by the regular abstraction framework:

Given: a nondeterministic transducer recognising a regular relation $\Delta \subseteq \Sigma^* \times \Sigma^*$; two NFAs recognising regular sets $C_I, C_U \subseteq \Sigma^*$ of initial and unsafe configurations, respectively; and a deterministic transducer recognising a regular interpretation \mathcal{V} over $\Gamma \times \Sigma$.

Decide: does $PReach(C_I) \cap C_U = \emptyset$ hold?

Recall that **SAFETY** is undecidable. In the rest of this section and in the next one we show that **ABSTRACTSAFETY** is **EXPSpace**-complete. Membership in **EXPSpace** was essentially proved in [20], while **EXPSpace**-hardness was left open. We briefly summarise the proof of membership in **EXPSpace** presented in [20], for future reference in our paper.

► **Remark 11.** The result we prove in Section 3.2 is slightly more general. In [20], membership in **EXPSpace** is only proved for RTSs whose transducers are length-preserving, while we prove it in general. General transducers allow one to model parameterised systems with process creation. For example, we can model a token passing algorithm in which the size of the array can dynamically grow and shrink by adding the transitions $\left(\begin{bmatrix} n \\ n \end{bmatrix} + \begin{bmatrix} t \\ t \end{bmatrix}\right)^+ \left(\begin{bmatrix} n \\ \# \end{bmatrix} + \begin{bmatrix} \# \\ n \end{bmatrix}\right)$ to the transition relation of Example 2.

3.2 AbstractSafety is in EXPSpace

We first show that the set of all inductive constraints of a regular abstraction framework is a regular language. Fix a regular abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ over an RTS (\mathcal{C}, Δ) . Let $n_\Delta, n_\mathcal{V}, n_I, n_U$ be the number of states of the transducers and NFAs of a given instance of **ABSTRACTSAFETY**.

► **Lemma 12** ([20]). *The set \overline{Ind} is regular. Further, one can compute an NFA with at most $n_\Delta \cdot n_\mathcal{V}^2$ states recognising \overline{Ind} , and a DFA with at most $2^{n_\Delta \cdot n_\mathcal{V}^2}$ states recognising Ind .*

Proof. By definition, we have

$$\begin{aligned}\overline{Ind} &= \{A \in \Gamma^* \mid \exists c, c' \in \mathcal{C}: c \in \mathcal{V}(A), c' \in \Delta(c), \text{ and } c' \notin \mathcal{V}(A)\} \\ &= \{A \in \Gamma^* \mid \exists c, c' \in \mathcal{C}: (A, c) \in \mathcal{V}, (c, c') \in \Delta \text{ and } (c', A) \in \overline{\mathcal{V}^{-1}}\}\end{aligned}$$

Let $Id_\Gamma = \{(A, A) \mid A \in \Gamma^*\}$. We obtain $\overline{Ind} = ((\mathcal{V} \circ \Delta \circ \overline{\mathcal{V}^{-1}}) \cap Id_\Gamma) \upharpoonright_1$. By the results at the end of Section 2, \overline{Ind} is recognised by a NFA with $n_\mathcal{V} \cdot n_\Delta \cdot n_\mathcal{V} = n_\Delta n_\mathcal{V}^2$ states, and so Ind is recognised by a DFA with $2^{n_\Delta \cdot n_\mathcal{V}^2}$ states. ◀

► **Lemma 13** ([20]). *The potential reachability relation $PReach$ is regular. Further, one can compute a nondeterministic transducer with at most $K := n_\mathcal{V}^2 \cdot 2^{n_\Delta \cdot n_\mathcal{V}^2}$ states recognising $PReach$, and a deterministic transducer with at most 2^K states recognising $PReach$.*

Proof. By definition, we have

$$\begin{aligned}\overline{PReach} &= \{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \exists A \in Ind: c \in \mathcal{V}(A) \text{ and } c' \notin \mathcal{V}(A)\} \\ &= \{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \exists A \in Ind: (c, A) \in \mathcal{V}^{-1} \text{ and } (A, c') \in \overline{\mathcal{V}}\}\end{aligned}$$

Let $Id_\Gamma = \{(A, A) \mid A \in \Gamma^*\}$. We obtain $\overline{PReach} = (\mathcal{V}^{-1} \circ (Id_\Gamma \cap Ind^2) \circ \overline{\mathcal{V}})$. Apply now the results at the end of Section 2 and Lemma 12. ◀

► **Theorem 14** ([20]). *ABSTRACTSAFETY is in EXPSpace.*

Proof. Immediate consequence of Lemma 13, see the full version of the paper [16]. ◀

4 AbstractSafety is EXPSpace-hard

In [19] it was shown that ABSTRACTSAFETY was PSPACE-hard, and the paper left the question of closing the gap between the upper and lower bounds open. We first recall and slightly alter the PSPACE-hardness proof of [19], and then present our techniques to extend it to EXPSpace-hardness.

The proof is by reduction from the problem of deciding whether a Turing machine \mathcal{M} of size n does not accept when started on the empty tape of size n . (For technical reasons, we actually assume that the tape has $n - 2$ cells.) Given \mathcal{M} , we construct in polynomial time an RTS \mathcal{S} and a set of initial configurations C_I that, loosely speaking, satisfy the following two properties: the execution of \mathcal{S} on an initial configuration simulates the run of \mathcal{M} on the empty tape, and $PReach(C_I) = Reach(C_I)$. We choose C_U as the set of configurations of \mathcal{S} in which \mathcal{M} ends up in the accepting state. Then \mathcal{S} is safe iff \mathcal{M} does not accept.

Turing machine preliminaries. We assume that \mathcal{M} is a deterministic Turing machine with states Q , tape alphabet Γ' , initial state q_0 and accepting state q_f .

We represent a configuration of \mathcal{M} as a word $\# \beta q \eta$ of length n , where \mathcal{M} is in state q , the content of the tape is $\beta \eta \in \Gamma'^*$, and the head of \mathcal{M} is positioned at the first letter of η . The symbol $\#$ serves as a separator between different configurations. The initial configuration is $\alpha_0 := \# q_0 B^{n-2}$, where B denotes the blank symbol of \mathcal{M} ; so the tape is initially empty.

We assume w.l.o.g. that the successor of a configuration in state q_f is the configuration itself, so the run of \mathcal{M} can be encoded as an infinite word $\alpha := \alpha_0 \alpha_1 \dots$ where α_i represents the i -th configuration of \mathcal{M} . For convenience, we write $\Lambda := Q \cup \Gamma' \cup \{\#\}$ for the set of symbols in α . It is easy to see that the symbol at position $i + n$ of α is completely determined

by the symbols at positions $i - 1$ to $i + 2$ and the transition relation of \mathcal{M} . We let $\delta(x_1x_2x_3x_4)$ denote the symbol which “should” appear at position $i + n$ when the symbols at positions $i - 1$ to $i + 2$ are $x_1x_2x_3x_4$; in particular, $\delta(x_1\#x_2x_4) = \#$.

Configurations of \mathcal{S} . We choose the set of configurations as $\mathcal{C} := \alpha_0\#(\Lambda \cup \{\square\})^*$, and the initial configurations as $C_I := \alpha_0\#\square^*$. Intuitively, the RTS starts with the representation of the initial configuration of \mathcal{M} , followed by some number of blank cells \square . During its execution, the RTS will “write” the run of \mathcal{M} into these blanks.

A configuration is unsafe if it contains some occurrence of q_f , the accepting state of \mathcal{M} , so $C_U := (\Lambda \cup \{\square\})^*\{q_f\}(\Lambda \cup \{\square\})^*$.

Transitions. For convenience, we will denote the i -th position of a word w as $w(i)$ instead of w_i . Given a configuration c , the set $\Delta(c)$ contains one single configuration c' , defined as follows. Let i be some position of c such that $c_{i+n} = \square$. Then c' coincides with c everywhere except at position $i + n$, where instead $c'(i + n) := \delta(c(i - 1)c(i)c(i + 1)c(i + 2))$. It is easy to see that Δ is a regular relation: The transducer nondeterministically guesses the position $i - 1$, reads the next four symbols, say $x_1\dots x_4$, stores $\delta(x_1\dots x_4)$ in its state, moves to position $i + n$, checks if $c(i + n) = \square$ and writes $c'(i + n) := \delta(x_1\dots x_4)$. The transducer has $O(n^2)$ states.

It follows from the definitions above that \mathcal{M} accepts the empty word iff \mathcal{S} can reach C_U from C_I , i.e. $\text{Reach}(C_I) \cap C_U \neq \emptyset$.

Regular abstraction framework. We define a regular abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ of polynomial size such that $\text{PReach}(C_I) = \text{Reach}(C_I)$. Hence, for every configuration $c \notin \text{Reach}(C_I)$, we must find an inductive constraint $A \in \mathcal{A}$ which separates C_I and c . (Note that C_I contains exactly one configuration of length $|c|$.)

As the reachable configurations are precisely the prefixes of α with some symbols replaced by \square , there is a position i s.t. $c(i) \notin \{\square, \alpha(i)\}$. Let us fix the smallest such i . As we noted above, $\alpha(i)$ is determined entirely by $\alpha(i - n - 1)\dots\alpha(i - n + 2)$ via the mapping δ . So the constraint “if $c(i - n - 1)\dots c(i - n + 2) = x_1\dots x_4$, then $c(i) \in \{\square, \delta(x_1\dots x_4)\}$ ” is inductive and separates C_I and c .

Therefore, it is sufficient to define an abstraction framework in which every constraint of the above form can be expressed. This is relatively straightforward. We set $\mathcal{A} := \square^*\Lambda^4\square^*\Lambda\square^*$. Given a constraint $A = \square^i x_1\dots x_4 \square^j x \square^k$, define $\mathcal{V}(A)$ as the set of all configurations c s.t. $c(i + 1)\dots c(i + 4) = x_1\dots x_4$ implies $c(i + j + 5) \in \{\square, x\}$. Clearly, \mathcal{V} is a regular relation which can be recognised by a transducer with 3 states.

► **Theorem 15** ([19]). *The abstract safety problem is PSPACE-hard, even for regular abstraction frameworks where the transducer for the interpretation has a constant number of states.*

From PSPACE-hardness to EXPSPACE-hardness

In order to prove EXPSPACE-hardness, we start with a machine \mathcal{M} of size n and run it on a tape with 2^n cells. However, if we proceed exactly as in the PSPACE-hardness proof, we encounter two obstacles: (1) The length of α_0 is 2^n , so our definitions of \mathcal{C} and C_I require automata of exponential size. (2) The transducer for the transition relation Δ needs to “count” to 2^n , as this is the distance between the corresponding symbols of α_i and α_{i+1} . Again, this requires an exponential number of states.

	00	000	# ⁰	q_0^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0
$\xrightarrow{\text{mark}(2,1)}$	01	000	# ¹	q_0^0	\square^1	\square^0	\square^1	\square^0	\square^1	\square^0	\square^1	\square^0	\square^1	\square^0
$\xrightarrow{\text{mark}(3,0)}$	01	100	# ¹	q_0^1	\square^1	\square^0	\square^1	\square^1	\square^1	\square^1	\square^1	\square^1	\square^1	\square^0
$\xrightarrow{\text{init}}$	00	000	# ⁰	q_0^0	\square^0	B^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0	\square^0
$\xrightarrow{\text{mark}(2,0)}$	10	000	# ⁰	q_0^1	\square^0	B^1	\square^0	\square^1	\square^0	\square^1	\square^0	\square^0	\square^0	\square^1
$\xrightarrow{\text{mark}(3,0)}$	10	100	# ⁰	q_0^1	\square^1	B^1	\square^1	\square^1	\square^0	\square^1	\square^1	\square^1	\square^1	\square^1
$\xrightarrow{\text{init}}$	00	000	# ⁰	q_0^0	\square^0	B^0	\square^0	\square^0	# ⁰	\square^0	\square^0	\square^0	\square^0	\square^0
$\xrightarrow{\dots}$	00	000	# ⁰	q_0^0	B^0	B^0	B^0	B^0	# ⁰	\square^0	\square^0	\square^0	\square^0	\square^0
$\xrightarrow{\dots}$	10	001	# ¹	q_0^1	B^0	B^1	B^1	B^1	# ¹	\square^1	\square^0	\square^1	\square^1	\square^1
$\xrightarrow{\text{write}}$	00	000	# ⁰	q_0^0	B^0	B^0	B^0	B^0	# ⁰	\square^0	q_1^1	\square^0	\square^0	\square^0
$\xrightarrow{\dots}$	01	010	# ¹	q_0^0	B^1	B^1	B^1	B^1	# ¹	\square^0	q_1^1	\square^1	\square^1	\square^1
$\xrightarrow{\text{write}}$	00	000	# ⁰	q_0^0	B^0	B^0	B^0	B^0	# ⁰	x^0	q_1^0	\square^0	\square^0	\square^0

■ **Figure 3** A sample run of the regular transition system described in Example 16. Here, $\text{mark}(x, y)$ means that the y -th bit of the prime number x is changed to 1, and thus every position not equivalent to $y \pmod{x}$ is unmarked. Note that the first position of the TM part (the one with #) is position 0. We write x^y instead of $\binom{y}{x}$. We highlight bits and symbols that were written to in pink (bits which are unmarked by the mark transition, but were already unmarked, are drawn in darker pink).

Obstacle (1) will be easy to overcome. Essentially, instead of starting the RTS with the entire initial configuration α_0 of \mathcal{M} already in place, we set $C_I := \# q_0 \square^*$ and modify the transitions of \mathcal{S} to also write out α_0 .

However, obstacle (2) poses a more fundamental problem. On its face, it is easy to construct an RTS that can count to 2^n by executing multiple transitions in sequence, e.g. by implementing a binary counter. However, we need to balance this with the needs of the abstraction framework: if the RTS is too sophisticated, our constraints can no longer capture its behaviour using only regular languages.

We now sketch an RTS \mathcal{S}' which extends the RTS \mathcal{S} from the PSPACE-hardness proof.

A two-phase system. In order to write the run of \mathcal{M} , the RTS \mathcal{S}' uses a “mark and write” approach. In a first phase, it executes n transitions to mark positions with distance m , where $m \geq 2^n$ is some fixed constant. Then, it nondeterministically guesses a marked position, reads and stores 4 symbols from that position, and moves to the next marker to write the symbol according to δ .

Let p_1, \dots, p_n be the first n prime numbers (i.e. $p_1 = 2, p_2 = 3$, etc.). Define $m := \prod_{j=1}^n p_j$ and $s := \sum_{j=1}^n p_j$. We have $m \geq 2^n$ and, by the Prime Number Theorem, $s \in O(n^2 \log n)$.

The configurations of \mathcal{S}' are of the form $w \binom{\mathbf{m}}{c}$, where $w \in \{0, 1\}^s$ stores the current state of the mark phase, $\mathbf{m} \in \{0, 1\}^*$ are the markers (0 means marked), and $c \in (\Lambda \cup \{\square\})^*$ is as for \mathcal{S} , with the reachable configurations being the prefixes of α with some symbols replaced by \square . We refer to $\binom{\mathbf{m}}{c}$ as the *TM part*.

The RTS has three kinds of transitions: $\Delta' := \Delta_{\text{mark}} \cup \Delta_{\text{write}} \cup \Delta_{\text{init}}$.

We solve this issue via \mathcal{V}_2 . For the constraint A_1 above there is going to be a constraint $A_2 \in \mathcal{A}_2$ s.t. the combination $\mathcal{V}_1(A_1) \cap \mathcal{V}_2(A_2)$ is inductive. Essentially, A_2 will ensure that it is impossible to write to position $i + m$ without reading from position i . (Note that for a particular constraint the value of i is fixed.)

Let $\mathcal{A}_2 := \{0, 1\}^s[0, n]^*$. Intuitively, a constraint $xy \in \mathcal{A}_2$ (where $|x| = s$) states: “if remainders for the first j primes have been chosen according to x , then exactly the positions k with $y(k) \geq j$ are marked, otherwise positions k with $y(k) = n$ are unmarked”, where j is the number of primes that have been chosen.

Again, the constraint A_1 is only concerned with one position i . Moreover, there is only one sequence of remainders r_1, \dots, r_n to choose for the Δ_{mark} transitions, s.t. position i is marked (i.e. $r_j \equiv i \pmod{p_j}$). So for each position k we can determine the index in the sequence of Δ_{mark} transitions at which position k will first become unmarked. Concretely, we have $y(k) := \min\{j \mid k \not\equiv i \pmod{p_j}\} - 1$.

This constraint is inductive and, crucially, the intersection of A_1 and A_2 is inductive as well. Essentially, every Δ_{mark} transition either continues the sequence r_1, \dots, r_n , and then the positions must be marked precisely according to y , or at some point a different remainder has been chosen, and the position i is unmarked and cannot be written to.

To summarise, constraint A_1 is sufficient to exclude any unsafe configuration and, in combination with A_2 , does so inductively. Therefore, if \mathcal{M} does not accept, then the RTS can be proven safe using the abstraction framework.

For the full proof, see the full version of the paper [16].

5 Learning regular sets of inductive constraints

Recall the algorithm for ABSTRACTSAFETY underlying Theorem 14. It computes an automaton recognising the set Ind of inductive constraints (Lemma 12); uses this automaton to compute a transducer recognising the potential reachability relation $PReach$ (Lemma 13); uses this transducer to compute an automaton recognising $PReach(C_I) \cap C_U$; and finally uses this automaton to check if $PReach(C_I) \cap C_U$ is empty (Theorem 14). The main practical problem of this approach is that, while the automaton for \overline{Ind} has polynomial size in the input, the automaton for Ind can be exponential, and, while the automaton for \overline{PReach} has polynomial size in Ind , the size of the automaton for $PReach$ can be exponential.

In practice one typically does not need all inductive constraints to prove safety. This can be illustrated even on the tiny RTS of Example 2.

► **Example 17.** Consider the RTS of the token passing system of Example 2, where $\Sigma = \{t, n\}$, and the second abstraction framework of Example 10, where $\Gamma = 2^\Sigma = \{\emptyset, \{t\}, \{n\}, \{t, n\}\}$. Recall that in this abstraction framework a constraint is an *exclusive disjunction* of literals, that is, a configuration satisfies the constraint if it satisfies *exactly one* of its literals. The minimal DFA recognising all non-trivial inductive constraints was shown on the left of Figure 2. The set of inductive constraints $\{t\}\{t\}^*$ is satisfied by the configurations n^*tn^* , and so the DFA on the right is already strong enough to prove any safety property.

We present a learning algorithm that computes automata recognising increasingly large sets $H \subseteq Ind$ of inductive constraints until either H is large enough to prove safety, or it becomes clear that even the whole set Ind is not large enough. More precisely, recall that, by definition, we have $PReach := \{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \forall A \in Ind: c \in \mathcal{V}(A) \rightarrow c' \in \mathcal{V}(A)\}$. Given a set $H \subseteq Ind$, define the relation $PReach_H$ exactly as $PReach$, just replacing Ind by H . Clearly, we have $PReach_H \supseteq PReach$ and $PReach_{Ind} = PReach$.

5.1 The learning algorithm

Let $\mathcal{S} = (\mathcal{C}, \Delta)$ and $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ be a regular transition system and a regular abstraction framework, respectively. Further, let C_I and C_U be regular sets of initial and unsafe configurations. The algorithm refines Angluin's algorithm L^* for learning a DFA for the full set Ind [7, 30]. Recall that Angluin's algorithm involves two agents, usually called Learner and Teacher. Learner sends Teacher membership and equivalence queries, which are answered by Teacher according to the following specification:

Membership Query:

- Input: a constraint $A \in \mathcal{A}$
- Output: \checkmark if $A \in Ind$, and \times otherwise.

Equivalence Query:

- Input: a DFA recognising a set $H \subseteq \mathcal{A}$.
- Output: \checkmark if $H = Ind$, otherwise a constraint $A \in (H \setminus Ind) \cup (Ind \setminus H)$.

Angluin's algorithm describes a strategy for Learner guaranteeing that Learner eventually learns the minimal DFA recognising Ind . The number of equivalence queries asked by Learner is at most the number of states of the DFA.

Answering the queries. We describe the algorithms used by Teacher to answer queries. For membership queries, Teacher constructs an NFA for $\overline{Ind} \cap \{A\}$ with $O(|A| \cdot n_\Delta \cdot n_V^2)$ states (see Lemma 12), and checks it for emptiness.

For equivalence queries, Teacher proceeds as follows :

1. Teacher first checks whether $H \setminus Ind \neq \emptyset$ holds by computing an NFA recognising $H \cap \overline{Ind}$ with $O(n_H \cdot n_\Delta \cdot n_V^2)$ states (see Lemma 12), and checking it for emptiness. If $H \setminus Ind$ is nonempty, then Teacher returns one of its elements.
2. Otherwise, Teacher constructs an automaton for $PReach_H(C_I) \cap C_U$ of size $O(2^{n_V} \cdot n_H)$ and checks it for emptiness. There are two cases:
 - a. If $PReach_H(C_I) \cap C_U = \emptyset$, then the system is safe; Teacher reports it and terminates. In this case, the learning algorithm is aborted without having learned a DFA for Ind , because it is no longer necessary.
 - b. Otherwise, Teacher chooses an element $(c, c') \in PReach_H \cap (C_I \times C_U)$, and searches for an inductive constraint A such that $c \in \mathcal{V}(A)$ and $c' \notin \mathcal{V}(A)$. We call this problem the *separability problem*, and analyze it further in Section 5.2.

5.2 The separability problem

The SEPARABILITY problem is formally defined as follows:

Given: a nondeterministic transducer recognising a regular relation $\Delta \subseteq \Sigma^* \times \Sigma^*$; a deterministic transducer recognising a regular interpretation \mathcal{V} over $\Gamma \times \Sigma$; and two configurations $c, c' \in \mathcal{C}$

Decide: is c' separable from c , i.e. does there exist $A \in Ind$ s.t. $c \in \mathcal{V}(A)$ and $c' \notin \mathcal{V}(A)$?

Contrary to ABSTRACTSAFETY, the complexity of SEPARABILITY is different for arbitrary transducers, and for length-preserving ones.

► **Theorem 18.** *SEPARABILITY is PSPACE-complete, even if Δ is length-preserving. If \mathcal{V} is length-preserving, then SEPARABILITY is NP-complete.*

Proof. See the full version of the paper [16]. ◀

■ **Table 1** Comparison of the sizes of the automata computed by the lazy and direct approaches. In each table, the first three columns contain the name of the RTS and the sizes of the automata for C_I and Δ . The fourth column (Pr.) indicates the checked property, where D, M, and O stand for “deadlock freedom”, “mutex” (at most one process in a given state), and “other” (custom properties of the particular RTS). The next two columns give the results for the lazy approach: sizes of the DFAs for H and $PReach_H$ (abbreviated as PR_H), and the next two the same results for the direct approach. The last column (Re.) indicates the result of the check: the property could be proved (\checkmark), could not (\times), or, in the case of multiple properties, how many of the properties were proved.

System	$ C_I $ $ \Delta $	Pr.	Lazy		Direct		Re.
			$ H $	$ PR_H $	$ Ind $	$ PR $	
Bakery	3 5	D M	1 1 4 3	9 8	\checkmark \checkmark		
Burns	1 6	D M	1 1 5 3	10 6	\checkmark \checkmark		
Dijkstra	2 17	D M	4 4 11 8	218 22	\checkmark \checkmark		
Dijkstra (ring)	2 12	D M	9 9 9 7	47 17	\checkmark \times		
Dining crypto.	2 8	D	23 18	86 19	$2/2$		
Herman	2 11	D O	3 2 1 2	8 7	\checkmark \checkmark		
Herman (linear)	2 3	D O	1 2 1 2	7 7	\times \checkmark		
Israeli-Jafon	3 10	D O	1 4 1 4	21 7	\checkmark \checkmark		
Token passing	2 3	O	4 4	9 7	\checkmark		
Lehmann-Rabin	1 7	D	5 6	29 13	\checkmark		
LR phil. 1	1 11	D	13 14	29 15	\times		
LR phil. 2	1 11	D	25 11	29 9	\checkmark		
Atomic phil.	1 8	D	13 9	22 20	\checkmark		
Mux array	2 4	D M	1 2 3 5	7 8	\checkmark \times		
Res. alloc.	1 5	D M	5 5 3 3	9 8	\checkmark \times		

System	$ C_I $ $ \Delta $	Pr.	Lazy		Direct		Re.
			$ H $	$ PR_H $	$ Ind $	$ PR $	
Berkeley	1 9	D O	1 1 4 4	12 9	\checkmark $2/3$		
Dragon	1 23	D O	1 1 15 7	37 11	\checkmark $6/7$		
Firefly	1 16	D O	1 1 4 3	12 7	\checkmark $0/4$		
Illinois	1 16	D O	1 1 4 3	18 14	\checkmark $0/2$		
MESI	1 7	D O	1 1 4 4	8 7	\checkmark $2/2$		
MOESI	1 7	D O	1 1 4 4	15 10	\checkmark $7/7$		
Synapse	1 5	D O	1 1 2 3	8 7	\checkmark $2/2$		

System	$ C_I $ $ \Delta $	Pr.	Lazy		Direct		Re.
			$ H $	$ PR_H $	$ Ind $	$ PR $	
Dijkstra (ring)	2 12	M	9 7			\checkmark	
LR phil. 1	1 11	D	34 11			\checkmark	
Mux array	2 4	M	5 3			\checkmark	
Res. alloc.	1 5	M	5 5			\checkmark	

Most applications of regular model checking to the verification of parameterised systems, and in particular all the examples studied in [19, 20], have length-preserving transition functions and length-preserving interpretations. For this reason, in our implementation we only consider this case, and leave an extension for future research. Since SEPARABILITY is NP-complete in the length-preserving case, it is natural to solve it by reduction to SAT. A brief description of the reduction is given in the full version of the paper [16].

5.3 Some experimental results

We have implemented the learning algorithm in a tool prototype, built on top of the libraries `automatalib` and `learnlib` [22] and the SAT solver `sat4j` [9]. We compare our learning approach with the one of [19], which constructs automata for Ind and $PReach$ using the regular abstraction framework of Example 7. In the rest of this section we call these two approaches the *lazy* and the *direct* approach, respectively. We use the same case studies as [19]. We compare the sizes of the DFA for the final hypothesis H and $PReach_H$ with the sizes of the DFA for Ind and $PReach$. The results are available at [33] and are shown in Table 1.

The left table in Table 1 shows results on RTSs modeling mutex and leader election algorithms, and academic examples, like various versions of the dining philosophers. The right top table shows results on models of cache-coherence protocols. Observe that Ind and

$PReach$ do not depend on the property, but H and $PReach_H$ do, because the algorithm can finish early. In this case, the sizes given in columns H and $PReach_H$ are the largest ones computed over all properties checked.

The main result is that the automata computed by our tool are significantly smaller than those for [19]. (Note that in all cases we compute minimal DFAs, and so the differences are not due to algorithms for the computation of automata.) Observe that in five cases the deadlock-freedom and the mutex properties could not be proved. In one case (deadlock-freedom of Herman (linear)) this is because the property does not hold. In the other four cases, the problem is that [19] uses only a specific regular abstraction framework, namely the one of Example 7. We can prove the property by refining the abstraction: we take the union of the “disjunctive” and the “exclusive disjunctive” abstractions of Example 10. The bottom-right table gives the results of these four cases.

Both tools take less than three seconds in 54 out of the 59 case studies in the left and top right tables. We do not report the exact times; the implementation of [19] uses MONA, while the experiments of this paper use `automatalib` and `learnlib`, and so small time differences may have any number of reasons. In the other five cases, the implementation of [19] still needs less than one second, while our implementation takes minutes (more than ten minutes in two cases). In these five cases the time performance is dominated by the SAT solver `sat4j`. We have not yet identified a pattern explaining why `sat4j` takes so much time, in particular the number and size of the formulas passed to it is similar to the other cases.

6 Conclusions

We have generalised the technique of [19, 20] for checking safety properties of RTS to arbitrary regular abstraction frameworks. We have shown that the abstract safety problem is EXPSPACE-complete, solving an open problem of [19, 20], by means of a complex reduction of independent interest. For particular abstraction frameworks the complexity can be better.

We have used automata learning to design a lazy algorithm that stops when the inductive constraints computed so far are enough to prove safety. Its combination with other learning techniques, as those proposed in [28, 31, 15, 32, 29], is a question for future research.

References

- 1 Parosh Aziz Abdulla. Regular model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):109–118, 2012.
- 2 Parosh Aziz Abdulla. Regular model checking: Evolution and perspectives. In *Model Checking, Synthesis, and Learning*, volume 13030 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2021.
- 3 Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. Parameterized verification through view abstraction. *Int. J. Softw. Tools Technol. Transf.*, 18(5):495–516, 2016.
- 4 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d’Orso. Regular model checking made simple and efficient. In *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2002.
- 5 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.
- 6 Parosh Aziz Abdulla, A. Prasad Sistla, and Muralidhar Talupur. Model checking parameterized systems. In *Handbook of Model Checking*, pages 685–725. Springer, 2018.
- 7 Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

- 8 Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.
- 9 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *J. Satisf. Boolean Model. Comput.*, 7(2-3):59–6, 2010.
- 10 Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large (extended abstract). In *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2003.
- 11 Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomás Vojnar. Abstract regular (tree) model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):167–191, 2012.
- 12 Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.
- 13 Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.
- 14 Ahmed Bouajjani and Tayssir Touili. Widening techniques for regular tree model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):145–165, 2012.
- 15 Yu-Fang Chen, Chih-Duo Hong, Anthony W. Lin, and Philipp Rümmer. Learning to prove safety over parameterised concurrent systems. In *FMCAD*, pages 76–83. IEEE, 2017.
- 16 Philipp Czerner, Javier Esparza, Valentin Krasotin, and Christoph Welzel-Mohr. Computing inductive invariants of regular abstraction frameworks. *CoRR*, abs/2404.10752, 2024. doi: 10.48550/arXiv.2404.10752.
- 17 Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2001.
- 18 Javier Esparza and Michael Blondin. *Automata Theory – An Algorithmic Approach*. MIT Press, 2023.
- 19 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Regular model checking upside-down: An invariant-based approach. In *CONCUR*, volume 243 of *LIPICs*, pages 23:1–23:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 20 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Regular model checking upside-down: An invariant-based approach. *CoRR*, abs/2205.03060, version 2, 2022.
- 21 Chih-Duo Hong and Anthony W. Lin. Regular abstractions for array systems. *Proc. ACM Program. Lang.*, 8(POPL):638–666, 2024.
- 22 Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib - A framework for active automata learning. In *CAV (1)*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer, 2015.
- 23 Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2000.
- 24 Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256(1-2):93–112, 2001.
- 25 Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.
- 26 Axel Legay. Extrapolating (omega-)regular model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):119–143, 2012.
- 27 Antoine Miné. The octagon abstract domain. *High. Order Symb. Comput.*, 19(1):31–100, 2006.
- 28 Daniel Neider. *Applications of automata learning in verification and synthesis*. PhD thesis, RWTH Aachen University, 2014.
- 29 Daniel Neider and Nils Jansen. Regular model checking using solver technologies and automata learning. In *NASA Formal Methods*, volume 7871 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2013.
- 30 Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer, 2011.

19:18 Computing Inductive Invariants of Regular Abstraction Frameworks

- 31 Abhay Vardhan. *Learning To Verify Systems*. PhD thesis, University of Illinois, 2006.
- 32 Abhay Vardhan, Koushik Sen, Mahesh Viswanathan, and Gul Agha. Learning to verify safety properties. In *ICFEM*, volume 3308 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2004.
- 33 Christoph Welzel-Mohr, Valentin Krasotin, Philipp Czerner, and Javier Esparza. dodo, February 2024. doi:10.5281/zenodo.12734991.

Behavioural Metrics: Compositionality of the Kantorovich Lifting and an Application to Up-To Techniques

Keri D’Angelo ✉ 
Cornell University, USA

Sebastian Gurke ✉ 
Universität Duisburg-Essen, Germany

Johanna Maria Kirss ✉
University of Copenhagen, Denmark

Barbara König ✉ 
Universität Duisburg-Essen, Germany

Matina Najafi ✉
Amirkabir University of Technology, Iran

Wojciech Różowski ✉ 
University College London, UK

Paul Wild ✉ 
FAU Erlangen-Nürnberg, Germany

Abstract

Behavioural distances of transition systems modelled via coalgebras for endofunctors generalize traditional notions of behavioural equivalence to a quantitative setting, in which states are equipped with a measure of how (dis)similar they are. Endowing transition systems with such distances essentially relies on the ability to lift functors describing the one-step behavior of the transition systems to the category of pseudometric spaces. We consider the category theoretic generalization of the Kantorovich lifting from transportation theory to the case of lifting functors to quantale-valued relations, which subsumes equivalences, preorders and (directed) metrics. We use tools from fibred category theory, which allow one to see the Kantorovich lifting as arising from an appropriate fibred adjunction. Our main contributions are compositionality results for the Kantorovich lifting, where we show that the lifting of a composed functor coincides with the composition of the liftings. In addition, we describe how to lift distributive laws in the case where one of the functors is polynomial (with finite coproducts). These results are essential ingredients for adapting up-to-techniques to the case of quantale-valued behavioural distances. Up-to techniques are a well-known coinductive technique for efficiently showing lower bounds for behavioural distances. We illustrate the results of our paper in two case studies.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases behavioural metrics, coalgebra, Galois connections, quantales, Kantorovich lifting, up-to techniques

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.20

Related Version Proofs and extra material can be found in the full version.

Full Version: <https://arxiv.org/abs/2404.19632> [10]

Funding *Sebastian Gurke, Barbara König and Paul Wild:* were supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 434050016 (SpeQt).

Johanna Maria Kirss: was supported by NSF grant DMS-2216871.

Wojciech Różowski: partially supported by ERC grant Autoprobe (no. 101002697).

Acknowledgements We would like to thank Avi Cramer for helpful discussions and the organizers of the ACT Adjoint School for enabling us to meet and write this paper together!



© Keri D’Angelo, Sebastian Gurke, Johanna Maria Kirss, Barbara König, Matina Najafi, Wojciech Różowski, and Paul Wild;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 20; pp. 20:1–20:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In concurrency theory, behavioural equivalences are a fundamental concept: they explain whether two states exhibit the same behaviour and there are efficient techniques for checking behavioural equivalence [25]. More recently, this idea has been generalized to behavioural metrics that measure the behavioural distance of two states [28, 11, 12]. This is in particular interesting for quantitative systems where the behaviour of two states might be similar but not identical.

There are two dimensions along which notions of behavioural distances for (quantitative) transition systems can be generalized: first, instead of only considering metrics that return a real-valued distance, one can use an arbitrary quantale, yielding the notion of quantale-valued relations or conformances that subsume equivalences, preorders and (directed) metrics. Second, one can abstract from the branching type of the transition system under consideration, using a functor specifying various forms of branching (non-deterministic, probabilistic, weighted, etc.). This leads us to a coalgebraic framework [24] that provides several techniques for studying and analyzing systems and has also been adapted to behavioural metrics [2]. A coalgebra is a function $c: X \rightarrow FX$ where X is a set of states and F is a (set) functor.

For defining (and computing) behavioural conformances in coalgebraic generality, a fundamental notion is the lifting of a functor F to \overline{F} , acting on conformances. In the case when F is a distribution functor and the conformances are metrics, there is a well-known way of obtaining \overline{F} through results from transportation theory using either Kantorovich or Wasserstein liftings, which are known to coincide through the so-called Kantorovich-Rubinstein duality [29]. Recent work [2] generalized these two approaches to lifting functors to the category of pseudometric spaces, as well as to more general quantale-valued relations [3, 5]. It turns out that at this level of generality, the analogue of the Kantorovich-Rubinstein duality does not hold in general anymore. In both the Kantorovich and Wasserstein approaches, given a set X and a conformance d (preorder, equivalence, metric, ...) on X , the lifted functor \overline{F} canonically determines a conformance on FX , based on (a set of) evaluation maps. Intuitively, these maps provide a way of testing the one-step behaviour of the system and generalize the calculation of the expected value taking place in the definitions of the Kantorovich/Wasserstein liftings. Combining the lifting with a subsequent reindexing with the coalgebra c results in a function whose (greatest) fixpoint is the desired behavioural conformance. In this paper we focus on directed conformances such as preorders or directed metrics (also called hemimetrics).

The aim of this paper is twofold: we consider the so-called Kantorovich lifting of functors [2] that – as opposed to the Wasserstein lifting [5] – offers some extra flexibility, since it allows the use of a set of evaluation maps and places fewer restrictions on both the functor and these predicate liftings. We study compositionality results for the Kantorovich lifting, answering the question under which conditions the lifting is compositional in the sense that $\overline{FG} = \overline{F}\overline{G}$. This compositionality result is then an essential ingredient in adapting up-to techniques to the setting of behavioural metrics based on the Kantorovich lifting, inspired by the results of [5], which were developed for the Wasserstein lifting. Up-to techniques are coinductive techniques that allow small witnesses for lower bounds of behavioural distances by exploiting an algebraic structure on the state space.

We first set up a framework based on Galois connections and fibred adjunctions, extending [4]. This sets the stage for the definition of the Kantorovich lifting based on this adjunction. We next answer the question of compositionality positively for the case where F is polynomial with finite coproducts, and show several negative results for combinations of powerset and distribution functor.

The positive compositionality result for the case where F is polynomial with finite coproducts opens the door to developing up-to techniques for trace-like behavioural conformances that are computed on determinized transition systems, or – more generally – determinized coalgebras. More concretely, we consider coalgebras of type $c: X \rightarrow FTX$ where F is a finite coproduct polynomial functor, providing the explicit branching type of the coalgebra, and T is a monad, describing the implicit branching. For instance, in the case of a non-deterministic automaton we would use $FX = 2 \times X^A$ and $T = \mathcal{P}$ (powerset functor). There is a well-known determinization construction [17, 26] that transforms such a coalgebra into $c^\#: TX \rightarrow FTX$ via a distributive law $\zeta: TF \Rightarrow FT$. This yields a determinized system $c^\#$ acting on the state space TX , which has an algebraic structure given by the multiplication of the monad. A behavioural conformance is then computed on TX and passed back to X using the unit of the monad.

As TX might be very large (e.g., in the case of $T = \mathcal{P}$) or even infinite (e.g., in the case of $T = \mathcal{D}$, distribution functor), it can be hard to compute conformances for $c^\#$. However, the algebraic structure on TX can be fruitfully employed using up-to techniques [23, 6] that allow to consider post-fixpoints up to the algebraic structure, making it much easier to display a witness proving a (lower) bound on the distance of two states. The validity of the up-to technique rests on a compatibility property that is ensured whenever the distributive law ζ can be lifted, i.e., if it is non-expansive wrt. the lifted conformances. We show that this holds, where an essential step in the proof relies on the compositionality results.

In this sense, we complement the up-to techniques in [5] that provide a similar result for the Wasserstein lifting. This enables us to use the up-to technique for Kantorovich liftings, which are more versatile and allow more control over the distance values, in particular in the presence of products and coproducts. Indeed, as Wasserstein liftings are based on couplings, which in general need not exist [2], they often produce trivial distance values. We will see later in the paper that several of our case studies can only be treated appropriately in the Kantorovich case. Furthermore, we can show the lifting of the distributive law for an entire class of functors (polynomial functors with finite coproducts), while [5] contained generic results for a different class of canonical liftings. In the non-canonical case it was necessary to prove a complex condition ensuring compositionality in [5].

We apply our technique to several examples, such as trace metrics for probabilistic automata and trace semantics for systems with exceptions. We give concrete instances where up-to techniques help and show how witnesses yielding upper bounds can be reduced in size or even become finite (as opposed to infinite).

2 Preliminaries

We begin by recalling some relevant definitions and fix some notation.

As outlined in the introduction, we use quantales as the domain for behavioural conformances. A *quantale* is a complete lattice $(\mathcal{V}, \sqsubseteq)$ that is equipped with a commutative monoid structure $(\mathcal{V}, \otimes, k)$ (where k is the unit of \otimes) such that \otimes is *join-continuous*, that is, $a \otimes \bigsqcup b_i = \bigsqcup a \otimes b_i$ for each $a \in \mathcal{V}$ and each family b_i in \mathcal{V} , where \bigsqcup denotes least upper bounds. Join-continuity of \otimes implies that the operation $a \otimes -$ has a right adjoint, which we denote by $d_{\mathcal{V}}(a, -)$. This means that we have $a \otimes b \sqsubseteq c \iff b \sqsubseteq d_{\mathcal{V}}(a, c)$ for all $a, b, c \in \mathcal{V}$. The operation $d_{\mathcal{V}}$ is called the *residuation* or *internal hom* of the quantale.

We work with three main examples of quantales. The first is the *Boolean quantale* 2_{\sqcap} , consisting of two elements $\perp \sqsubseteq \top$ and with binary meet \sqcap as the monoid structure. In this quantale, the unit k is \top , and residuation is Boolean implication: $d_{\mathcal{V}}(a, b) = a \rightarrow b = \neg a \sqcup b$.

The second main example is the *unit interval quantale* $[0, 1]_{\oplus}$, where the underlying lattice is the unit interval $[0, 1]$ under the reversed order (that is, $\sqsubseteq = \geq$), and with *truncated addition* $a \oplus b = \min(a + b, 1)$ as the monoid structure. In this case, the unit of \oplus is 0, and its residuation is *truncated subtraction*, which is given by $d_{\mathcal{V}}(a, b) = b \ominus a = \max(b - a, 0)$. The third main example is the quantale $[0, \infty]_+$ of *extended positive reals*, with structure analogous to $[0, 1]_{\oplus}$, i.e. with reversed lattice order and using the extended addition (with ∞) as the monoid operation.

► **Remark 1.** As many of our examples use the real-valued quantales $[0, 1]_{\oplus}$ and $[0, \infty]_+$, where the order is reversed, we reserve the use of the symbols \geq and \leq for the usual order in the reals, and instead use \sqsubseteq and \sqsupseteq when working with general quantales. Similarly, we use \sqcup and \sqcap for joins and meets in the quantalic order, but switch to \inf and \sup when working in $[0, 1]_{\oplus}$ or $[0, \infty]_+$.

We consider several types of conformances based on a quantale \mathcal{V} . First, given a set X , we may consider \mathcal{V} -valued endorelations on X , that is, maps of type $d: X \times X \rightarrow \mathcal{V}$. We call these structures \mathcal{V} -graphs [20], and write $\mathcal{V}\text{-Graph}_X$ for the set of \mathcal{V} -graphs with underlying set X . Each set $\mathcal{V}\text{-Graph}_X$ is a complete lattice where both the order and joins are pointwise, that is $d \sqsubseteq d'$ if $d(x, y) \sqsubseteq d'(x, y)$ for all $x, y \in X$. Given two \mathcal{V} -graphs $d_X \in \mathcal{V}\text{-Graph}_X$ and $d_Y \in \mathcal{V}\text{-Graph}_Y$ we say that a map $f: X \rightarrow Y$ is *non-expansive* or a \mathcal{V} -functor if $d_X \sqsubseteq d_Y \circ (f \times f)$ in $\mathcal{V}\text{-Graph}_X$ and in this case we write $f: (X, d_X) \rightarrow (Y, d_Y)$. \mathcal{V} -graphs and non-expansive maps form a category $\mathcal{V}\text{-Graph}$.

► **Remark 2.** In some parts of the literature, the category $\mathcal{V}\text{-Graph}$ is denoted by $\mathcal{V}\text{-Rel}$ instead [5]. We opt for $\mathcal{V}\text{-Graph}$ as in [20], as $\mathcal{V}\text{-Rel}$ more often denotes the category with sets as objects and \mathcal{V} -valued relations between them as morphisms [13].

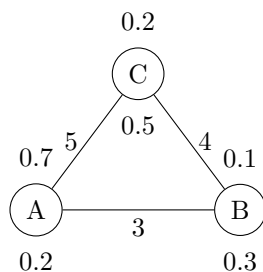
Second, within $\mathcal{V}\text{-Graph}$ we consider the subcategory consisting of those \mathcal{V} -graphs that satisfy additional axioms, corresponding to a generalized notion of a metric space, or, equivalently, (small) categories enriched over \mathcal{V} [20]. A \mathcal{V} -category is an object of $\mathcal{V}\text{-Graph}_X$ for some set X where we additionally have $k \sqsubseteq d(x, x)$ and $d(x, y) \otimes d(y, z) \sqsubseteq d(x, z)$ for all $x, y, z \in X$. Instantiated to the quantales 2_{\sqcap} and $[0, 1]_{\oplus}$, \mathcal{V} -categories correspond precisely to *preorders* and *1-bounded hemimetric spaces*, respectively. The quantale \mathcal{V} itself becomes a \mathcal{V} -category when equipped with the residuation $d_{\mathcal{V}}$. The class of all \mathcal{V} -categories is denoted by $\mathcal{V}\text{-Cat}$, and the set of \mathcal{V} -categories based on a set X is denoted by $\mathcal{V}\text{-Cat}_X$.

In this paper, we use a coalgebraic framework. Recall that a *coalgebra* is a pair (X, c) , where X is the *state space* and $c: X \rightarrow FX$ is the transition structure parametric on an endofunctor $F: \text{Set} \rightarrow \text{Set}$. We also utilize two specific functors for (counter)examples below. The *powerset functor* is defined as $\mathcal{P}(X) = \{U \mid U \subseteq X\}$ on sets and $\mathcal{P}(f)(U) = \{f(x) \mid x \in U\}$ on functions. The *countably supported distribution functor* is defined as $\mathcal{D}(X) = \{p: X \rightarrow [0, 1] \mid \sum_{x \in X} p(x) = 1, \text{supp}(p) \text{ is countable}\}$ on sets, where $\text{supp}(p) = \{x \mid p(x) \neq 0\}$, and as $\mathcal{D}(f)(p)(y) = \sum\{p(x) \mid x \in X, f(x) = y\}$ on functions.

Given $f_i: X \rightarrow Y_i$, $i \in \{1, 2\}$, we denote by $\langle g_1, g_2 \rangle: X \rightarrow Y_1 \times Y_2$ the mediating morphism of the product, namely $\langle g_1, g_2 \rangle(x) = (g_1(x), g_2(x))$. Given $g_i: X_i \rightarrow Y$, $i \in \{1, 2\}$, $[g_1, g_2]: X_1 + X_2 \rightarrow Y$ is the mediating morphism of the coproduct, namely $[g_1, g_2](x) = g_i(x)$ if $x \in X_i$.

3 Motivation from Transportation Theory

In this section, we give a brief description of the original Kantorovich distance on probability distributions, before we introduce its category theoretic generalization. Motivated by the transportation problem [29], the Kantorovich distance on probability distributions aims to maximize the transport by finding the optimal flow of commodities that satisfies demand from supplies and minimizes the flow cost. In fact, it is based on the dual version of this problem that asks for the optimal price function. For the sake of the example, consider a metric space defined on a three element set $X = \{A, B, C\}$ with a distance function $d: X \times X \rightarrow [0, \infty]$, such that $d(A, A) = d(B, B) = d(C, C) = 0$, $d(A, B) = d(B, A) = 3$, $d(A, C) = d(C, A) = 5$ and $d(B, C) = d(C, B) = 4$. Based on this distance we now want to define a distance on probability distributions on the set X , which is a function $d^\dagger: \mathcal{D}(X) \times \mathcal{D}(X) \rightarrow [0, \infty]$. As a concrete example, let us take the distributions P and Q , such that $P(A) = 0.7$, $P(B) = 0.1$ and $P(C) = 0.2$, while $Q(A) = 0.2$, $Q(B) = 0.3$ and $Q(C) = 0.5$.



In order to define their distance, we can interpret the three elements A, B, C as places where a certain product is produced and consumed (imagine the places are maple syrup farms, each with an adjacent café where one can eat the pancakes with the aforementioned syrup). The geographical distance between the maple syrup farms is given by the distance function d , while the above distributions model the supply (Q) and demand (P) of the product in proportion to the total supply or demand. We assume that the total value of supply is the same as the total value of demand. As the owners of the farms, we are interested in transporting the product in a way to avoid excess supply and meet all demands. We can deal with this issue in two ways: do the transport on our own or find a logistics firm which will do it for us. The Kantorovich lifting relies on the latter perspective. We assume that for each place it sets up a price for the logistics company at which it will buy a unit of our product (at farms with overproduction) or sell it (at cafés with excess demand). This is equivalent to giving a function $f: X \rightarrow [0, \infty]$. We require that prices are *competitive*, that is for all $x, y \in X$, we have that $|f(x) - f(y)| \leq d(x, y)$, which is equivalent to saying that f is a non-expansive function from d into the Euclidean distance d_e on $[0, \infty]$. Given a pricing f , the profit made by the transportation company is given by $c_f = \sum_{x \in X} f(x)P(x) - \sum_{x \in X} f(x)Q(x) = \sum_{x \in X} f(x)(P(x) - Q(x))$. Because the transportation company wants to make the most profit, it is aiming to pick a pricing f maximizing the formula given above. Combining all the moving parts together we are left with formula $d^\dagger(P, Q) = \max\{\sum_{x \in X} f(x)(P(x) - Q(x)) \mid f: (X, d_X) \rightarrow ([0, \infty], d_e)\}$ defining the Kantorovich distance between probability distributions P and Q .

It is not straightforward to see, but in this example an optimal pricing function is $f(A) = 0$, $f(B) = 3$, $f(C) = 5$, which can be easily seen to be non-expansive and yields a distance of 2.1.

More abstractly, the formula above can be dissected into three pieces:

1. Taking all pricing plans f , which are non-expansive with respect to the Euclidean distance on $[0, \infty]$.
2. Evaluating each of the pricing plans, by calculating the expected value of f given a distribution on the set X .
3. Picking a pricing plan which maximizes the difference between the expected values.

In the following, we will concentrate on the directed case, where distance functions can be asymmetric and d_e is the directed Euclidean distance, that is, $d_e(x, y) = y \ominus x$.

In the category-theoretic generalization [2, 3] the calculation of the expected value (step 2) is replaced with (the set of) evaluation functions, intuitively scoring or testing the observable behaviour. At the same time, the steps of taking all non-expansive plans for a given metric and of generating a metric that maximizes the difference between expected values (steps 1 and 3) generalize to the setting of quantales and their residuation. In the next section, we show that the generalizations of those two steps form a fibred adjunction.

4 Setting Up a Fibred Adjunction

One of the key aspects of this paper is equipping sets of states of coalgebras with an extra structure of conformances (in particular preorders or hemimetrics). The very idea of “extra structure” can be phrased formally through the lenses of fibrations and fibred category theory, extending the ideas of [4]. In particular, we show that those results can be strengthened to the setting of fibred adjunctions.

The category \mathcal{V} -SPred. The adjunction-based framework from [4], besides working with \mathcal{V} -graphs, makes use of sets of \mathcal{V} -valued predicates, i.e., maps of the form $p: X \rightarrow \mathcal{V}$. We will use $\mathcal{V}\text{-SPred}_X$ to denote the collection of sets of \mathcal{V} -valued predicates over some set X . A morphism between sets $S \subseteq \mathcal{V}^X$ and $T \subseteq \mathcal{V}^Y$ is a function $f: X \rightarrow Y$ satisfying $f^\bullet(T) := \{q \circ f \mid q \in T\} \subseteq S$, where f^\bullet describes reindexing. We obtain a category $\mathcal{V}\text{-SPred}$ with objects being pairs $(X, S \subseteq \mathcal{V}^X)$ and arrows are defined as above.

Grothendieck completion. It turns out that both $\mathcal{V}\text{-Graph}/\mathcal{V}\text{-Cat}$ and $\mathcal{V}\text{-SPred}$ can be viewed as fibred categories [20, 5]. We here only consider fibred categories arising from the Grothendieck construction, which can be viewed equivalently as a special kind of split indexed categories, that in our case are functors $A: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Pos}$. Intuitively, such functors assign to each set a poset of “extra structure” and take functions $f: X \rightarrow Y$ to monotone maps canonically reindexing the “extra structure” on set Y by f .

The *Grothendieck completion* [14] of a functor $A: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Pos}$ is a category denoted $\int A$, whose objects are pairs (X, i) where $X \in \mathbf{Set}$ and $i \in A(X)$. The arrows $(X, i) \rightarrow (Y, j)$ are maps $f: X \rightarrow Y$ such that $i \sqsubseteq A(f)(j)$, where \sqsubseteq is the partial order on $A(X)$. The corresponding fibration is given by the forgetful functor $U: \int A \rightarrow \mathbf{Set}$ taking each pair (X, i) to its underlying set X . The fibre of each set X corresponds to the poset $A(X)$.

$\mathcal{V}\text{-Graph}/\mathcal{V}\text{-Cat}$ and $\mathcal{V}\text{-SPred}$ as Grothendieck completions. The category $\mathcal{V}\text{-Graph}$ arises as the Grothendieck completion of the functor $\Phi: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Pos}$ that takes each set X to $\Phi(X) = (\mathcal{V}\text{-Graph}_X, \sqsubseteq)$, the lattice of \mathcal{V} -valued relations equipped with the pointwise order \sqsubseteq . Given a function $f: X \rightarrow Y$, we define $\Phi(f) = f^*$ by reindexing, where $f^*(d_Y) = d_Y \circ (f \times f)$. Analogously for $\mathcal{V}\text{-Cat}$.

Similarly, to obtain \mathcal{V} -SPred, we define a functor $\Psi : \text{Set}^{\text{op}} \rightarrow \text{Pos}$ that maps each set X to $\Psi(X) = (\mathcal{V}\text{-SPred}_X, \supseteq)$, the lattice of collections of \mathcal{V} -valued predicates on X ordered by reverse inclusion. Furthermore $\Psi(f) = f^\bullet$.

Galois connection on the fibres. In the adjunction-based framework from [4], the Kantorovich lifting of a functor is phrased through the means of a contravariant Galois connection between the fibres of $\mathcal{V}\text{-Cat}$ and $\mathcal{V}\text{-SPred}$ and we here generalize from $\mathcal{V}\text{-Cat}$ to $\mathcal{V}\text{-Graph}$. For each set X , we define a map $\alpha_X : \mathcal{V}\text{-SPred}_X \rightarrow \mathcal{V}\text{-Graph}_X$ given by:

$$\alpha_X(S)(x_1, x_2) = \prod_{f \in S} d_{\mathcal{V}}(f(x_1), f(x_2)) \quad (S \subseteq \mathcal{V}^X)$$

Intuitively, α_X takes a collection S of \mathcal{V} -valued predicates on X and generates the greatest conformance on X that turns all predicates into non-expansive maps. For the other part of the Galois connection, we have a map $\gamma_X : \mathcal{V}\text{-Graph}_X \rightarrow \mathcal{V}\text{-SPred}_X$ defined by the following:

$$\gamma_X(d_X) = \{f : X \rightarrow \mathcal{V} \mid d_{\mathcal{V}} \circ (f \times f) \sqsupseteq d_X\}$$

Given a conformance $d_X : X \times X \rightarrow \mathcal{V}$, γ_X generates a set of \mathcal{V} -valued predicates on X which are non-expansive maps from d_X to the residuation distance. As mentioned before, we can instantiate the previous result [4, Theorem 7] and obtain the following:

► **Theorem 3** ([4]). *Let X be an arbitrary set, $d_X : X \times X \rightarrow \mathcal{V}$ a \mathcal{V} -graph and $S \subseteq \mathcal{V}^X$ a collection of \mathcal{V} -valued predicates. Then α_X and γ_X are both antitone (in \subseteq, \sqsubseteq) and form a contravariant Galois connection:*

$$d_X \sqsubseteq \alpha_X(S) \iff S \subseteq \gamma_X(d_X).$$

► **Example 4.** In the setting of Section 3, γ corresponds to Step 1 and α to Step 3.

Fibred Adjunction. Theorem 3 is a “local” property that only holds fiberwise. However, it turns out that we can argue something stronger, namely that we have a fibred adjunction situation. This is a “global” property, as fibred functors additionally preserve the notion of reindexing. Note that every natural transformation between functors of type $\text{Set}^{\text{op}} \rightarrow \text{Pos}$ (a so-called morphism of split indexed categories) [14, Definition 1.10.5] induces a fibred functor between the corresponding Grothendieck completions.

One can quite easily verify that α is natural on $\mathcal{V}\text{-Graph}$, while γ is only laxly natural on $\mathcal{V}\text{-Graph}$ and natural on $\mathcal{V}\text{-Cat}$. For the latter result, we rely on a quantalic version of the McShane-Whitney extension theorem [22, 30], mentioned also in [20] for the real-valued case.

► **Theorem 5.** *Let $d_X \in \mathcal{V}\text{-Cat}_X$ and $d_Y \in \mathcal{V}\text{-Cat}_Y$ be elements of $\mathcal{V}\text{-Cat}$. If $i : (Y, d_Y) \rightarrow (X, d_X)$ is an isometry, then for any non-expansive map $f : (Y, d_Y) \rightarrow (\mathcal{V}, d_{\mathcal{V}})$ there exists a non-expansive map $g : (X, d_X) \rightarrow (\mathcal{V}, d_{\mathcal{V}})$ such that $f = g \circ i$.*

► **Proposition 6.** *We have that $\alpha : \Psi \Rightarrow \Phi$ is natural and $\gamma : \Phi \Rightarrow \Psi$ is laxly natural, that is for all functions $f : X \rightarrow Y$ and all \mathcal{V} -valued relations d_Y on the set Y we have that $(f^\bullet \circ \gamma_Y)(d_Y) \subseteq (\gamma_X \circ f^*)(d_Y)$. When restricted to $\mathcal{V}\text{-Cat}$, $\gamma : \Phi \Rightarrow \Psi$ is natural.*

Note that we can safely make this restriction, while still keeping α to be well-defined, as its image always lies within $\mathcal{V}\text{-Cat}$.

► **Proposition 7.** *For all sets X and $S \subseteq \mathcal{V}^X$, $\alpha_X(S)$ is a \mathcal{V} -category, i.e., an object of $\mathcal{V}\text{-Cat}$. The co-closure $\alpha_X \circ \gamma_X$ is the identity when restricted to $\mathcal{V}\text{-Cat}$. Combined, this implies that for $d \in \mathcal{V}\text{-Graph}_X$, $\alpha_X(\gamma_X(d))$ is the metric closure of d , i.e. the least element of $\mathcal{V}\text{-Cat}_X$ above d .*

Because of the Grothendieck construction ([14, Theorem 1.10.7]), α and γ respectively correspond to fibred functors $\alpha: \mathcal{V}\text{-SPred} \rightarrow \mathcal{V}\text{-Cat}$ and $\gamma: \mathcal{V}\text{-Cat} \rightarrow \mathcal{V}\text{-SPred}$. Both functors keep morphisms unchanged and act on objects by respectively applying the appropriate components of α and γ . Now, we can state the strengthened version of Theorem 3, by showing that α and γ form a fibred adjunction. Note that due to the choice of the orderings, γ becomes the left and α the right adjoint (cf. [19]).

► **Theorem 8.** *There is an adjunction $\gamma \dashv \alpha$.*

5 The Coalgebraic Kantorovich Lifting

5.1 Definition of the Coalgebraic Kantorovich Lifting

The coalgebraic Kantorovich lifting [2] (originally defined for the real-valued case and for a single evaluation map) – extended to codensity liftings in [18] – is parametric in a set of *evaluation functions* for a set functor F . Evaluation functions are maps of type $F\mathcal{V} \rightarrow \mathcal{V}$ (generalizing the expected value computation in the traditional Kantorovich lifting) and as such can be used to lift \mathcal{V} -valued predicates on a set X to \mathcal{V} -valued predicates on the set FX . More precisely, given an evaluation function $ev: F\mathcal{V} \rightarrow \mathcal{V}$ and a predicate $f: X \rightarrow \mathcal{V}$ we obtain the predicate $ev \circ Ff: FX \rightarrow \mathcal{V}$. This operation extends to sets of evaluation functions and sets of \mathcal{V} -valued predicates, where a set Λ^F of evaluation functions for F induces the fibred functor $\Lambda^F: \mathcal{V}\text{-SPred} \rightarrow \mathcal{V}\text{-SPred}$, defined on $S \subseteq \mathcal{V}^X$ as follows:

$$\Lambda_X^F(S) = \{ev \circ Ff \mid ev \in \Lambda^F, f \in S\} \subseteq \mathcal{V}^{FX}.$$

The Kantorovich lifting can now be restated via the fibred adjunction introduced previously. Given F and Λ^F as above, we can define its Kantorovich lifting K_{Λ^F} as follows:

$$K_{\Lambda^F} = \alpha F \circ \Lambda^F \circ \gamma$$

or more concretely, for an object d_X of $\mathcal{V}\text{-Graph}$ and $s, t \in FX$:

$$K_{\Lambda^F}(d_X)(s, t) = \prod_{ev \in \Lambda^F} \prod_{f \in \gamma_X(d_X)} d_{\mathcal{V}}(ev(Ff(s)), ev(Ff(t))).$$

If the set Λ^F is clear from the context we sometimes write \bar{F} instead of K_{Λ^F} .

► **Lemma 9.** *The Kantorovich lifting of a functor $F: \text{Set} \rightarrow \text{Set}$ is a functor $\bar{F} = K_{\Lambda^F}: \mathcal{V}\text{-Graph} \rightarrow \mathcal{V}\text{-Graph}$, and fibred when restricted to $\mathcal{V}\text{-Cat}$.*

► **Remark 10.** Instantiating the construction above with the distribution functor \mathcal{D} and a single evaluation function \mathbb{E} taking the expected values yields the usual Kantorovich lifting, while in the case of powerset functor \mathcal{P} and a single evaluation function sup , one obtains the (directed) Hausdorff metric. Despite those two instantiations corresponding to well-known constructions, there is no well-defined notion of *canonical lifting* and there are often different possibilities for a given functor. For example, the usual symmetric Hausdorff distance arises by additionally considering the dual evaluation function inf [32]. We will later also see that constant factors admit more than one choice of evaluation functions.

5.2 Compositionality of the Kantorovich Lifting

When an endofunctor is given as the composition of two or more individual set functors, it is natural to ask under which conditions its Kantorovich lifting is also the composition of Kantorovich liftings of the respective functors. Specifically, our aim in this section is to

identify situations where this composition happens already at the level of the underlying sets of evaluation maps. If ev_F is an evaluation function for F and ev_G is an evaluation function for G , then an evaluation function for FG is given by $ev_F * ev_G := ev_F \circ F ev_G$. Extending this to sets of evaluation functions, we put $\Lambda^F * \Lambda^G = \{ev_F * ev_G \mid ev_F \in \Lambda^F, ev_G \in \Lambda^G\}$ for sets Λ^F and Λ^G of evaluation functions for functors F and G , respectively.

► **Definition 11 (Compositionality).** Given two functors F and G and sets Λ^F and Λ^G of evaluation functions, we say that we have *compositionality* if $K_{\Lambda^F} \circ K_{\Lambda^G} = K_{\Lambda^F * \Lambda^G}$.

Expanding definitions, compositionality amounts to showing that

$$K_{\Lambda^F} \circ K_{\Lambda^G} = \alpha FG \circ \Lambda^F \circ \gamma G \circ \alpha G \circ \Lambda^G \circ \gamma = \alpha FG \circ \Lambda^F \circ \Lambda^G \circ \gamma = K_{\Lambda^F * \Lambda^G}. \quad (1)$$

One inequality (“ \sqsubseteq ”) always holds: As α and γ form a Galois connection [4], we have $\text{id}_{\mathcal{V}\text{-SPred}_X} \subseteq \gamma GX \circ \alpha GX$, and thus we may use antitonicity of α to deduce “ \sqsubseteq ” in (1). Baldan et al. [2, Lemma 7.5] prove this for the special case of pseudometric liftings.

The other inequality, “ \supseteq ”, does not hold in general, and requires more work. Still, one notices that a sufficient condition is that $\Lambda^F \circ \gamma \circ \alpha \subseteq \gamma F \circ \alpha F \circ \Lambda^F$:

$$\begin{aligned} K_{\Lambda^F} \circ K_{\Lambda^G} &= \alpha FG \circ \Lambda^F \circ \gamma G \circ \alpha G \circ \Lambda^G \circ \gamma \\ &\supseteq \alpha FG \circ \gamma FG \circ \alpha FG \circ \Lambda^F \circ \Lambda^G \circ \gamma = \alpha FG \circ \Lambda^{FG} \circ \gamma = K_{\Lambda^F * \Lambda^G}, \end{aligned}$$

using that $\alpha \circ \gamma \circ \alpha = \alpha$ for every Galois connection. Note that it is enough to prove the sufficient condition on non-empty sets, since γ always yields a non-empty set.

Before discussing the problem of systematically constructing sets of evaluation functions such that the sufficient condition (lax commutativity of Λ^F with the closure induced by the Galois connection) holds, we consider a few examples where compositionality fails:

► **Example 12.** Consider the powerset functor \mathcal{P} with the predicate lifting sup ($\Lambda^{\mathcal{P}} = \{\text{sup}\}$), and the discrete distribution functor \mathcal{D} with the predicate lifting \mathbb{E} that takes expected values ($\Lambda^{\mathcal{D}} = \{\mathbb{E}\}$). With just these predicate liftings, compositionality fails for all four combinations $\mathcal{P}\mathcal{P}$, $\mathcal{P}\mathcal{D}$, $\mathcal{D}\mathcal{P}$, $\mathcal{D}\mathcal{D}$. We show this for the case of $\mathcal{P}\mathcal{D}$ and discuss the others in [10]. Let X be the two-element set $\{x, y\}$, equipped with the discrete metric d (that is, $d(x, y) = d(y, x) = 1$), so that in particular all maps $g: X \rightarrow [0, 1]$ are non-expansive. We also consider the non-expansive function $f_{\mathcal{D}}: (\mathcal{D}X, K_{\{\mathbb{E}\}}d) \rightarrow ([0, 1], d_{[0,1]_{\oplus}})$ given by $f_{\mathcal{D}}(p \cdot x + (1-p) \cdot y) = \min(p, 1-p)$. Put $U = \{1 \cdot x, 1 \cdot y\}$ and $V = \{1 \cdot x, 1/2 \cdot x + 1/2 \cdot y, 1 \cdot y\}$. Then $\text{sup } f_{\mathcal{D}}[U] = \max(0, 0) = 0$ and $\text{sup } f_{\mathcal{D}}[V] = \max(0, 1/2, 0) = 1/2$, so that $K_{\{\text{sup}\}}(K_{\{\mathbb{E}\}}d)(U, V) \geq 1/2$. For every $g: X \rightarrow [0, 1]$ one finds that

$$(\text{sup} * \mathbb{E})(g)(U) = \max(g(x), g(y)) = \max(g(x), g^{(x)+g(y)/2}, g(y)) = (\text{sup} * \mathbb{E})(g)(V),$$

implying that $K_{\{\text{sup} * \mathbb{E}\}}d(U, V) = 0$.

5.3 Finite Coproduct Polynomial Functors

We now assume that the first functor (F with the lifting $\bar{F} = K_{\Lambda^F}$) is in fact a polynomial functor (with finite coproducts) and we show that in this case compositionality holds automatically for certain sets of predefined evaluation maps. This will later allow us to use compositionality to define up-to techniques for large classes of coalgebras that are based on such functors.

Consider the set of polynomial functors (with finite coproducts) given by

$$F ::= C_B \mid \text{Id} \mid \prod_{i \in I} F_i \mid F_1 + F_2$$

20:10 Behavioural Metrics: Compositionality of the Kantorovich Lifting

where C_B is the constant functor mapping to some set B and Id is the identity functor. We support products over arbitrary index sets, but we restrict to finitary coproducts for simplicity.

For such polynomial functors we can obtain compositionality in a structured manner, by constructing suitable sets of predicate liftings alongside with the functors themselves. We recursively define a set Λ^F of evaluation functions for each polynomial functor F as follows:

constant functors: $F = C_B$. Here we choose Λ^F to be any set of maps of type $B \rightarrow \mathcal{V}$. (For instance, when $B = \mathcal{V}$ we can put $\Lambda^F = \{\text{id}_{\mathcal{V}}\}$.)

identity functor: $F = \text{Id}$. We put $\Lambda^F = \{\text{id}_{\mathcal{V}}\}$.

product functors: $F = \prod_{i \in I} F_i$. Put $\Lambda^F = \{ev_i \circ \pi'_i \mid i \in I, ev_i \in \Lambda^{F_i}\}$ where $\pi'_i: \prod_{i \in I} F_i \mathcal{V} \rightarrow F_i \mathcal{V}$ are the projections.

coproduct functors: $F = F_1 + F_2$. We put $\Lambda^F = \{[ev_1, \top] \mid ev_1 \in \Lambda^{F_1}\} \cup \{[\perp, ev_2] \mid ev_2 \in \Lambda^{F_2}\} \cup \{[\perp, \top]\}$, where \top and \perp denote constant maps into \mathcal{V} .

► **Remark 13.** We note that the construction for coproduct functors is associative, that is, for functors F_1, F_2 and F_3 the sets $\Lambda^{(F_1+F_2)+F_3}$ and $\Lambda^{F_1+(F_2+F_3)}$ coincide up to isomorphism.

Note also that the exponentiation $F^A X = (FX)^A$ is special case of the product where $I = A$ and $F_i = F$ for all $i \in I$.

► **Remark 14.** Throughout the paper, we restrict our attention to finite coproducts for the sake of simplicity, but we would like to note that our construction could be generalized to infinite sets. In general, since our lifting for the coproduct will be based on prioritization, we need to compare the sets in order of preference, i.e. have the additional structure of a well-order. This immediately works for countable sets.

The choice of evaluation maps above induces the following liftings, leading to the natural expected distances in the directed case.

► **Proposition 15.** *Given a polynomial functor F and a set of evaluation maps Λ^F as defined above, the corresponding lifting $\bar{F} = K_{\Lambda^F}$ is defined as follows on objects of $\mathcal{V}\text{-Graph}$: $\bar{F}(d_X) = d_X^F: FX \times FX \rightarrow \mathcal{V}$ where*

constant functors: $d_X^F: B \times B \rightarrow \mathcal{V}$, $d_X^F(b, c) = \prod_{ev \in \Lambda^F} d_{\mathcal{V}}(ev(b), ev(c))$.

identity functor: $d_X^F: X \times X \rightarrow \mathcal{V}$ with $d_X^F = \alpha_X(\gamma_X(d))$.

product functors: $d_X^F: \prod_{i \in I} F_i X \times \prod_{i \in I} F_i X \rightarrow \mathcal{V}$, $d_X^F(s, t) = \prod_{i \in I} d_X^{F_i}(\pi_i(s), \pi_i(t))$ where $\pi_i: \prod_{i \in I} F_i X \rightarrow F_i X$ are the projections.

coproduct functors: $d_X^F: (F_1 X + F_2 X) \times (F_1 X + F_2 X) \rightarrow \mathcal{V}$, where

$$d_X^F(s, t) = \begin{cases} d^{F_i}(s, t) & \text{if } s, t \in F_i X \text{ for } i \in \{1, 2\} \\ \top & \text{if } s \in F_1 X, t \in F_2 X \\ \perp & \text{if } s \in F_2 X, t \in F_1 X \end{cases}$$

Under this choice of evaluation functions we can show the following, which implies compositionality (cf. Section 5.2):

► **Proposition 16.** *For every polynomial functor F and the corresponding set Λ^F of evaluation maps (as above) we have $\Lambda^F \circ \gamma \circ \alpha \subseteq \gamma F \circ \alpha F \circ \Lambda^F$ on non-empty sets of predicates.*

Using the arguments of Section 5.2, we infer:

► **Corollary 17.** *Let F and G be functors, and Λ^F and Λ^G be sets of predicate liftings for them. If F and λ^F are as in Proposition 16, then $K_{\Lambda^F} \circ K_{\Lambda^G} = K_{\Lambda^F * \Lambda^G}$.*

► **Example 18.** We consider a running example specifying standard directed trace metrics for probabilistic automata as introduced in [17]. We take the polynomial functor $F = [0, 1] \times _{}^A$ (“machine functor”), monad $T = \mathcal{D}$ and quantale $\mathcal{V} = [0, 1]_{\oplus}$. Furthermore we use expectation (\mathbb{E}) as evaluation map for T and as evaluation maps for the functor F we take ev_* mapping to the first component and ev_a (for each $a \in A$) with $ev_a(r, g) = g(a)$. These evaluation maps are of the type described in this section and hence we have compositionality.

Note that this example is not directly realizable in the Wasserstein approach [5]: the issue with the Wasserstein lifting is that whenever no coupling of two elements exists, the distance is automatically the bottom element in the quantale. This can be seen for the functor F where $(r_1, x), (r_2, x) \in FX$ have no coupling whenever $r_1 \neq r_2$. Hence it is harder to parameterize and would not work here.

Using a set of evaluation maps Λ^F as opposed to a single evaluation map gives us additional flexibility.

6 Application: Up-To Techniques

We now adapt results from [5] on up-to techniques from Wasserstein to Kantorovich liftings. In particular, we instantiate the fibrational approach to coinductive proof techniques from [6] that allows to prove lower bounds for greatest fixpoints, using post-fixpoints up-to as witnesses. As shown in the running example and in Section 7 this can greatly help to reduce the size of such witnesses, even allowing finitary witnesses which would be infinite otherwise.

6.1 Introduction to Up-To Techniques

We first recall the notion of a bialgebra [15], a coalgebra with a compatible algebra structure.

► **Definition 19.** Consider two functors F, T and a natural transformation $\zeta : TF \Rightarrow FT$. An F - T -bialgebra for ζ is a tuple (Y, a, c) such that $a : TY \rightarrow Y$ is a T -algebra and $c : Y \rightarrow FY$ is an F -coalgebra so that the diagram below commutes.

$$\begin{array}{ccccc} TY & \xrightarrow{a} & Y & \xrightarrow{c} & FY \\ \downarrow Tc & & & & \uparrow Fa \\ TFY & \xrightarrow{\zeta_Y} & & & FTY \end{array}$$

In order to construct such bialgebras, distributive laws exchanging functors and monads are helpful.

► **Definition 20.** A *distributive law* or *EM-law* of a monad $T : \mathbf{C} \rightarrow \mathbf{C}$ with unit $\eta : \text{Id} \Rightarrow T$ and multiplication $\mu : TT \Rightarrow T$ over a functor $F : \mathbf{C} \rightarrow \mathbf{C}$ is a natural transformation $\zeta : TF \Rightarrow FT$ such that the following diagrams commute:

$$\begin{array}{ccc} FX & & T^2FX \\ \eta_{FX} \downarrow & \searrow F\eta_X & \downarrow \mu_{FX} \\ TFx & \xrightarrow{\zeta_X} & FTX \\ & & \downarrow F\mu_X \\ & & FT^2X \end{array} \quad \begin{array}{ccc} T^2FX & \xrightarrow{T\zeta_X} & TF^2X \\ \downarrow \mu_{FX} & & \downarrow F\mu_X \\ TFx & \xrightarrow{\zeta_X} & FTX \end{array}$$

Whenever T is a monad and ζ is an EM-law, then an F - T -bialgebra can be obtained by determining a coalgebra $c : X \rightarrow FTX$. More concretely, we obtain $c^\# : Y \rightarrow FY$ where $Y = TX$ and $c^\# = F\mu_X \circ \zeta_{TX} \circ Tc$. The algebra map is $a = \mu_X : TY \rightarrow Y$.

20:12 Behavioural Metrics: Compositionality of the Kantorovich Lifting

We now assume a bialgebra (Y, a, c) and Kantorovich liftings $\bar{T} = K_{\Lambda T}$, $\bar{F} = K_{\Lambda F}$ of T, F . Based on this we can define a *behaviour function* beh via

$$\mathcal{V}\text{-Graph}_Y \xrightarrow{\bar{F}} \mathcal{V}\text{-Graph}_{FY} \xrightarrow{c^*} \mathcal{V}\text{-Graph}_Y$$

Remember that c^* denotes reindexing via c . The greatest fixpoint of beh corresponds to a behavioural conformance (e.g., behavioural equivalence or bisimulation metric).

► **Example 21.** We continue with Example 18. We use the standard distributive law $\zeta: TF \Rightarrow FT$ given by the following components where $\mathbb{E}_\mu \pi_1 = \mathbb{E}(\mathcal{D}\pi_1(\mu))$:

$$\begin{aligned} \zeta_X: \mathcal{D}([0, 1] \times X^A) &\rightarrow [0, 1] \times \mathcal{D}X^A \\ \zeta_X(\mu) &= (\mathbb{E}_\mu \pi_1, a \mapsto \mathcal{D}(\text{eval}_a \circ \pi_2)(\mu)) \end{aligned}$$

where $\text{eval}_a(f) = f(a)$. Given an Eilenberg-Moore coalgebra $c: X \rightarrow FTX$ (more concretely: $c: X \rightarrow [0, 1] \times \mathcal{D}X^A$) and its determinization $c^\#: TX \rightarrow FTX$, the behavioural distance on TX arises as the greatest fixpoint (in the quantale order) of the map $\text{beh} = (c^\#)^* \circ \bar{F}$ defined above.

By unravelling the fixpoint equation one can see that it coincides with the directed trace metric on probability distributions that is defined as follows: for each state $x \in X$ let $tr_x: A^* \rightarrow [0, 1]$ be a map that assigns to each word (trace) $w \in A^*$ the expected payoff for this word when read from x , where the payoff of a state x' is $\pi_1(c(x'))$. Then

$$\nu\text{beh}(p, q) = \sup_{w \in A^*} \left(\sum_{x \in X} tr_x(w) \cdot q(x) \ominus \sum_{x \in X} tr_x(w) \cdot p(x) \right)$$

If p, q are Dirac distributions δ_x, δ_y , we have: $\nu\text{beh}(\delta_x, \delta_y) = \sup_{w \in A^*} (tr_y(w) \ominus tr_x(w))$.

One can typically avoid computing the full fixpoint νbeh when checking the behavioural distance of two states; this is facilitated through the use of an *up-to function* u defined via

$$\mathcal{V}\text{-Graph}_Y \xrightarrow{\bar{T}} \mathcal{V}\text{-Graph}_{TY} \xrightarrow{\Sigma_a} \mathcal{V}\text{-Graph}_Y$$

where $\Sigma_f: \mathcal{V}\text{-Graph}_X \rightarrow \mathcal{V}\text{-Graph}_Y$ is defined as $\Sigma_f(d)(y_1, y_2) = \bigsqcup_{f(x_i)=y_i} d(x_1, x_2)$ for $f: X \rightarrow Y$ (direct image).

Both functions (beh, u) are monotone functions on a complete lattice. Hence we can use the Knaster-Tarski theorem [27] and the theory of up-to techniques [23]. In particular, given a monotone function $f: L \rightarrow L$ over a complete lattice (L, \sqsubseteq) , we have the guarantee that $\ell \sqsubseteq f(\ell)$ for $\ell \in L$ guarantees $\ell \sqsubseteq \nu f$, i.e., a post-fixpoint of f is always a lower bound for the greatest fixpoint νf , an essential proof rule in coinductive reasoning. Even more widely applicable are proof rules based on up-to functions. An up-to function is a monotone function $u: L \rightarrow L$ that is f -compatible (i.e., $u \circ f \sqsubseteq f \circ u$). Then we can infer that $\ell \sqsubseteq f(u(\ell))$ (i.e., ℓ is a post-fixpoint up-to u) implies $\ell \sqsubseteq \nu f$ (ℓ is a lower bound for the greatest fixpoint). Typically u is extensive ($\ell \sqsubseteq u(\ell)$) and hence it is “easier” to find a post-fixpoint up-to rather than a post-fixpoint.

From [6] we obtain the following result that ensures compatibility:

► **Proposition 22** ([6]). *Whenever the EM-law $\zeta: TF \Rightarrow FT$ lifts to $\zeta: \bar{T}\bar{F} \Rightarrow \bar{F}\bar{T}$, we have that $u \circ \text{beh} \sqsubseteq \text{beh} \circ u$ (for u, beh as defined above), i.e., u is beh-compatible.*

Hence, we can deduce that every post-fixpoint up-to witnesses a lower bound of the greatest fixpoint. More concretely: $d_Y \sqsubseteq \text{beh}(u(d_Y))$ implies $d_Y \sqsubseteq \nu\text{beh}$ (where $d_Y \in \mathcal{V}\text{-Graph}_Y$) (coinduction up-to proof principle).

6.2 Lifting Distributive Laws

To use the proof technique laid out in the previous section, we have to show that ζ lifts accordingly. We start by defining distributive laws and lifting them to \mathcal{V} -Graph.

Let F be a polynomial functor (cf. Section 5.3) and (T, μ, η) a monad over \mathbf{Set} . Following [16, Exercise 5.4.4], EM-laws $\zeta : TF \Rightarrow FT$ can then be constructed inductively over the structure of F , i.e., for F being an identity, constant, product and coproduct functor. In the coproduct case we extend [16] by weakening the requirement that T preserves coproducts.

In the following, we inductively construct an EM-law $\zeta : TF \Rightarrow FT$ and first lift it to $\zeta : \overline{TF} \Rightarrow \overline{FT}$ (and then to $\zeta : \overline{T}\overline{F} \Rightarrow \overline{F}\overline{T}$). For the evaluation maps we assume that Λ^F is defined as in Section 5.3 and that $\Lambda^T = \{ev_T\}$, where $ev_T : T\mathcal{V} \rightarrow \mathcal{V}$ is a T -algebra.

constant functors: For $F = C_B$ we have that $TFX = TB$ and $FTX = B$, and so define the EM-law as $\zeta : TC_B \Rightarrow C_B$, where the (unique) component $\zeta_X : TB \rightarrow B$ is an arbitrary T -algebra on B . (From now on, we assume that evaluation maps $ev : B \rightarrow \mathcal{V}$ for constant functors are T -algebra homomorphisms between $\zeta : TB \rightarrow B$ and $ev_T : T\mathcal{V} \rightarrow \mathcal{V}$.)

identity functor: For $F = \text{Id}$, we let the EM-law be the identity map $\text{id} : T \Rightarrow T$.

product functors: For $F = \prod_{i \in I} F_i$, assuming we have distributive laws $\zeta^i : TF_i \Rightarrow F_i T$, the EM-law is

$$\langle \zeta^i \circ T\pi_i \rangle : T \prod_{i \in I} F_i \Rightarrow \prod_{i \in I} F_i T.$$

coproduct functors: For $F = F_1 + F_2$, assume that we have distributive laws $\zeta^i : TF_i \Rightarrow F_i T$ and a natural transformation $g : T((-) + (-)) \Rightarrow T + T$ between bifunctors. The EM-law is given by

$$(\zeta^1 + \zeta^2) \circ g_{F_1, F_2} : T(F_1 + F_2) \Rightarrow TF_1 + TF_2 \Rightarrow F_1 T + F_2 T$$

► **Definition 23.** Let T be a monad and let $g : T((-) + (-)) \Rightarrow T + T$ be a natural transformation as above. We say that g is *compatible with the unit* η of the monad if for all sets Y_1, Y_2 the left diagram below commutes. Analogously, g is *compatible with the multiplication* μ of the monad if the right diagram commutes for all sets Y_1, Y_2 .

$$\begin{array}{ccc} & Y_1 + Y_2 & \\ \eta_{Y_1+Y_2} \swarrow & & \searrow \eta_{Y_1+Y_2} \\ T(Y_1 + Y_2) & \xrightarrow{g_{Y_1, Y_2}} & TY_1 + TY_2 \end{array} \qquad \begin{array}{ccc} TT(Y_1 + Y_2) & \xrightarrow{\mu_{Y_1+Y_2}} & T(Y_1 + Y_2) \\ Tg_{Y_1, Y_2} \downarrow & & \downarrow g_{Y_1, Y_2} \\ T(TY_1 + TY_2) & \xrightarrow{g_{TY_1, TY_2}} & TTY_1 + TTY_2 \xrightarrow{\mu_{Y_1+Y_2}} TY_1 + TY_2 \end{array}$$

► **Proposition 24.** Assume that the natural transformation g is compatible with unit and multiplication of T . Then the transformation ζ as defined above is an EM-law of T over F .

In order to lift natural transformations (respectively distributive laws), we will use the following result:

► **Proposition 25.** Let F, G be functors on \mathbf{Set} and let the sets of evaluation maps of F and G be denoted by Λ^F and Λ^G . Let $\zeta : F \Rightarrow G$ be a natural transformation. If

$$\Lambda^G \circ \zeta_{\mathcal{V}} := \{ev_G \circ \zeta_{\mathcal{V}} \mid ev_G \in \Lambda^G\} \subseteq \Lambda^F, \quad (2)$$

then ζ lifts to $\zeta : \overline{F} \Rightarrow \overline{G}$ in \mathcal{V} -Graph, where $\overline{F} = K_{\Lambda^F}$, $\overline{G} = K_{\Lambda^G}$.

We can show that the inclusion (2) (even equality) holds under some conditions.

► **Definition 26.** Let $g : T((-) + (-)) \Rightarrow T + T$ be a natural transformation as introduced above and let $ev_T : T\mathcal{V} \rightarrow \mathcal{V}$ be the evaluation map of the monad. We say that g is *well-behaved* wrt. ev_T if the following diagrams commute for $f_i : X_i \rightarrow \mathcal{V}$, where \perp, \top are constant maps of the appropriate type.

$$\begin{array}{ccc} T(X_1 + X_2) \xrightarrow{T[f_1, \top X_2]} T\mathcal{V} & T(X_1 + X_2) \xrightarrow{T[\perp X_1, f_2]} T\mathcal{V} & T(X_1 + X_2) \xrightarrow{T[\perp X_1, \top X_2]} T\mathcal{V} \\ \downarrow g_{X_1, X_2} & \downarrow g_{X_1, X_2} & \downarrow g_{X_1, X_2} \\ TX_1 + TX_2 \xrightarrow{[ev_T \circ T f_1, \top TX_2]} \mathcal{V} & TX_1 + TX_2 \xrightarrow{[\perp TX_1, ev_T \circ T f_2]} \mathcal{V} & TX_1 + TX_2 \xrightarrow{[\perp TX_1, \top TX_2]} \mathcal{V} \end{array} \quad \begin{array}{c} \downarrow ev_T \\ \downarrow ev_T \\ \downarrow ev_T \end{array}$$

► **Lemma 27.** Let F be a polynomial functor and T a monad with $\Lambda^T = \{ev_T\}$.

For distributive laws ζ as described above where the component g is well-behaved wrt. ev_T and evaluation maps as defined in Section 5.3, we have that

$$(\Lambda^F * \Lambda^T) \circ \zeta_{\mathcal{V}} = \Lambda^T * \Lambda^F.$$

Then, when we have a coalgebra of the form $Y \rightarrow FTY$ for F polynomial and T a monad as above, and we determinize it to get a coalgebra $X \rightarrow FX$ for $X = TY$, we obtain a bialgebra with the algebra structure given by the monad multiplication $\mu_Y : TX \rightarrow X$. The EM-law obtained then also forms a distributive law for the bialgebra. By Proposition 25 and Lemma 27 we know that the distributive law ζ lifts to \mathcal{V} -Graph, i.e., $\zeta : \overline{FT} \Rightarrow \overline{T\overline{F}}$ where $\overline{FT} = K_{\Lambda^F * \Lambda^T}$ and $\overline{T\overline{F}} = K_{\Lambda^T * \Lambda^F}$.

We now show that natural transformations g as required above do exist for the powerset and subdistribution monad for suitable quantales. Note that they are “asymmetric” and prioritize one of the two sets over the other.

► **Proposition 28.** Let $T = \mathcal{P}$ be the powerset monad with evaluation map $ev_T = \sup$ for $\mathcal{V} = [0, 1]_{\oplus}$. Then g_{X_1, X_2} below is a natural transformation that is compatible with unit and multiplication of T and is well-behaved.

$$g_{X_1, X_2} : \mathcal{P}(X_1 + X_2) \rightarrow \mathcal{P}X_1 + \mathcal{P}X_2 \quad g_{X_1, X_2}(X') = \begin{cases} X' \cap X_1 & \text{if } X' \cap X_1 \neq \emptyset \\ X' & \text{otherwise} \end{cases}$$

► **Proposition 29.** Let $T = \mathcal{S}$ be the subdistribution monad where $\mathcal{S}(X) = \{p : X \rightarrow [0, 1] \mid \sum_{x \in X} p(x) \leq 1\}$. Assume that its evaluation map is $ev_T = \mathbb{E}$ for the quantale $\mathcal{V} = [0, \infty]_+$ (where we assume that $p \cdot \infty = \infty$ if $p > 0$ and 0 otherwise). Then g_{X_1, X_2} below is a natural transformation that is compatible with unit and multiplication of T and is well-behaved.

$$g_{X_1, X_2} : \mathcal{S}(X_1 + X_2) \rightarrow \mathcal{S}X_1 + \mathcal{S}X_2 \quad g_{X_1, X_2}(p) = \begin{cases} p|_{X_1} & \text{if } \text{supp}(p) \cap X_1 \neq \emptyset \\ p|_{X_2} & \text{otherwise} \end{cases}$$

It is left to show that the EM-law $\zeta : FT \Rightarrow TF$ lifts to $\zeta : \overline{FT} \Rightarrow \overline{T\overline{F}}$, where $\overline{F} = K_{\Lambda^F}$, $\overline{T} = K_{\Lambda^T}$ where $\Lambda^T = \{ev_T\}$, and where the evaluation maps for F are obtained as described earlier. We namely prove that its components are all non-expansive, i.e., \mathcal{V} -Graph-morphisms, commutativity is already known. Using the previous results we obtain:

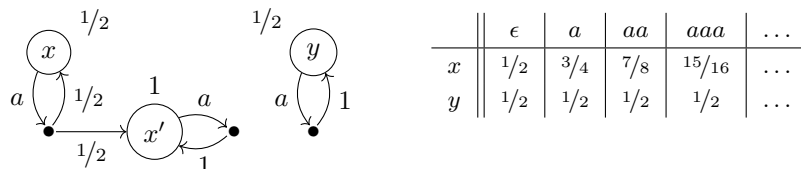
$$\overline{F\overline{T}} \stackrel{(1)}{\sqsubseteq} \overline{T\overline{F}} \stackrel{(2)}{\Rightarrow} \overline{F\overline{T}} \stackrel{(3)}{\cong} \overline{F\overline{T}}$$

(1) follows from the inequality in Section 5.2. This implies that the identity $\text{id}_{T\overline{F}X} : \overline{T\overline{F}}X \rightarrow \overline{T\overline{F}}X$ is non-expansive (a \mathcal{V} -Graph-morphism), resulting in the natural transformation $\overline{T\overline{F}} \stackrel{\text{id}}{\Rightarrow} \overline{T\overline{F}}$.

- (2) is implied by the results of this section (Lemma 27, Proposition 25)
 (3) is guaranteed by the fact that $\overline{FT} = \overline{F}T$ if F is polynomial (and we have suitable evaluation maps), hence compositionality holds (Proposition 16)

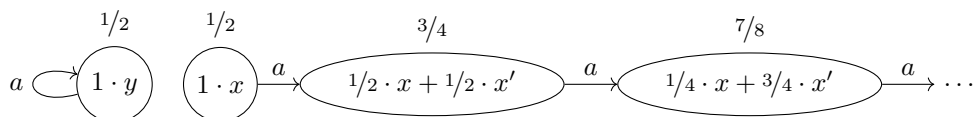
► **Example 30.** We continue Example 21 and first observe that the distributive law given there is obtained by the inductive construction given above.

For this concrete example fix the label set as a singleton: $A = \{a\}$. Consider the coalgebra with states $X = \{x, x', y\}$ drawn below on the left. The payoff is written next to each state.



Here, the trace map has the values for states x, y given in the table above on the right, leading to the behavioural distance $\nu\text{beh}(\delta_x, \delta_y) = 1/2$ (the supremum of all differences).

Determinizing the probabilistic automaton above leads to an automaton with infinite state space, even if we only consider the reachable states (which are probability distributions):



Our aim is now to define a witness distance of type $d: TX \times TX \rightarrow [0, 1]$ that is a post-fixpoint up-to in the quantale order and a pre-fixpoint for the order on the reals ($d \geq \text{beh}(u(d))$). From this we can infer that $\nu\text{beh} \leq d$, obtaining an upper bound. We set $d(1 \cdot x, 1 \cdot y) = 1/2$, $d(1 \cdot x', 1 \cdot y) = 1/2$ and $d(p, q) = 1$ for all other pairs of probability distributions p, q . We now show that d is a pre-fixpoint of beh in the order on the reals, i.e., our aim is to prove for all p, q that $d(p, q) \geq \text{beh}(u(d))(p, q)$. This is obvious for the cases where $d(p, q) = 1$, hence only two cases remain:

- If $p = 1 \cdot x, q = 1 \cdot y$, we have:

$$\begin{aligned}
 \text{beh}(u(d))(1 \cdot x, 1 \cdot y) &= \max\{u(d)(1/2 \cdot x + 1/2 \cdot x', y), |1/2 - 1/2|\} \\
 &= u(d)(1/2 \cdot x + 1/2 \cdot x', y) \\
 &\leq 1/2 \cdot d(1 \cdot x, 1 \cdot y) + 1/2 \cdot d(1 \cdot x', 1 \cdot y) \\
 &= 1/2 \cdot 1/2 + 1/2 \cdot 1/2 = 1/2 = d(1 \cdot x, 1 \cdot y)
 \end{aligned}$$

- The case $p = 1 \cdot x', q = 1 \cdot y$ can be shown analogously.

We are using the following inequalities: (i) $u(d)(p, q) \leq d(p, q)$ (follows from the definition of u); (ii) $u(d)(r_1 \cdot p_1 + r_2 \cdot p_2, r_1 \cdot q_1 + r_2 \cdot q_2) \leq r_1 \cdot d(p_1, q_1) + r_2 \cdot d(p_2, q_2)$ (metric congruence, see [10] for more details.). This concludes the argument.

Note that here up-to techniques in fact allow us to consider finitary witnesses for bounds for behavioural distances, even when the determinized coalgebra has an infinite state space.

7 Case Study: Transition Systems with Exceptions

In addition to the running example treated in the paper we now consider a second case study on transition systems with exceptions that helps to concretely show upper bounds (lower bounds in the quantalic order) for behavioural distances via appropriate witnesses.

20:16 Behavioural Metrics: Compositionality of the Kantorovich Lifting

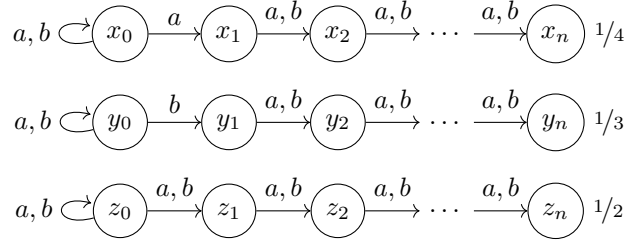
We consider a case study involving the coproduct, in particular the polynomial functor $F = [0, 1] + (-)^A$ and the monad $T = \mathcal{P}$ with evaluation map sup . In a coalgebra of type $c: X \rightarrow FTX$, a state can either perform transitions or terminate with some output value taken from the interval $[0, 1]$; in applications this value could for example be considered as the severity of the error encountered upon terminating a computation. Hence the (directed) distance of two states x, y can intuitively be interpreted as measuring how much worse the errors reached from a state y are compared to the errors from x .

Note that a state can also terminate without an exception by transitioning to the empty set. Due to the asymmetry in the distributive law for the coproduct, a state $X_0 \in \mathcal{P}(X)$ in the determinized automaton $c^\#$ will throw an exception as long as one of the elements $x \in X_0$ throws an exception ($c[X_0] \cap [0, 1] \neq \emptyset$). In this case, $c^\#(X_0) = \text{sup}(c[X_0] \cap [0, 1])$ and X_0 performs a transition if all states in X_0 do so in the original coalgebra.

The behavioural distance on TX obtained as the greatest fixpoint (in the quantale order) of beh can be characterized as follows: for a set of states $X_0 \subseteq X$ and a word $w \in A^*$ let $ec(X_0, w)$ be the length of the least prefix which causes an exception when starting in X_0 (undefined if there are no exceptions) and let $E(X_0, w) \subseteq [0, 1]$ be the set of exception values reached by that prefix. We define a distance $d_w^E: \mathcal{P}X \times \mathcal{P}X \rightarrow [0, 1]$ as $d_w^E(X_1, X_2) = 0$ if $ec(X_2, w)$ is undefined, $d_w^E(X_1, X_2) = 1$ if $ec(X_1, w)$ is undefined and $ec(X_2, w)$ defined. In the case where both are defined we set $d_w^E(X_1, X_2) = \text{sup} E(X_2, w) \ominus \text{sup} E(X_1, w)$ if $ec(X_1, w) = ec(X_2, w)$, $d_w^E(X_1, X_2) = 1$ if $ec(X_1, w) > ec(X_2, w)$ and $d_w^E(X_1, X_2) = 0$ if $ec(X_1, w) < ec(X_2, w)$. Then it can be shown that for $X_1, X_2 \subseteq X$:

$$\nu\text{beh}(X_1, X_2) = \text{sup}_{w \in A^*} d_w^E(X_1, X_2)$$

As a concrete example we take the label set $A = \{a, b\}$ and the coalgebra c given below, which is inspired by a similar example in [7]:



Here, the outputs of the final states are $c(x_n) = 1/4$, $c(y_n) = 1/3$ and $c(z_n) = 1/2$, as indicated. It holds that $\nu\text{beh}(\{x_0, y_0\}, \{z_0\}) = 1/4$. Intuitively this is true, since the largest distance is achieved if we follow a path from x_0 where a is the n -last letter, yielding exception value $1/4$, while the same path results in the value $1/2$ from z_0 , hence we obtain distance $1/2 \ominus 1/4 = 1/4$.

Note that the determinization of the transition system above is of exponential size and the same holds for a representation of a post-fixpoint for witnessing an upper bound for behavioural distance. We construct a \mathcal{V} -valued relation d of linear size witnessing that $\nu\text{beh}(\{x_0, y_0\}, \{z_0\}) \leq 1/4$ via up-to techniques. To this end let d be defined by

$$d(\{x_0, y_0\}, \{z_0\}) = 1/4 \quad d(\{x_i\}, \{z_i\}) = 1/4 \quad d(\{y_i\}, \{z_i\}) = 1/6$$

and distance 1 for all other arguments.

It suffices to show that d is a pre-fixed point of beh up-to u (wrt. \leq). We can use the property that $u(d)(X_1 \cup X_2, Y_1 \cup Y_2) \leq \max\{d(X_1, Y_1), d(X_2, Y_2)\}$ (see [10] for more details). Now the claim follows from unfolding the fixpoint equation by considering a - and b -transitions:

$$\begin{aligned} \text{beh}(u(d))(\{x_0, y_0\}, \{z_0\}) &= \max\{u(d)(\{x_0, x_1, y_0\}, \{z_0, z_1\}), u(d)(\{x_0, y_0, y_1\}, \{z_0, z_1\})\} \\ &\leq \max\{d(\{x_0, y_0\}, \{z_0\}), d(\{x_1\}, \{z_1\}), d(\{y_1\}, \{z_1\})\} = 1/4 \end{aligned}$$

The arguments for $d(\{x_i\}, \{z_i\})$, $d(\{y_i\}, \{z_i\})$ are similar, concluding the proof.

8 Conclusion

Related work. While the notion of Kantorovich distance on probability distributions is much older, Kantorovich liftings in a categorical framework have first been introduced in [2] and have since been generalized, for instance to codensity liftings [19] or to lifting fuzzy lax extensions [31].

A coalgebraic theory of up-to techniques was presented in [6] and has been instantiated to the setting of coalgebraic behavioural metrics in [5]. The latter paper concentrated on Wasserstein liftings, which leads to a significantly different underlying theory. Furthermore, Wasserstein liftings are somewhat restricted, since they rely only on a single evaluation map and on couplings (which sometimes do not exist, making it difficult to define more fine-grained metrics). We are not aware of a way to handle the two case studies directly in the Wasserstein approach.

Setting up the fibred adjunction (Section 4) and the definition of the Kantorovich lifting (Section 5.1) has some overlap to [4] and the recent [19]. Our focus is primarily on showing fibredness (naturality) via a quantalic version of the extension theorem.

The aim of [19] is on combining behavioural conformance via algebraic operations, which is different than our notion of compositionality via functor liftings. There is some similarity in the aims of both papers, namely the lifting of distributive laws and the motivation to study up-to techniques. From our point of view, the obtained results are largely orthogonal: while the focus of [19] is on providing n -ary operations for composing conformances and games and it provides a more general high-level account, we focus concretely on compositionality of functor liftings (studying counterexamples and treating the special case of polynomial functors), giving concrete recipes for lifting distributive laws and applying the results to proving upper bounds via up-to techniques.

Our up-to techniques are a form of up-to convex contextual closure: here they arise as specific instances of a general construction, but they have already been investigated in the Wasserstein setting [5] and earlier in [9]. Similar constructions are studied in [21].

Future work. Our aim is to better understand the metric congruence results employed in the case studies, comparing them with the similar proof rules in [21]. Compositionality of functor liftings fails in important cases (cf. Example 12), which could be fixed by using different sets of evaluation maps such as the Moss liftings in [31]. We also plan to study case studies involving the convex powerset functor [8]. Finally, we want to develop witness generation methods, by constructing suitable post-fixpoints up-to on-the-fly, similar to [1].

References

- 1 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. *Logical Methods in Computer Science*, 13(2:13):1–25, 2017. doi:10.23638/LMCS-13(2:13)2017.
- 2 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic behavioural metrics. *Logical Methods in Computer Science*, 14(3), 2018. Selected Papers of the 6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015). doi:10.23638/LMCS-14(3:20)2018.
- 3 Harsh Beohar, Sebastian Gurke, Barbara König, and Karla Messing. Hennessy-Milner theorems via Galois connections. In *Proc. of CSL '23*, volume 252 of *LIPICs*, pages 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.CSL.2023.12.
- 4 Harsh Beohar, Sebastian Gurke, Barbara König, Karla Messing, Jonas Forster, Lutz Schröder, and Paul Wild. Expressive quantale-valued logics for coalgebras: an adjunction-based approach. In *Proc. of STACS '24*, volume 289 of *LIPICs*, pages 10:1–10:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.STACS.2024.10.
- 5 Filippo Bonchi, Barbara König, and Daniela Petrişan. Up-to techniques for behavioural metrics via fibrations. *Mathematical Structures in Computer Science*, 33(4–5):182–221, 2023. doi:10.1017/s0960129523000166.
- 6 Filippo Bonchi, Daniela Petrişan, Damien Pous, and Jurriaan Rot. A general account of coinduction up-to. *Acta Informatica*, 54(2):127–190, 2017. doi:10.1007/s00236-016-0271-4.
- 7 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. of POPL '13*, pages 457–468. ACM, 2013. doi:10.1145/2429069.2429124.
- 8 Filippo Bonchi, Alexandra Silva, and Ana Sokolova. The power of convex algebras. In *Proc. of CONCUR '17*, volume 85 of *LIPICs*, pages 23:1–23:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.CONCUR.2017.23.
- 9 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized bisimulation metrics. In *Proc. of CONCUR '14*. Springer, 2014. LNCS/ARCoSS 8704. doi:10.1007/978-3-662-44584-6_4.
- 10 Keri D'Angelo, Sebastian Gurke, Johanna Maria Kirss, Barbara König, Matina Najafi, Wojciech Różowski, and Paul Wild. Behavioural metrics: Compositionality of the Kantorovich lifting and an application to up-to techniques, 2024. arXiv:2404.19632. doi:10.48550/arXiv.2404.19632.
- 11 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Transactions on Software Engineering*, 35(2):258–273, 2009. doi:10.1109/TSE.2008.106.
- 12 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318:323–354, 2004. doi:10.1016/j.tcs.2003.09.013.
- 13 Dirk Hofmann. Topological theories and closed objects. *Adv. Math.*, 215(2):789–824, 2007. doi:10.1016/j.aim.2007.04.013.
- 14 Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundation of Mathematics*. Elsevier, 1999.
- 15 Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In *Essays Dedicated to Joseph A. Goguen*, pages 375–404. Springer, 2006. LNCS 4060. doi:10.1007/11780274_20.
- 16 Bart Jacobs. *Introduction to Coalgebra. Towards Mathematics of States and Observations*. Cambridge University Press, December 2016. doi:10.1017/CB09781316823187.
- 17 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. *Journal of Computer and System Sciences*, 81(5):859–879, 2015. doi:10.1016/j.jcss.2014.12.005.
- 18 Shin-ya Katsumata and Tetsuya Sato. Codensity liftings of monads. In *Proc. of CALCO '15*, volume 35 of *LIPICs*, pages 156–170. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.CALCO.2015.156.

- 19 Mayuko Kori, Kazuki Watanabe, Jurriaan Rot, and Shin ya Katsumata. Composing codensity bisimulations. In *Proc. of LICS '24*, pages 52:1–52:13. ACM, 2024. doi:10.1145/3661814.3662139.
- 20 Francis William Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del seminario matematico e fisico di Milano*, 43(1):135–166, 1973. doi:10.1007/BF02924844.
- 21 Radu Mardare, Prakash Panangaden, and Gordon Plotkin. Quantitative algebraic reasoning. In *Proc. of LICS '16*, pages 700–709. ACM, 2016. doi:10.1145/2933575.2934518.
- 22 E. J. McShane. Extension of range of functions. *Bull. Amer. Math. Soc.*, 40(12):837–842, 1934.
- 23 Damien Pous. Complete lattices and up-to techniques. In *Proc. of APLAS '07*, pages 351–366. Springer, 2007. LNCS 4807. doi:10.1007/978-3-540-76637-7_24.
- 24 J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.
- 25 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011.
- 26 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1:09), 2013. doi:10.2168/LMCS-9(1:9)2013.
- 27 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
- 28 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331:115–142, 2005. doi:10.1016/j.tcs.2004.09.035.
- 29 Cédric Villani. *Optimal Transport – Old and New*, volume 338 of *A Series of Comprehensive Studies in Mathematics*. Springer, 2009.
- 30 Hassler Whitney. Analytic extensions of differentiable functions defined in closed sets. *Transactions of the American Mathematical Society*, 36(1):63–89, 1934.
- 31 Paul Wild and Lutz Schröder. Characteristic logics for behavioural metrics via fuzzy lax extensions. In *Proc. of CONCUR '20*, volume 171 of *LIPICs*, pages 27:1–27:23. Schloss Dagstuhl, 2020. doi:10.4230/LIPICs.CONCUR.2020.27.
- 32 Paul Wild and Lutz Schröder. Characteristic logics for behavioural hemimetrics via fuzzy lax extensions. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:19)2022.

Reversible Transducers over Infinite Words

Luc Dartois  

Université Paris Est Creteil, LACL, F-94010 Créteil, France

Paul Gastin  

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
CNRS, ReLaX, IRL 2000, Siruseri, India

Loïc Germerie Guizouarn  

Université Paris Est Creteil, LACL, F-94010 Créteil, France

R. Govind  

Uppsala University, Sweden

Shankaranarayanan Krishna  

Indian Institute of Technology Bombay, Mumbai, India

Abstract

Deterministic two-way transducers capture the class of regular functions. The efficiency of composing two-way transducers has a direct implication in algorithmic problems related to synthesis, where transformation specifications are converted into equivalent transducers. These specifications are presented in a modular way, and composing the resultant machines simulates the full specification. An important result by Dartois et al. [10] shows that composition of two-way transducers enjoy a polynomial composition when the underlying transducer is reversible, that is, if they are both deterministic and co-deterministic. This is a major improvement over general deterministic two-way transducers, for which composition causes a doubly exponential blow-up in the size of the inputs in general. Moreover, they show that reversible two-way transducers have the same expressiveness as deterministic two-way transducers. However, the notion of reversible two-way transducers over infinite words as well as the question of their expressiveness were not studied yet.

In this article, we introduce the class of reversible two-way transducers over infinite words and show that they enjoy the same expressive power as deterministic two-way transducers over infinite words. This is done through a non-trivial, effective construction inducing a single exponential blow-up in the set of states. Further, we also prove that composing two reversible two-way transducers over infinite words incurs only a polynomial complexity, thereby providing an efficient procedure for composition of transducers over infinite words.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases Transducers, Regular functions, Reversibility, Composition, SSTs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.21

Related Version *Full Version:* <https://arxiv.org/abs/2406.11488> [12]

1 Introduction

Transducers extend finite state automata with outputs. While finite state automata are computational models for regular languages, transducers are computational models for transformations between languages. Finite state automata remain robust in their expressiveness accepting regular languages across various descriptions like allowing two-way-ness, non-determinism and otherwise. However, this is not the case with transducers: first of all, non-deterministic transducers realize relations while deterministic transducers realize functions. Further, two-way transducers are strictly more expressive than one-way transducers: for instance, the function reverse which computes the reverse of all input words in its domain is realizable by deterministic two-way transducers, but not by one-way transducers.



© Luc Dartois, Paul Gastin, Loïc Germerie Guizouarn, R. Govind, and Shankaranarayanan Krishna; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 21; pp. 21:1–21:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One of the cornerstone results of formal language theory is the beautiful connection which establishes that the class of regular languages corresponds to those recognized by finite state automata, to the class of languages definable in MSO logic, and to the class of languages whose syntactic monoid is finite. Engelfriet and Hoozeboom [17] generalized this correspondence between machines, logics and algebra in the case of regular languages to regular transformations. They showed that regular transformations are those which are captured by two-way transducers and by Monadic second-order (MSO) transductions a la Courcelle [9]. Inspired by this seminal work of Engelfriet and Hoozeboom, there has been an increasing interest over recent years in characterizing the class of functions defined by deterministic two-way transducers [2, 3, 16].

One such characterization is that of *reversible* two-way transducers [10] over finite words. Reversible transducers are those which are deterministic and also co-deterministic. While determinism says that any given state, on any given input symbol, does not transition to two distinct states, co-determinism says that no two distinct states can transition to the same state on any input symbol. Reversibility makes the composition operation in two-way transducers very efficient : the composition of reversible transducers has polynomial state complexity. [10] showed that reversible two-way transducers capture the class of regular transformations. However, reversible transducers over infinite words have not been studied.

In another line of work, [1] initiated the study of transformations on infinite words. They considered functional, copy-less streaming string transducers (SST) with a Müller acceptance condition. An SST is a one-way automaton with registers; the outputs of each transition are stored in registers as words over the register names and the output alphabet. In a run, the contents of the registers are composed. The Müller acceptance condition is defined as follows: in any accepting run which settles down in a Müller set, the output is defined as a concatenation x_1, x_2, \dots, x_n of registers where only x_n is updated by appending words to x_n . [1] proved the equivalence of this class of SST to deterministic two-way transducers with Müller acceptance and having an ω -regular look-ahead. They also showed that these are equivalent to MSO transductions over infinite words. The ω -regular look-aheads were necessary to obtain the expressiveness of MSO transductions on infinite words.

In this paper, we continue the study of two-way transducers over infinite words. We introduce two-way deterministic transducers with generalized parity acceptance condition (gp2DT). Our main result is a non-trivial generalization of [10], where we show that gp2DT's can be made reversible, obtaining gp2RT (two-way reversible transducers with generalized parity acceptance). Our conversion of gp2DT to gp2RT incurs a single exponential blow-up, and goes via a new kind of SSTs that we introduce, namely, copyless SST with a generalized parity acceptance condition (gpSST). The parity condition used in both machines employs a finite set χ of coloring functions, where each $c \in \chi$ assigns to the transitions of the underlying machine, a natural number. An infinite run ρ is accepting if for each coloring function $c \in \chi$, the minimum number which appears infinitely often is even. This conjunction of parity conditions is one of the generalized parity accepting conditions introduced in [8]. We summarize our main contributions below.

1. We first show that starting from a gp2DT, we can obtain an equivalent gpSST where the number of states of the gpSST is exponential in the number of states and coloring functions of the gp2DT. This is a fairly non-trivial generalization of the classical Shepherdson construction [19] which goes from two-way automata to one-way automata.
2. Then we show that, starting from a gpSST \mathcal{A} , we can obtain an equivalent gp2RT \mathcal{B} which is polynomial in the number of states and registers of \mathcal{A} . This construction is a bit technical: we show that \mathcal{B} is obtained as the composition of a deterministic one-way

generalized parity transducer \mathcal{D} and an gp2RT \mathcal{F} . To complete the proof, we show that (i) \mathcal{D} can be converted to an equivalent gp2RT with polynomial blow-up, and (ii) gp2RTs are closed under composition with a polynomial complexity.

Thus, our results extend [10] to the setting of infinite words, retaining the expressivity of deterministic machines and a polynomial complexity for composition. The main challenges when going to the infinite word setting is in dealing with the acceptance conditions. Unlike the finite word setting where acceptance is something to take care of at the end, here we need to deal with it throughout the run. The difficulty in doing this comes from the fact that we cannot compute the set of co-accessible states at a given input position. It must be noted that in the proof [10] for finite words, the equivalent reversible transducer was constructed by computing the set of accessible and co-accessible states at each position of the input word. Indeed, computing the co-accessible states at each input position requires an infinite computation or an oracle, and hence, the proof of [10] fails for infinite words. Instead, we introduce the intermediate model of gpSST where we employ a dedicated “out” register that serves as the output tape.

Our result of extending [10] can be seen as a positive contribution to synthesis, especially in settings where one requires more expressive specifications that cannot be implemented by sequential machines. These include instances where we have access to an unbounded input-buffer, or where the past is reproducible. For these, two-way transducers are an attractive model. In these settings, thanks to the polynomial composition result for reversible transducers, they are a natural modeling choice. Using reversible transducers allows for a modular approach for specification transformation and can give rise to efficient solutions to algorithmic problems on transformation of specifications. To the best of our knowledge, there is no such translation for infinite words; the closest result in this direction, but for finite words, is [11], which gives an efficient procedure for converting specifications given as RTE (regular transducer expressions) to reversible transducers.

Continuing with transformations on infinite words, [15] investigated a practical question on functions over infinite words, namely, “given a function over infinite words, is it computable?”. They established that the decidability of this question boils down to checking the continuity of these functions. Further, they conjectured that any continuous regular function can be computed by a deterministic two-way transducer over infinite words without ω -regular look-ahead. [5] took up this conjecture and showed that any continuous rational function over infinite words can be extended to a function which is computable by deterministic two-way transducers over infinite words without ω -regular look-ahead. Most recently, [6] conjectured that deterministic two-way transducers with the Büchi acceptance condition capture the class of continuous, regular functions.

Apart from its application to synthesis, gp2RT also realize continuous functions. This implies that the conjecture of [6] fails, since gp2RT are more expressive than the class of deterministic two-way transducers with Büchi acceptance. A simple example illustrating this is the function $f : \{a, b\}^\omega \rightarrow \{a, b\}^\omega$ such that $f(u) = u$ if the number of a 's in u is finite, and is undefined otherwise. f is continuous since it is continuous on its domain; f cannot be realized by a deterministic transducer with Büchi acceptance, but it can be realized by a gp2RT. Note however that the extension is only able to refine the domain, and not the production. In particular, by simply dropping the accepting condition of a gp2RT, we obtain a function realized by a deterministic two-way transducer with a Büchi condition. Moreover, our constructions (going from deterministic two-way to reversible) for this class become simpler. And conversely, we show that two-way reversible transducers with no acceptance condition have the same expressiveness as those with the Büchi acceptance condition, which in turn have the same expressiveness as two-way deterministic transducers with Büchi acceptance.

The choice of a generalized parity acceptance condition, more specifically a conjunction of parity conditions, was motivated by the need for an efficient procedure for the intersection of automata. This intersection is necessary for the composition of transducers. It is well known [4] that the intersection of standard parity, Müller, or Rabin automata induce an exponential blowup in the number of states, while Streett or conjunction of parity incur only a polynomial blowup.

Organization of the Paper. Section 2 defines the two models we introduce in the paper, namely, **gpSST** and **gp2DT**. Section 3 states our main result : starting from a **gp2DT**, we can obtain an equivalent **gp2RT**. Most of the remaining sections are devoted to the proof of this result. In sections 4 and 5 respectively, we prove the closure under composition of **gp2RTs** with a polynomial complexity and the polynomial conversion from one-way generalized parity transducers to **gp2RT**. Section 6 uses both these results, where we describe the conversion from **gpSST** to **gp2RT** with a polynomial complexity. Section 7 contains one of the most non-trivial constructions of the paper, namely, going from **gp2DT** to **gpSST** with a single exponential blow-up. Finally, Section 8 wraps up by discussing the connection between continuity and the topological closure of **gp2DTs**. Omitted proofs can be found in the full version [12].

2 Preliminaries

Let A be an *alphabet*, i.e., a finite set of letters. A finite or infinite word w over A is a (possibly empty) sequence $w = a_0 a_1 a_2 \dots$ of letters $a_i \in A$. The set of all finite (resp. infinite) words is denoted by A^* (resp. A^ω), with ε denoting the empty word. We let $A^\infty = A^* \cup A^\omega$. A *language* is a subset of the set of all words.

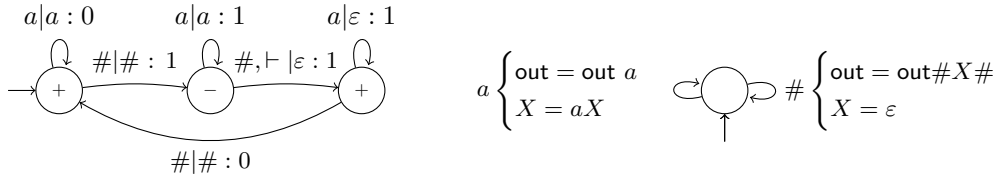
Two-way Parity Automata and Transducers

Let A be a finite alphabet and let $\vdash \notin A$ be a left delimiter symbol. We write $A_\vdash = A \cup \{\vdash\}$.

A two-way generalized parity automaton (**gp2A**) is a tuple $\mathcal{A} = (Q, A, \Delta, q_0, \chi)$, where the finite set of states Q is partitioned into a set of forward states Q^+ and a set of backward states Q^- . The initial states is $q_0 \in Q^+$, $\Delta \subseteq Q \times A_\vdash \times Q$ is the transition relation and χ is a finite set of coloring functions $c: \Delta \rightarrow \mathbb{N}$ which are used to define the acceptance condition. We assume that if $(p, \vdash, q) \in \Delta$, then $p \in Q^-$ and $q \in Q^+$: on reading \vdash , the reading head does not move.

A configuration of a **gp2A** over an input word $w \in A^\omega$ is some $\vdash u p v$ where $p \in Q$ is the current state and $u \in A^*$, $v \in A^\omega$ with $w = uv$. The configuration admits several successor configurations as defined below.

1. If $p \in Q^+$, then the input head reads the first symbol $a \in A$ of the suffix $v = av' \in A^\omega$. Let $(p, a, q) \in \Delta$ be a transition. If $q \in Q^+$, then the successor configuration is $\vdash ua q v'$. Likewise, if $q \in Q^-$, then the successor configuration is $\vdash u q av'$. Thus, if the current and target states are both in Q^+ , then the reading head moves right. If the current state is forward and the target state is backward, then the reading head does not move.
2. If $p \in Q^-$, then the input head reads the last symbol $a \in A_\vdash$ of the prefix $\vdash u$. Let $(p, a, q) \in \Delta$ be a transition. If $q \in Q^+$, the successor configuration is $\vdash u q v$. If $q \in Q^-$ then $a \neq \vdash$, we write $u = u'a$ with $u' \in A^*$ and the successor configuration is $\vdash u' q av$. Thus, if the current state is backward and the target state is forward, the reading head does not move. If both states are backward, then the reading head moves left.



■ **Figure 1** An example of gp2RT (left) and gpSST (right) defining the function *map-copy-reverse* (*mcr*) defined on $(A \uplus \{\#\})^\omega \rightarrow (A \uplus \{\#\})^\omega$ by: $mcr(u_1\#u_2\#\dots) = u_1\#\tilde{u}_1\#u_2\#\tilde{u}_2\#\dots$ for words with an infinite number of letter #, and $mcr(u_1\#\dots\#u_n\#u) = u_1\#\tilde{u}_1\#\dots\#u_n\#\tilde{u}_n\#u$ if $u \in A^\omega$, where \tilde{v} denotes the mirror image of v . There is only one coloring function, denoted on the transitions after the colon. The color of all transitions of the gpSST is 0.

A run ρ of \mathcal{A} is a finite or infinite sequence of configurations starting from an initial configuration $\vdash \varepsilon q_0 w$ where $w \in A^\omega$ is the input word:

$$\vdash q_0 w = \vdash u_0 q_0 v_0 \rightarrow \vdash u_1 q_1 v_1 \rightarrow \vdash u_2 q_2 v_2 \rightarrow \vdash u_3 q_3 v_3 \rightarrow \vdash u_4 q_4 v_4 \dots$$

We say that ρ reads the whole word $w \in A^\omega$ if $\sup\{|u_n| \mid n > 0\} = \infty$. The set of transitions used by ρ infinitely often is denoted $\text{inf}(\rho) \subseteq \Delta$. The word w is accepted by \mathcal{A} , i.e., $w \in \text{dom}(\mathcal{A})$ if ρ reads the whole word w and $\min(c(\text{inf}(\rho)))$ is even for all $c \in \chi$.

The generalized parity automaton \mathcal{A} is called

- *one-way* if $Q^- = \emptyset$,
- *deterministic* if for all pairs $(p, a) \in Q \times A_+$, there is at most one state $q = \delta(p, a)$ such that $(p, a, q) \in \Delta$, in this case we identify the transition relation Δ with the partial function $\delta: Q \times A \rightarrow Q$,
- *co-deterministic* if for all pairs $(q, a) \in Q \times A_+$, there is at most one state p such that $(p, a, q) \in \Delta$,
- *reversible* if it is both deterministic and co-deterministic.

A two-way generalized parity transducer gp2T is a tuple $\mathcal{T} = (Q, A, \Delta, q_0, \chi, B, \lambda)$ where $\mathcal{A} = (Q, A, \Delta, q_0, \chi)$ is a *deterministic* gp2A, called the underlying automaton of \mathcal{T} , B is a finite *output* alphabet, and $\lambda: \Delta \rightarrow B^*$ is the output function. As in the case of gp2A, a gp2T is one-way/co-deterministic/reversible if so is the underlying generalized parity automaton. Let gp2DT (resp. gp2RT) denote two-way deterministic (resp. reversible) generalized parity transducers. The notion of run and accepting run is inherited from the underlying gp2A. For $w \in A^\omega$ such that $w \in \text{dom}(\mathcal{A})$, let the accepting run ρ of w be

$$\vdash q_0 w = \vdash u_0 q_0 v_0 \xrightarrow{t_1} \vdash u_1 q_1 v_1 \xrightarrow{t_2} \vdash u_2 q_2 v_2 \xrightarrow{t_3} \vdash u_3 q_3 v_3 \xrightarrow{t_4} \vdash u_4 q_4 v_4 \dots$$

where $t_i \in \Delta$ is the i -th transition taken during the run, i.e., from $\vdash u_{i-1} q_{i-1} v_{i-1}$ to $\vdash u_i q_i v_i$. For $i > 0$, let $\gamma_i = \lambda(t_i)$ be the output produced by the i -th transition of ρ . If $\gamma_1 \gamma_2 \gamma_3 \gamma_4 \dots \in B^\omega$, then $w \in \text{dom}(\mathcal{T})$ and we let $\llbracket \mathcal{T} \rrbracket(w) = \gamma_1 \gamma_2 \gamma_3 \gamma_4 \dots$ be the output word computed by \mathcal{T} . Hence, the semantics of a gp2T is a partial function $\llbracket \mathcal{T} \rrbracket: A^\omega \rightarrow B^\omega$ with $\text{dom}(\mathcal{T}) \subseteq \text{dom}(\mathcal{A})$.

Parity Streaming String Transducers

Let R be a finite set of variables called *registers*. A *substitution* of R into an alphabet B is a mapping $\sigma: R \rightarrow (R \uplus B)^*$. It is called *copyless* if for all $r \in R$, r appears at most once in the concatenation of all the $\sigma(r')$ for $r' \in R$. We denote by Λ_R^B the set of all copyless substitutions

of R into B . A copyless generalized parity Streaming String Transducer (gpSST) is given by a tuple $\mathcal{T} = (Q, A, \Delta, q_0, \chi, B, R, \text{out}, \lambda)$ where $\mathcal{A} = (Q, A, \Delta, q_0, \chi)$ is a deterministic one-way generalized parity automaton called the underlying automaton of \mathcal{T} , R is a finite set of registers, $\text{out} \in R$ is a distinguished register, called the output register, $\lambda: \Delta \rightarrow \Lambda_R^B$ is the update function satisfying additionally $\lambda(t)(\text{out}) \in \text{out} \cdot (R \uplus B)^*$ for all $t \in \Delta$.

A configuration of a copyless generalized parity SST \mathcal{T} is a tuple (q, ν) where $q \in Q$ and $\nu: R \rightarrow B^*$ is an assignment. The initial configuration is (q_0, ν_0) where $\nu_0(r) = \varepsilon$ for all $r \in R$. Since the automaton \mathcal{A} is *deterministic*, we simply describe a run on an input word $w = a_0 a_1 a_2 \dots$ as a sequence a sequence of transitions applying the corresponding substitutions to the assignments:

$$(q_0, \nu_0) \xrightarrow{a_0} (q_1, \nu_1) \xrightarrow{a_1} (q_2, \nu_2) \xrightarrow{a_2} (q_3, \nu_3) \dots$$

where (q_0, ν_0) is the initial configuration and for all $i \geq 0$ we have $t_i = (q_i, a_i, q_{i+1}) \in \Delta$ and $\nu_{i+1} = \nu_i \circ \lambda(t_i)^1$. Notice that, from the restriction of the update function, we deduce that $\nu_0(\text{out}), \nu_1(\text{out}), \nu_2(\text{out}), \dots$ is a (weakly) increasing sequence of output words in B^* . If this sequence is unbounded then $w \in \text{dom}(\mathcal{T})$ and we let $\llbracket \mathcal{T} \rrbracket(w) = \bigsqcup_{i \geq 0} \nu_i(\text{out}) \in B^\omega$ be the limit (least upper-bound) of this sequence. Hence, the semantics of a gpSST is a partial function $\llbracket \mathcal{T} \rrbracket: A^\omega \rightarrow B^\omega$ with $\text{dom}(\mathcal{T}) \subseteq \text{dom}(\mathcal{A})$.

3 Main Result

We are now ready to state our main result, which is an effective procedure to construct a reversible two-way transducer for a deterministic machine.

The proof relies on constructions that go through gpSST, and are presented in the subsequent sections.

► **Theorem 1.** *Given a deterministic gp2DT T with n states, k color conditions and ℓ colors, we can construct a gp2RT S with $O(\ell^{2kn}(2n)^{4n+1})$ states, k color conditions and ℓ colors such that $\llbracket T \rrbracket = \llbracket S \rrbracket$.*

Proof. Let T be a gp2DT with n states, k color conditions and ℓ colors. Using Theorem 6, we can construct an equivalent gpSST T' with $O(n(\ell^k)^n(2n+1)^{2n-1})$ states, $2n$ variables, k color conditions and ℓ colors. Then by Theorem 4, we can construct a gp2RT S equivalent to T' whose size is quadratic in the number of states and linear in the number of variables. More precisely, S has $O((n(\ell^k)^n(2n+1)^{2n-1})^2(2n)) = O(\ell^{2kn}(2n)^{4n+1})$ states, k color conditions and ℓ colors, concluding the proof. ◀

4 Composition of gp2RT

The main reason to use reversible two-way machines is that they are easily composable. Given two composable reversible transducers, we can construct one whose size is linear in both machines, and whose transition function is rather straight-forward. It is made explicit in the following theorem and proof.

¹ An assignment $\nu: R \rightarrow B^*$ is extended to a morphism $\nu: (R \uplus B)^* \rightarrow B^*$ by $\nu(b) = b$ for all $b \in B$. Hence, if $\sigma \in \Lambda_R^B$ is a substitution then $\nu' = \nu \circ \sigma$ is an assignment defined by $\nu'(r) = \nu(\sigma(r))$ for all $r \in R$. For instance, if $\sigma(r) = br'cbr$ then $\nu'(r) = b\nu(r')cb\nu(r)$.

► **Theorem 2.** *Given two gp2RT \mathcal{S} and \mathcal{T} , of size n and m respectively, and such that the output alphabet of \mathcal{S} is the input alphabet of \mathcal{T} , we can construct a gp2RT \mathcal{U} , also denoted by $\mathcal{T} \circ \mathcal{S}$, of size $O(nm)$ such that $\llbracket \mathcal{U} \rrbracket = \llbracket \mathcal{T} \rrbracket \circ \llbracket \mathcal{S} \rrbracket$.*

Sketch of proof. The set of states of the machine \mathcal{U} is the cartesian product of the sets of states of \mathcal{S} and \mathcal{T} . Given an input word u of \mathcal{S} , \mathcal{U} simulates \mathcal{S} . At each step, instead of producing a possibly empty word $v \in B^+$, it simulates the run $\rho_{\mathcal{T}}$ of \mathcal{T} over v . If $\rho_{\mathcal{T}}$ exits v on the right, then \mathcal{U} continues the simulation of \mathcal{S} up to the next transition producing a nonempty word. Otherwise, it rewinds the run of \mathcal{S} to get its previous production, and simulates \mathcal{T} on it, starting from the right.

The conjunction of parity conditions allows to an easy construction for intersection, which is similar to what is expected here. A word u should be accepted if u belongs to the domain of \mathcal{S} and $\llbracket \mathcal{S} \rrbracket(u)$ belongs to the domain of \mathcal{T} . By doing the conjunction of both acceptance, we are able to recognize the domain of $\mathcal{U} = \mathcal{T} \circ \mathcal{S}$. ◀

Proof. Let $\mathcal{S} = (Q, A, \delta, q_0, \chi, B, \lambda)$ and $\mathcal{T} = (P, B, \alpha, p_0, \chi', C, \beta)$. We define the composition $\mathcal{U} = \mathcal{T} \circ \mathcal{S} = (R, A, \mu, r_0, \chi'', C, \nu)$ where $R = Q \times P$ is split as

$$R^+ = Q^+ \times P^+ \cup Q^- \times P^- \quad R^- = Q^- \times P^+ \cup Q^+ \times P^- .$$

The initial state is $r_0 = (q_0, p_0)$ and μ, ν and χ'' are defined below.

To properly define μ and ν , we extend α and β to finite words, and more precisely to the productions of \mathcal{S} . Given a word $v = \lambda(q, a) \in B^*$ for some $(q, a) \in Q \times A$, and a state p of \mathcal{T} , we define $\rho_p(v)$ to be the maximal run of \mathcal{T} over v starting in state p on the left (resp. right) of v if $p \in P^+$ (resp. $p \in P^-$). Then we define $\alpha^*(p, v)$ as the state reached by $\rho_p(v)$ when exiting v . It is undefined if $\rho_p(v)$ loops within v . Note that if $\alpha^*(p, v)$ belongs to P^+ (resp. P^-), then \mathcal{T} exits v on the right (resp. on the left). We define $\beta^*(p, v)$ as the concatenation of the productions of $\rho_p(v)$. If $\alpha^*(p, v)$ is defined, then $\beta^*(p, v)$ is finite. Note that $\rho_p(\varepsilon)$ is an empty run, so we have $\alpha^*(p, \varepsilon) = p$ and $\beta^*(p, \varepsilon) = \varepsilon$.

For the generalized parity conditions, we let $\chi'' = \{\bar{c} \mid c \in \chi \cup \chi'\}$ and we extend the functions $c \in \chi'$ to finite runs $\rho_p(v)$. More precisely, we let $c^*(p, v)$ be the minimum c -value taken by the transitions of $\rho_p(v)$. When $v = \varepsilon$ then $\rho_p(v)$ is an empty run and we set $c^*(p, v)$ to the largest odd value in all values taken by c on transitions of \mathcal{T} . Then, given a state (q, p) of \mathcal{U} ,

- If $p \in P^+$ then we let $v = \lambda(q, a)$. We set $\nu((q, p), a) = \beta^*(p, v)$ and, with $q' = \delta(q, a)$ and $p' = \alpha^*(p, v)$ we define

$$\mu((q, p), a) = \begin{cases} (q', p') & \text{if } p' \in P^+, \\ (q, p') & \text{if } p' \in P^- \end{cases} \quad \text{and} \quad \bar{c}((q, p), a) = \begin{cases} c(q, a) & \text{if } c \in \chi, \\ c^*(p, v) & \text{if } c \in \chi'. \end{cases}$$

- If $p \in P^-$ then we let q' be such that $q = \delta(q', a)$ and $v = \lambda(q', a)$. Note that q' is unique by co-determinism of \mathcal{S} . We set $\nu((q, p), a) = \beta^*(p, v)$ and, with $p' = \alpha^*(p, v)$ we define

$$\mu((q, p), a) = \begin{cases} (q, p') & \text{if } p' \in P^+, \\ (q', p') & \text{if } p' \in P^- \end{cases} \quad \text{and} \quad \bar{c}((q, p), a) = \begin{cases} c(q', a) & \text{if } c \in \chi, \\ c^*(p, v) & \text{if } c \in \chi'. \end{cases}$$

The intuition behind the transition function is that \mathcal{U} simulates \mathcal{S} to feed a simulation of \mathcal{T} . If \mathcal{T} moves forward on its input, then \mathcal{U} simulates \mathcal{S} forward. If \mathcal{T} moves backward on its input, then \mathcal{U} backtracks the computation of \mathcal{S} . During a switch of direction, \mathcal{S} stays put.

The acceptance condition of conjunctive parity was chosen specifically to allow for smooth composition. By using both sets of parities, the transducer \mathcal{U} ensures that the input word is accepted by \mathcal{S} , and that its production is accepted by \mathcal{T} . It is worth noting that since \mathcal{U} can rewind the run of \mathcal{S} , it can take a transition (and consequently its colors) multiple times on a given input position. This increases the multiplicity of the transitions taken during a non-looping run by a constant factor since a deterministic transducer never visits twice a given position in the same state. Hence, the set of colors of \mathcal{S} that \mathcal{U} sees infinitely often on a non-looping run is the same as the ones seen by \mathcal{S} . ◀

5 gpDT to gp2RT

Similar to finite words, given a deterministic one-way generalized parity transducer, one can construct a reversible one realizing the same function.

► **Theorem 3.** *Let \mathcal{T} be a gpDT, we can construct a gp2RT \mathcal{T}' of size $O(n^2)$ such that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{T}' \rrbracket$.*

Sketch of proof. The construction is reminiscent of the tree-outline construction for co-deterministic transducers of [10]. The difference is that here, we begin with a deterministic transducer with an infinite input word, so instead of starting from the root of the tree, which is at the end of a finite input word, our outline has to start from a leaf at the beginning of the input word, corresponding to the initial configuration. We also generalize the construction by allowing any degree of non (co-)determinism: while in [10], at most two branches could merge on any vertex, here we allow any number.

We begin with the underlying automaton: from a one-way deterministic generalized parity automaton (gpDA) \mathcal{A} , we build a two-way reversible generalized parity automaton (gp2RA) \mathcal{A}' simulating the behavior of \mathcal{A} . For any accepted input word w , we consider the infinite acyclic graph (simply called a tree) representing all the partial runs of \mathcal{A} merging with the accepting run of \mathcal{A} on w (note that because \mathcal{A} is deterministic, there is only one accepting run for a given word). Automaton \mathcal{A}' will simulate two synchronized reading heads going along the outline of this tree, as illustrated in Figure 2.

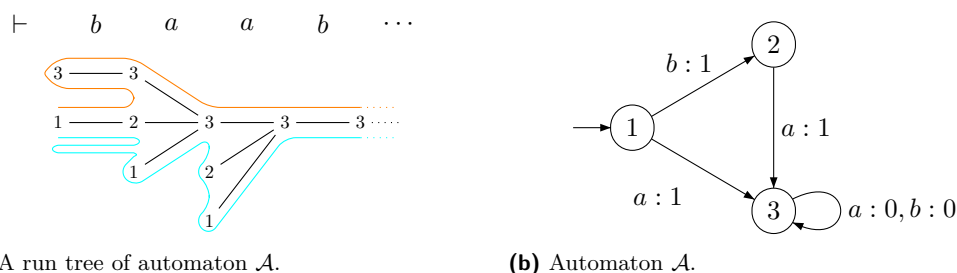
The two heads are required to make \mathcal{A}' equivalent to \mathcal{A} : we need to be able to discriminate configurations of a run of \mathcal{A}' occurring in the accepting run of \mathcal{A} from the ones added to account for the non-initial runs. One reading head follows the outline of the run tree from above, and the other one from below. The configurations where the two reading heads point to the same state of \mathcal{A} correspond to those occurring in the accepting run of this automaton.

The reading heads are placed above and below the initial state, and they move together to the right, until one of them encounters a branching in the tree. When this happens, the gp2RA moves backwards to go around the branch. When the branch dies (which necessarily happens because from each position, the prefix of a word is finite), the exploration continues to the right.

As the two heads are synchronized, and because branches may not all be of the same length, when one head needs going left the other one may impose right moves in order to reach another branch, on which it will be able to go left far enough to follow the first head.

A run of \mathcal{A}' can be seen as a straightforward journey along the flattened outline of the tree, hence the reversibility.

Once \mathcal{A}' is defined, we define \mathcal{T}' : it is the gp2RT having \mathcal{A}' as underlying automaton, and whose output function is that of \mathcal{T} from states where the two reading heads point to the same state, and ε otherwise. ◀

(a) A run tree of automaton \mathcal{A} .(b) Automaton \mathcal{A} .

■ **Figure 2** The automaton \mathcal{A} depicted in Figure 2b recognizes all infinite words over alphabet $\{a, b\}$ having an a in first or second position (it has only one coloring function, represented after the colons in the transitions). Figure 2a is the part of the tree corresponding to the run of \mathcal{A} on the prefix $baab$ of an accepted word. Each node of the tree is a configuration of \mathcal{A} , represented here by a control state. Its horizontal position allows to deduce the position in the input word, depicted above the tree. The horizontal straight path represents the accepting run. Notice that when the top reading head needs to go backward to go around a branch, the bottom ones follows and goes backward on the accepting run.

6 From gpSST to gp2RT

We extend another result from [10] about constructing a reversible two-way transducer from a Streaming String Transducer. The procedure and the complexity are similar. The main difference is that the procedure only work thanks to the distinguished register `out` of gpSST. Without it, production could depend on an infinite property of the input word, which is not realizable by a deterministic (and hence reversible) machine.

► **Theorem 4.** *Let \mathcal{T} be a gpSST with n states and m registers. Then we can construct a gp2RT \mathcal{S} with $O(n^2m)$ states such that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{S} \rrbracket$.*

Sketch of proof. We prove that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{F} \rrbracket \circ \llbracket \mathcal{D} \rrbracket$ where \mathcal{D} is a gpDT and \mathcal{F} is a gp2RT. The transducer \mathcal{D} has the same underlying automaton as \mathcal{T} , but instead of applying a substitution σ to the registers, \mathcal{D} enriches the input letter with σ . Then, the transducer \mathcal{F} uses the flow of registers output by \mathcal{D} to output the contents of the relevant registers in a reversible fashion. Finally, we construct a gp2RT \mathcal{D}' equivalent to \mathcal{D} by Theorem 3 and we obtain the desired gp2RT $\mathcal{S} = \mathcal{F} \circ \mathcal{D}'$ by Theorem 2.

Formal Construction. Let $\mathcal{T} = (Q, A, \delta, q_0, \chi, B, R, \text{out}, \lambda)$ be a gpSST. We give the constructions of \mathcal{D} and \mathcal{F} , and dedicate the proofs of the reversibility of \mathcal{F} and the correctness of the construction to the Appendix.

We define the gpDT by $\mathcal{D} = (Q, A, \delta, q_0, \chi, \Lambda_R^B, \gamma)$ where $\gamma(q, a) = \lambda(q, a)$.

The reversible transducer is $\mathcal{F} = (Q', \Lambda_R^B, \alpha, q'_0, \emptyset, B, \beta)$ where:

- $Q' = R \times \{i, o\}$ with $Q^+ = R \times \{o\}$ and $Q^- = R \times \{i\}$. We will denote by r_i (resp. r_o) the state (r, i) (resp. (r, o)). Informally, being in state r_i means that we need to compute the content of r , while state r_o means we have just finished computing it.
- The initial state is $q'_0 = \text{out}_o$.
- There is no accepting condition;
- α and β both read a state in Q' and a substitution $\sigma \in \Lambda_R^B$, or the leftmarker \vdash , which is treated as a substitution σ_{\vdash} associating ε to every register. We define α and β as follow:
 - If the state is r_i for some $r \in R$.
 - * If $\sigma(r) = v \in B^*$ contains no register, then $\alpha(r_i, \sigma) = r_o$ and $\beta(r_i, \sigma) = v$.
 - * If $\sigma(r) = vs\gamma$ with $v \in B^*$ and $s \in R$ is the first register appearing in $\sigma(r)$, then $\alpha(r_i, \sigma) = s_i$ and $\beta(r_i, \sigma) = v$.

21:10 Reversible Transducers over Infinite Words

- If the state is r_o for some $r \in R$. Recall that from the definition of copyless SSTs, for any register r , there exists at most one register t , such that r occurs in $\sigma(t)$, and in this case r occurs exactly once in $\sigma(t)$.
 - * Suppose that for some register s we have $\sigma(s) = \gamma r v$ with $v \in B^*$. Then $\alpha(r_o, \sigma) = s_o$ and $\beta(r_o, \sigma) = v$.
 - * Suppose that for some register t we have $\sigma(t) = \gamma r v s \gamma'$ with $v \in B^*$ and $s \in R$. Then $\alpha(r_o, \sigma) = s_i$ and $\beta(r_o, \sigma) = v$.
 - * If r does not appear in any $\sigma(s)$, then the computation stops and rejects. This somehow means that we are computing the contents of a register that is dropped in the original SST. This will not happen if what is fed to \mathcal{F} is produced by \mathcal{D} .

Correctness of the construction. First, let us remark that the domain of \mathcal{D} is the set of input words u such that \mathcal{T} has an infinite accepting run over u since they share the same underlying automaton. So the domain of \mathcal{T} is the set of words in the domain of \mathcal{D} on which \mathcal{T} produces an infinite word.

Let $(q_0, \nu_0) \xrightarrow{a_0} (q_1, \nu_1) \xrightarrow{a_1} (q_2, \nu_2) \xrightarrow{a_2} (q_3, \nu_3) \cdots$ be the accepting run of some input word $u = a_0 a_1 a_2 \cdots \in A^\omega$ in the domain of the gpSST \mathcal{T} . For $j \geq 0$, let $\sigma_j = \lambda(q_j, a_j)$ so that $\llbracket \mathcal{D} \rrbracket(u) = \sigma_0 \sigma_1 \sigma_2 \cdots$.

We prove by induction that for every position $j \geq 0$ of u , the run of the transducer \mathcal{F} on $\llbracket \mathcal{D} \rrbracket(u)$ reaches the state out_o in position j having produced the content $\nu_j(\text{out})$ of the run of \mathcal{T} on u up to position j . For $j = 0$, there is nothing to prove as the registers are initially empty and out_o is the initial state of \mathcal{F} .

Now suppose that the run of \mathcal{F} on $\llbracket \mathcal{D} \rrbracket(u)$ reaches some position j in state out_o , having produced $\nu_j(\text{out})$. Recall that, by definition of λ , the substitution $\sigma_j = \lambda(q_j, a_j)$ used by \mathcal{T} at position j is such that $\sigma_j(\text{out}) = \text{out} \cdot \gamma$. Then if there is no other register, i.e., if $\gamma = v \in B^*$, by definition of α , \mathcal{F} moves to $j + 1$ in state out_o and produces v , so that its cumulated production is $\nu_j(\text{out}) \cdot v = \nu_{j+1}(\text{out})$.

The interesting case is of course when some registers are flown to out . Let r be the second register of $\sigma_j(\text{out})$, i.e., $\sigma_j(\text{out})$ starts with $\text{out} \cdot v r$ with $v \in B^*$. Then, by definition of α and β , \mathcal{F} stays at position j switching to state r_i and producing v . Then, using Claim 5, \mathcal{F} reaches r_o at position j producing the content of $\nu_j(r)$. We repeat this process to exhaust all registers appearing in $\sigma(\text{out})$, reaching finally state out_o at position $j + 1$ with cumulated production $\nu_j(\sigma_j(\text{out})) = \nu_{j+1}(\text{out})$, proving the induction.

▷ **Claim 5.** For all positions $j \geq 0$ and registers $r \in R$, there exists a right-to-right run (r_i, r_o) of \mathcal{F} starting and ending at position j and which produces the content of $\nu_j(r)$.

Proof. The proof is by induction on j . If $j = 0$ then $\nu_0(r) = \varepsilon$ and the run of \mathcal{F} starting at position 0 in state r_i reads σ_- . By definition of α and β the run produces $\sigma_-(r) = \varepsilon$ and switches from r_i to r_o , proving the claim for $j = 0$.

Now assume that the claim is true for j . Consider the run ρ of \mathcal{F} starting in state r_i at position $j + 1$. The run ρ starts by reading σ_j . If $\sigma_j(r) = v \in B^*$ then the run produces $v = \nu_{j+1}(r)$ and switches from r_i to r_o , proving the claim. The second case is when $\sigma_j(r)$ starts with some vs with $v \in B^*$ and $s \in R$. Then, the first transition of ρ produces v and moves to position j in state s_j . By induction hypothesis, there is a right-right (s_i, s_o) -run starting at j and producing $\nu_j(s)$. Then, the run reads σ_j in state s_o and, either goes to r_o in position $j + 1$ producing $v' \in B^*$ if $\sigma_j(r)$ ends with sv' (s is the last register flown to r), or goes to t_i in position j producing $v' \in B^*$ if $\sigma_j(r)$ contains the factor $sv't$ (t is the next

register flown to r). By iterating this process again, we exhaust the registers flown to r , produces their content meanwhile. Finally, the run ρ ends in position $j + 1$ with state r_o and has produced $\nu_{j+1}(r) = \nu_j(\sigma_j)(r)$, proving the claim. \triangleleft

Coming back to the proof of correctness, we have shown that for all positions $j \geq 0$, \mathcal{F} has an initial run on $\llbracket \mathcal{D} \rrbracket(u)$ reaching position j in state out_o and producing $\nu_j(\text{out})$. This proves that $\llbracket \mathcal{D} \rrbracket(u)$ is in the domain of \mathcal{F} (the maximal initial run of \mathcal{F} on $\llbracket \mathcal{D} \rrbracket(u)$ reads the whole word and \mathcal{F} accepts only if \mathcal{T} produces infinitely often) and $\llbracket \mathcal{F} \rrbracket(\llbracket \mathcal{D} \rrbracket(u)) = \bigsqcup_{j \geq 0} \nu_j(\text{out}) = \llbracket \mathcal{T} \rrbracket(u)$. Therefore, $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{F} \rrbracket \circ \llbracket \mathcal{D} \rrbracket$.

Finally, we construct a gp2RT \mathcal{D}' equivalent to \mathcal{D} by Theorem 3 and we obtain the desired gp2RT $\mathcal{S} = \mathcal{F} \circ \mathcal{D}'$ by Theorem 2. \blacktriangleleft

7 From gp2DT to gpSST

The construction presented in this section is the most involved of the paper. It is adapted from [14]. Given a deterministic two-way transducer, we construct a gpSST that realizes the same function. Here again, the main complications from infinite words are dealing with the acceptance condition and the impossibility to get the final configuration of the run.

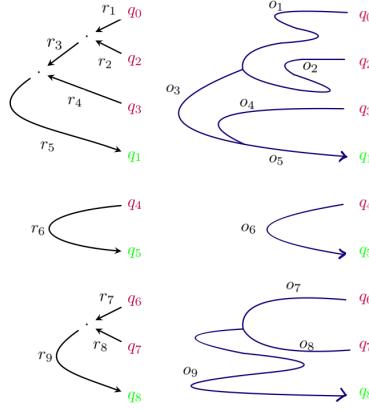
► **Theorem 6.** *Given a gp2DT \mathcal{T} with $n > 0$ states and k coloring functions over ℓ colors, we can construct a gpSST \mathcal{S} with $O(\ell^{kn}(2n)^{2n})$ states, $2n - 1$ registers and k coloring functions over ℓ colors such that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{S} \rrbracket$.*

Sketch of proof. We improve on the classical Shepherdson construction [19] from two-way machines to one-way. In this construction, the one-way machine computes information about the runs of the two-way machine on the prefix read up to the current position. More precisely, it stores the state reached on reading the prefix starting at the initial state, as well as a succinct representation of the information about all possible partial runs occurring on the prefix later on. In the following we call these partial runs *right-right runs*, as they both enter and exit the word from the right of the word. Further, by associating a register to each run in this representation, we can construct an SST equivalent to the two-way machine.

Upon reading some letter $a \in A$, the prefix w we are interested in grows to wa , and consequently, we have to update the information about the right-right runs. While some right-right runs on the prefix w may be extended to right-right runs on wa , some runs (which cannot be extended) may die, and hence needs to be removed. A third possibility is that, upon reading a letter a , some right-right runs may merge. This implies that the construction would not be copyless, as if two right-right runs over ua are the extension of a same right-right run over u , the register storing the production of the run over u needs to be copied in both runs over ua .

In order to compute a copyless SST, we improve this construction by refining the information stored by the one-way machine: it stores not only the set of right-right runs, but also whether they merge and the respective order of the merges. Essentially, the latter representation keeps track of the *structure* of the right-right runs on the prefix read up to the current position, as well as the *output* generated by these runs. The resulting information can be represented as a forest, which is a (possibly empty) set of trees. Then we associate a register to each edge of the forest, so that the update function can be made copyless. The number of registers required is still linear in the number of states. \blacktriangleleft

Formally, we call the structure used to model the right-right runs *merging forests*, which we define as follow:



■ **Figure 3** Example of a merging forest (on the left) corresponding to right-right runs (on the right) of a two-way machine for some prefix u of an input word.

► **Definition 7.** Given a set of states $Q = Q^+ \uplus Q^-$, a set of coloring functions χ and an integer $\ell > 0$, we define the merging forests on (Q, χ, ℓ) , denoted \mathcal{MF} , as the set of forests F such that:

- the leaves of all trees of F are labeled by distinct elements of Q^- ,
- the roots of all trees of F are labeled by distinct elements of Q^+ ,
- all unary nodes (exactly one child) are roots.

Abusing notations, the leaves are also labeled by a χ -tuple of integers less than ℓ , i.e. a color for each coloring function.

Informally, an element $F \in \mathcal{MF}$ describes a set of right-right runs, such that if q is the root of a tree and p is one of its leaves, then (p, q) is a right-right run. Notice that, if two leaves x and y belong to the same tree, then the right-right runs starting in x and y will merge. The structure of the tree reflects the order in which the runs sharing the same root merge. The integer labels of leaves serve as coloring for the conjunction of parity acceptance condition. An example of a merging forest is depicted in Figure 3.

Note that in Figure 3, the states in Q^- are depicted in purple, while the states in Q^+ are depicted in green. The forest comprises of 3 trees - one with root q_1 and leaves q_0, q_2 and q_3 , the second with root q_5 and leaf q_4 , and the third with root q_8 and leaves q_6 and q_7 . For each tree, there are right-right runs from its leaves to the root. For instance, from the merging forest in Figure 3, we can infer that there are three right-right runs entering u on the right in states q_0, q_2 and q_3 respectively, and emerging out of u in state q_1 , moreover the run starting from q_0 first merges with the run starting from q_2 and these two runs then merge with the one starting from q_3 . The figure also depicts the register corresponding to the edges of the forest. Further, the output generated by the part of the run highlighted in yellow is stored in the register r_1 , the output for the part highlighted in green is stored in r_2 , blue in r_3 and red in r_5 .

We are now ready to give the formal construction of Theorem 6. We defer the proof of correctness to Appendix.

Proof of Theorem 6. Given a gp2DT $\mathcal{T} = (Q, A, \delta, q_0, \chi, B, \lambda)$, we construct a gpSST $\mathcal{S} = (Q', A, \alpha, q'_0, \chi', B, R, \text{out}, \beta)$ such that:

- $Q' = Q \times \mathcal{MF}$,
- $q'_0 = (q_0, F_0)$ where F_0 is the forest having no internal nodes, and edges from a leaf u labeled $p \in Q^-$ to a root v labeled $q \in Q^+$ if $\delta(p, \vdash) = q$.

- R is a set of registers of size $2|Q| - 1$ with a distinguished register `out`.
- the coloring functions $\chi' = \{c' \mid c \in \chi\}$ are described below.
- the definitions of α and β are more involved and given below.

For each merging forest $F \in \mathcal{MF}$, we fix a map ξ_F associating distinct registers from $R \setminus \{\text{out}\}$ to edges of F . This is possible since the number of edges in F is at most $2|Q| - 2$ (see Lemma 11).

For simplicity sake, we assume that for each edge (u, v) of F_0 corresponding to transition $\delta(p, \vdash) = q$, the register $\xi_{F_0}(u, v)$ is initialized with the production $\lambda(p, \vdash)$. Note that considering initialized registers does not add expressiveness, as it could be simulated using a new unreachable initial state. Other registers are initially empty.

The definitions of the transition function α and the update function β are intertwined. Let $(q, F) \in Q \times \mathcal{MF}$ be a state of \mathcal{S} and a a letter of A . We describe the state $(p, F') = \alpha((q, F), a)$. First we construct an intermediate *graph* G that does not satisfy the criteria of the merging forests, then explain how G is transformed into a merging forest $F' \in \mathcal{MF}$. The steps of the construction are depicted in Figure 4.

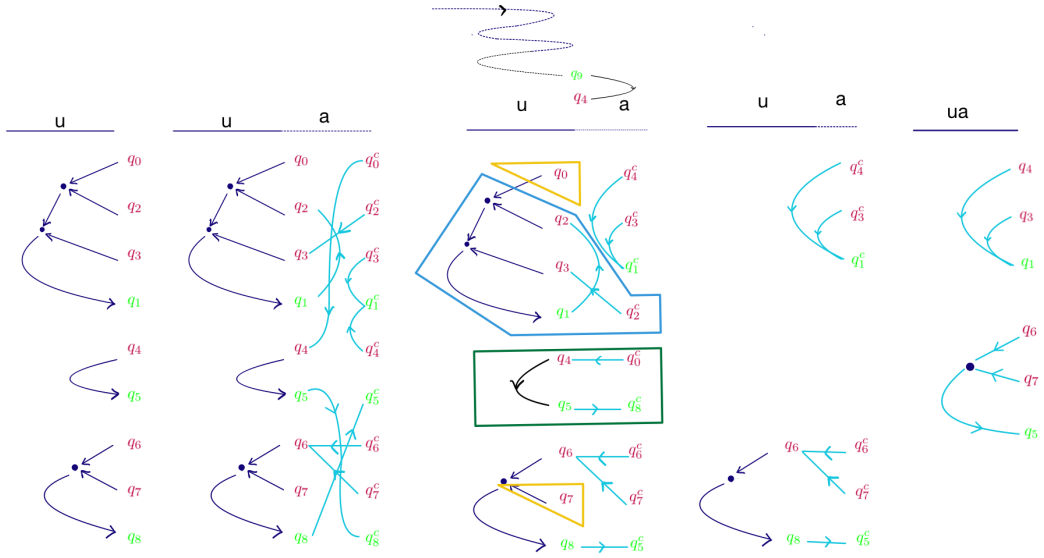
Construction of G . The graph G is built from F as follows. First, we add new isolated nodes $Q_c = \{q_c \mid q \in Q\}$ to the forest F . These nodes will serve as the roots and leaves of F' . For each transition $\delta(p, a) = q$, we will add an edge connecting these new nodes and the roots and leaves of F . We also extend the labelling ξ_F to a labelling ξ_G by adding productions $\lambda(p, a)$ to the new edges of G .

Let $p \in Q$ and let $q = \delta(p, a)$. If $p \in Q^+$ is the label of the root u of some tree in F , we add an edge from u to either q_c if $q \in Q^+$, or to v if $q \in Q^-$ and there exists a leaf v labeled q in F . If $p \in Q^-$, we add an edge from p_c to either q_c if $q \in Q^+$, or to v if $q \in Q^-$ and there exists a leaf v labeled q in F . The added edge is labeled $\lambda(p, a)$ by ξ_G . The construction is illustrated in the first two steps of Figure 4. Note that G may now have cycles.

The new state p . To compute the first component of the new state of \mathcal{S} , let $r = \delta(q, a)$. If r belongs to Q^+ , then $p = r$ and $\lambda(q, a)$ is appended to the output register: $\beta((q, F), a)(\text{out}) = \text{out} \cdot \lambda(q, a)$. Moreover, the colors are directly inherited: $\bar{c}((q, F), a) = c(q, a)$ for all $c \in \chi$. Otherwise, assume that there is a leaf u in F labeled $r \in Q^-$ (if not, the transition is undefined). We consider the maximal path in G starting from u . If this path is looping or if it ends in a root of F then the transition is undefined. Otherwise, it ends in a new node v of G . Let $p \in Q^+$ be the state such that $v = p_c$. We append to the `out` register first $\lambda(q, a)$ and then the ξ_G labels of the edges of the path from u to v in G , in the order of the path. These ξ_G labels are either registers given by ξ_F for edges of F , or local outputs of the form $\lambda(x, a)$ for the new edges, i.e., those in $G \setminus F$. Moreover, for each $c \in \chi$, we let $c'((q, F), a)$ be the minimum of (1) the c -values of the labels of leaves of F appearing in the path from u to v in G , and (2) the c -values of the transitions used to create the new edges of G in this path.

The third figure of Figure 4 illustrates the case where $\delta(q, a) = q_4 \in Q^-$. We then look at the path starting in q_4 , which leads to the state $(q_8)_c$ after reading a .

Construction of F' and the update of registers other than `out`. This is illustrated by the third and fourth figures of Figure 4. We erase all nodes (and adjacent edges) that are on a cycle of G since such cycles cannot be part of an accepting run of \mathcal{T} (see the part of G boxed in blue). The resulting graph is now a acyclic. We also erase all nodes and edges which are not on a path from a leaf in Q_c^- to a root in Q_c^+ since they cannot be part of a right-right run on ua (see the parts boxed in yellow). Finally, we erase the tree with root $v = p_c$ in the



■ **Figure 4** Illustration of the procedure to calculate F' from F and a letter a . The first figure is F , the second is the graph G built from F and the transition occurring at a . The third figure illustrates the trimming done from G to obtain a forest depicted in the fourth figure. Finally, we merge unary internal nodes to obtain the merging forest of the last figure.

second case of the definition of p above. Indeed, should any right-right run simulated by this tree appear later, the resulting run would loop on a finite prefix of the input (see the part boxed in green).

The resulting forest has leaves in Q_c^- and roots in Q_c^+ . Each remaining new leaf or root $q_c \in Q_c$ is labeled q , i.e., by dropping the c index. A new leaf $q_c \in Q_c^-$ is labeled by a χ -tuple $(m_c)_{c \in \mathcal{X}}$ of integers less than ℓ where m_c is the minimum of (1) the c -component of the labels of leaves of F appearing in the branch in G from q_c to its root, and (2) the values $c(p, a)$ of the transitions used to create the new edges of G in this branch. We forget the initial labeling of leaves and roots of F .

To obtain a merging forest, it remains to remove unary internal nodes. This is shown between the fourth and fifth figures of Figure 4. We replace each *maximal* path $\pi = u_0, u_1, u_2, \dots, u_{n-1}, u_n$ ($n \geq 1$) with u_1, \dots, u_{n-1} unary nodes by a single edge $e = (u_0, u_n)$. We obtain the merging forest F' . The update function is simultaneously defined by

$$\beta((q, F), a)(\xi_{F'}(e)) = \xi_G(u_0, u_1)\xi_G(u_1, u_2) \cdots \xi_G(u_{n-1}, u_n).$$

Notice that for each remaining edge f of G we have either $f \in F$ and $\beta((q, F), a)(\xi_{F'}(f)) = \xi_G(f) = \xi_F(f)$ or f is a new edge and $\beta((q, F), a)(\xi_{F'}(f))$ is set to some $\lambda(s, a)$. Since each edge f of F contributes to at most one edge e of F' , or to the path from u to $v = p_c$ which flows into the out register (but not both), it implies that the substitution $\beta((q, F), a)$ is copyless. ◀

8 Continuity and topological closure of a gp2DT

The classical topology on infinite words (see e.g. [18]) defines the distance between two infinite words u and v as $d(u, v) = 2^{-|u \wedge v|}$, where $u \wedge v$ is the longest common prefix of u and v . Then a function $f: A^\omega \rightarrow B^\omega$ is continuous at $x \in \text{dom}(f)$ if

$$\forall i \geq 0, \exists j \geq 0 \forall y \in \text{dom}(f), |x \wedge y| \geq j \implies |f(x) \wedge f(y)| \geq i$$

A function f is continuous if it is continuous at every $x \in \text{dom}(f)$. We refer to [15] for more details.

Since a gp2DT is in particular deterministic, it realizes a continuous function. Indeed, the longer two input words share a common prefix, the longer their output will also do.

By comparison, a non-deterministic transducer can make choices depending on an infinite property of the input, e.g. whether there is an infinite number of as , and thus realize a noncontinuous function.

The question of characterizing the continuous functions realizable by transducers was studied in [5, 6]. In [5], it was proved that for any continuous function realized by a non-deterministic one-way transducer T , there exists a deterministic two-way transducer S such that for any $u \in \text{dom}(T)$, $\llbracket T \rrbracket = \llbracket S \rrbracket$. This means that if a non-deterministic one-way transducer realizes a continuous function, although it can use the non-determinism to refine its domain, the continuity property forbids it from producing non-deterministically.

In [6], it was conjectured that the class of deterministic regular functions, i.e. functions defined by deterministic two-way transducers with a Büchi acceptance condition, corresponds to the class of functions realized by a non-deterministic two-way transducer, called regular functions, that are continuous.

Since the class of gp2DT is strictly more expressive than the class of deterministic regular functions of [6], but still realize continuous functions, the conjecture of [6] fails. One can for example consider the function f such that $f(u) = u$ if u has a finite number of as and is undefined otherwise. Then f is continuous, as it is continuous on its domain, but it cannot be realized by a deterministic two-way transducer with a Büchi condition.

However, the refinement here only acts on the domain of the function, and not the production. Indeed, let T be a gp2DT which realizes a function f . We can define the topological closure of $\text{dom}(T)$, which we denote $\widehat{\text{dom}(T)}$ as the words u such that there exists a sequence $(u_i)_{i \geq 1}$ where for all i , $u_i \in \text{dom}(T)$ and $\forall n, \exists i \forall j \geq i, |u \wedge u_j| \geq n$

We say that the sequence $(u_i)_{i \geq 1}$ converges to u . Note that if two sequences $(u_i)_{i \geq 1}$ and $(v_j)_{j \geq 1}$ belong to $\text{dom}(T)$ and converge to the same word w , then the elements will share longer and longer prefixes. Since T is deterministic, both sequences will then also produce words that share longer and longer prefixes. Thus we can define \hat{f} , whose domain is $\widehat{\text{dom}(T)}$, where $\hat{f}(u)$ is the limit of the images of any sequence $(u_i)_{i \geq 1}$ that belong to $\text{dom}(T)$ and which converges to u . The function \hat{f} is in fact realized by the transducer T where the accepting condition is dropped. The domain of its underlying automaton is then a closed set in the classical topology over infinite words, as it is recognized by a deterministic Büchi automaton where all transitions are final (see [18, Proposition 3.7, p. 147]). Note however that since the semantics of our transducers requires an input to produce an infinite word to be in the domain of the transducer, the domain of \hat{f} might not be a closed set.

Reversible two-way transducers with no accepting condition

Following this, let us consider the class of reversible transducers with no accepting condition (2RT), which is equivalent to saying all transitions are final within a Büchi condition.

The constructions presented in this article get simpler, with a better complexity:

► **Theorem 8.** *Given a deterministic two-way transducer T with n states and no accepting condition, we can construct a 2RT S with $O(n^{4n+1})$ states such that $\llbracket T \rrbracket = \llbracket S \rrbracket$.*

Proof. Let T be a deterministic two-way transducer with n states. Since there is no accepting condition, the construction from the proof of Theorem 6 drops the arrays of integers from the merging forest. Then the number of merging forests is in $O(n^{2n})$, and applying Theorem 6 results in an SST T' with $O(n^{2n})$ states and $2n$ variables. Hence, by applying Theorem 4 to T' , we get an equivalent 2RT S with $O(n^{4n+1})$ states. ◀

Surprisingly, the class of 2RT has the expressive power of deterministic Büchi two-way transducers. This is due to the fact that for an input to be accepted, it requires the corresponding output to be infinite. We can *hide* the Büchi acceptance condition in this restriction. The following theorem proves that starting from a reversible Büchi transducer, we can construct one which does not use any acceptance condition. To notice that reversible and deterministic Büchi transducers have the same expressive power, one can rely on our main theorem, specializing it to a Büchi condition. Indeed, a Büchi acceptance condition corresponds to a unique coloring function associating 0 to accepting transitions and 1 otherwise. Then if we are given a deterministic Büchi two-way transducer, we are able to produce a reversible one using a coloring function on the same domain, hence a reversible Büchi two-way transducer.

► **Theorem 9.** *Let T be a reversible Büchi two-way transducer with n states. We can construct an equivalent 2RT S with $3n$ states such that $\llbracket T \rrbracket = \llbracket S \rrbracket$.*

Proof. Let T be a reversible Büchi two-way transducer with n states. We define the 2RT S similarly to T , but with three *modes* of operation (and hence thrice the states): *simulation*, *rewind* and *production*. The transducer S starts by simulating T without producing anything. Upon reaching an accepting transition t , it switches to rewind mode to and unfolds the run of T back to the previous accepting transition (or the start of the run). It finally follows the run of T and produces the corresponding output up to seeing the accepting transition t , at which point it restarts the simulation.

Then for a given word u , if u belongs to the domain of T , the run of T over u will see accepting transitions infinitely often, and hence S will go to production mode infinitely often too. Conversely, if u does not belong to the domain of T , it means that either the run of T over u only sees a finite number of accepting transitions, or does not produce an infinite word. In the latter case, S will also produce a non finite word. In the former case, S will remain in simulation mode and never produce anything, and thus u will not be in the domain of S .

We can remark that S is reversible if T is since first within modes, transitions of S are transitions of T , and secondly, the transitions from one mode to another happen on every accepting transition. Hence transitions that come to a given mode from another are exactly the ones that exit it, so two transitions cannot go to a same state upon reading the same letter. ◀

9 Conclusion

The main contribution of this paper is the result which shows that deterministic two-way transducers over infinite words with a generalized parity acceptance condition are reversible. We also show that reversible two-way transducers over infinite words are closed under composition. Our results can help in an efficient construction of two-way reversible transducers from specifications presented as RTE [16] or SDRTE [13] over infinite words. Earlier work [11] in this direction on finite words relied on an efficient translation from non-deterministic transducers used in parsing specifications to reversible ones; our results can hopefully help extend these to infinite words.

References

- 1 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 65–74. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.18.

- 2 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 9:1–9:10. ACM, 2014.
- 3 Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. In Mizuho Hoshi and Shinnosuke Seki, editors, *22nd International Conference on Developments in Language Theory, DLT 2018*, volume 11088 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 2018.
- 4 Udi Boker. Why These Automata Types? In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 143–163. EasyChair, 2018. doi:10.29007/C3BJ.
- 5 Olivier Carton and Gaëtan Douéneau-Tabot. Continuous rational functions are deterministic regular. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPICs*, pages 28:1–28:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.MFCS.2022.28.
- 6 Olivier Carton, Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Sarah Winter. Deterministic regular functions of infinite words. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 121:1–121:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.121.
- 7 Arthur Cayley. A theorem of trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889. URL: <https://books.google.fr/books?id=M7c4AAAAIAAJ>.
- 8 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized Parity Games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures*, pages 153–167, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-71389-0_12.
- 9 Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.
- 10 Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On reversible transducers. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 113:1–113:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.113.
- 11 Luc Dartois, Paul Gastin, R. Govind, and Shankara Narayanan Krishna. Efficient construction of reversible transducers from regular transducer expressions. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 50:1–50:13. ACM, 2022. doi:10.1145/3531130.3533364.
- 12 Luc Dartois, Paul Gastin, Loïc Germerie Guizouarn, R. Govind, and Shankaranarayanan Krishna. Reversible transducers over infinite words, 2024. arXiv:2406.11488.
- 13 Luc Dartois, Paul Gastin, and Shankara Narayanan Krishna. Sd-regular transducer expressions for aperiodic transformations. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470738.
- 14 Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic string transducers. In *Developments in Language Theory - 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings*, pages 125–137, 2016. doi:10.1007/978-3-662-53132-7_11.
- 15 Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Synthesis of computable regular functions of infinite words. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/LMCS-18(2:23)2022.

- 16 Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular Transducer Expressions for Regular Transformations. In Martin Hofmann, Anuj Dawar, and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic In Computer Science (LICS'18)*, pages 315–324, Oxford, UK, July 2018. ACM Press.
- 17 Joost Engelfriet and Hendrik Jan Hooeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001. doi:10.1145/371316.371512.
- 18 Dominique Perrin and Jean-Eric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*, volume 141. Elsevier, 2004.
- 19 John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2):198–200, 1959.

A Appendix for Section 5

We give here the formal construction as well as the proof of correctness for Theorem 3:

Proof of Theorem 3. Let $\mathcal{T} = (Q, A, \delta, q_0, \chi, B, \lambda)$ be a gpDT. We begin by showing how to construct a gp2RA \mathcal{A}' accepting the same ω -language as \mathcal{A} .

Fix an arbitrary total order \preceq over Q . For $q \in Q$ and $a \in A$ we let $\delta_a^{-1}(q) = \{p \in Q \mid \delta(p, a) = q\}$, and we also set $\delta_a^{-1}(q) = \emptyset$, except if $q \neq q_0$ (it is undefined otherwise). Let $\mathcal{S}_a(q, q')$ be a predicate, true if q' is minimal with respect to \prec such that $q \prec q'$ and $\delta(q, a) = \delta(q', a)$. Let $\overline{Q} = \{\overline{q} \mid q \in Q\}$ and $\underline{Q} = \{\underline{q} \mid q \in Q\}$ be two copies of Q . Define $\mathcal{A}' = (Q', A, \delta, q'_0, \chi')$ by

- $Q' = Q'^+ \uplus Q'^- = ((\underline{Q} \cup \overline{Q}) \times (\underline{Q} \cup \overline{Q})) \setminus \{(\underline{q}, \underline{q}), (\overline{q}, \overline{q}) \mid q \in Q\}$ with $q'_0 = (q_0, \overline{q_0})$ and
 - $Q'^+ = (\underline{Q} \times \overline{Q}) \cup (\overline{Q} \times \underline{Q})$,
 - $Q'^- = ((\underline{Q} \times \underline{Q}) \cup (\overline{Q} \times \overline{Q})) \setminus \{(\overline{q}, \overline{q}), (\underline{q}, \underline{q}) \mid q \in Q\}$.

In a state $(r, s) \in Q'$, the first (resp. second) component is for the head which is ‘above’ (orange line) (resp. ‘below’ (blue line)) the accepting run (black straight line) in Figure 2a. In both cases, a state $\underline{q} \in \underline{Q}$ (resp. $\overline{q} \in \overline{Q}$) means that the corresponding head (colored line) is above (resp. below) the state.

- Transitions: first two cases for Q'^+ states and then for Q'^- states

$$1. \quad \delta'((\underline{p}, \overline{q}), a) = \begin{cases} (\overline{p'}, \overline{q}) & \text{if } \mathcal{S}_a(p, p') \text{ for some } p' \in Q \\ (\underline{p}, \underline{q'}) & \text{elseif } \mathcal{S}_a(q', q) \text{ for some } q' \in Q \\ (\underline{\delta(p, a)}, \overline{\delta(q, a)}) & \text{otherwise.} \end{cases}$$

$$2. \quad \delta'((\overline{p}, \underline{q}), a) = \begin{cases} (\underline{p'}, \underline{q}) & \text{if } \mathcal{S}_a(p', p) \text{ for some } p' \in Q \\ (\overline{p}, \overline{q'}) & \text{elseif } \mathcal{S}_a(q, q') \text{ for some } q' \in Q \\ (\overline{\delta(p, a)}, \underline{\delta(q, a)}) & \text{otherwise.} \end{cases}$$

$$3. \quad \delta'((\overline{p}, \overline{q}), a) = \begin{cases} (\underline{p}, \overline{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \\ (\overline{p}, \underline{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\min \delta_a^{-1}(p), \min \delta_a^{-1}(q)) & \text{otherwise.} \end{cases}$$

$$4. \quad \delta'((\underline{p}, \underline{q}), a) = \begin{cases} (\overline{p}, \underline{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \\ (\underline{p}, \overline{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\max \delta_a^{-1}(p), \max \delta_a^{-1}(q)) & \text{otherwise.} \end{cases}$$

- $\chi' = \{c' \mid c \in \chi\}$ with $c'((r, s), a) = \begin{cases} c(q, a) & \text{if } (r, s) = (\underline{q}, \overline{q}) \\ \max\{c(p, a) \mid p \in Q, a \in A\} & \text{otherwise.} \end{cases}$

We claim that \mathcal{A}' is reversible. From the definition of δ' , \mathcal{A}' is clearly deterministic. We show that \mathcal{A}' is also co-deterministic. There are three potential transitions leading to a given state in Q' by reading a given letter, only one of which can be part of δ' .

The following case analysis shows this:

$$\delta_a'^{-1}((\underline{p}, \bar{q})) = \begin{cases} (\bar{p}, \bar{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \\ (\underline{p}, \underline{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\overline{\max \delta_a^{-1}(p)}, \underline{\min \delta_a^{-1}(q)}) & \text{otherwise} \end{cases}$$

$$\delta_a'^{-1}((\bar{p}, \underline{q})) = \begin{cases} (\underline{p}, \underline{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \text{ (so } p \neq q_0) \\ (\bar{p}, \bar{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\underline{\min \delta_a^{-1}(p')}, \overline{\max \delta_a^{-1}(q')}) & \text{otherwise} \end{cases}$$

$$\delta_a'^{-1}((\bar{p}', \bar{q}')) = \begin{cases} (\underline{p}, \bar{q}') & \text{if } \mathcal{S}_a(p, p') \text{ for some } p \in Q \\ (\bar{p}', \underline{q}) & \text{elseif } \mathcal{S}_a(q, q') \text{ for some } q \in Q \\ (\delta(p', a), \delta(q', a)) & \text{otherwise.} \end{cases}$$

$$\delta_a'^{-1}((\underline{p}', \underline{q}')) = \begin{cases} (\bar{p}, \underline{q}') & \text{if } \mathcal{S}_a(p', p) \text{ for some } p \in Q \\ (\underline{p}', \bar{q}) & \text{elseif } \mathcal{S}_a(q', q) \\ (\delta(p', a), \delta(q', a)) & \text{otherwise.} \end{cases}$$

We conclude that \mathcal{A}' is co-deterministic, and therefore reversible.

Intuitively, automaton \mathcal{A}' will follow the run of \mathcal{A} , adding extra steps to deal with the states that are co-reachable from states of this run. States of \mathcal{A}' of the form (\underline{q}, \bar{q}) correspond to states q in the run of \mathcal{A} . The key idea behind the construction of \mathcal{A}' is that in a run of this automaton, configurations of the form $\vdash u(\underline{q}, \bar{q})v$ will occur in the same order as the configurations $\vdash uqv$ in the run of \mathcal{A} . This is the point of the following claim.

▷ **Claim 10.** Let $\rho = \vdash u_0 q_0 v_0 \rightarrow \vdash u_1 q_1 v_1 \rightarrow \vdash u_2 q_2 v_2 \rightarrow \dots$ be an accepting run of \mathcal{A} on $w \in A^\omega$ (we have $w = u_i v_i$ for all $i \geq 0$ where u_i is the prefix of length i of w). There is an accepting run ρ' of \mathcal{A}' on w such that the projection of ρ' on the configurations with states of the form (\underline{p}, \bar{p}) is $\vdash (q_0, \bar{q}_0)w \xrightarrow{\pm} \vdash u_1(q_1, \bar{q}_1)v_1 \xrightarrow{\pm} \vdash u_2(q_2, \bar{q}_2)v_2 \xrightarrow{\pm} \dots$.

Proof. Let G be the configuration graph of \mathcal{A} on the input word $w \in A^\omega$. The vertices of G are all configurations $\vdash uqv$ with $w = uv$, $u \in A^*$ and $q \in Q$. Edges correspond to transitions: we have an edge $\vdash upav \rightarrow \vdash uaqv$ if $\delta(p, a) = q$. Since \mathcal{A} is deterministic, there is at most one outgoing edge from each configuration and since \mathcal{A} is one-way, thus the graph G is acyclic.

Let T be the connected component of G that contains the initial configuration $\vdash q_0 w$. Note that the run ρ corresponds to the only infinite path in G starting from $\vdash q_0 w$. Any configuration $\vdash u_i p_i v_i$ of T which is not on ρ ($p_i \neq q_i$) will eventually merge with ρ : $\vdash u_i p_i v_i \xrightarrow{*} \vdash u_{j-1} p_{j-1} v_{j-1} \rightarrow \vdash u_j q_j v_j$ with $i < j$ and $p_{j-1} \neq q_{j-1}$. We say that $\vdash u_i p_i v_i$ is below (resp. above) ρ if $p_{j-1} \prec q_{j-1}$ (resp. $q_{j-1} \prec p_{j-1}$).

Let us consider the run ρ' of \mathcal{A}' on w . Due to the definition of the transition function of \mathcal{A}' , the run ρ' only moves along T . Indeed a configuration $\vdash u(r, s)v$ of ρ' encodes the position of two tokens, each placed either above or below a configuration of T . Moreover, the first token is always above the branch ρ while the second token is always below. This can be shown by case analysis of δ' . The two transitions where the upper token goes from above a branch to below are the following:

- $\delta'((\underline{p}, \bar{q}), a) = ((\overline{p'}, \bar{q}))$ if $(\mathcal{S}_a(p, p'))$: there, we know that $p \prec p'$, so the token ends up on a branch that is above where it was;
- $\delta'((\underline{p}, \bar{q}), a) = (\overline{p}, \bar{q})$ if $\delta_a^{-1}(p) = \emptyset$ and $a \neq \vdash$, so p must have reached the end of the branch it was placed on, and we know it was not placed on ρ , because when this branch ends, $a = \vdash$.

A similar observation can be made for transitions where the lower token goes from below a branch to above.

We denote by T_i the subtree of T containing all configurations having a path to $\vdash u_i q_i v_i$. We aim to prove that ρ' reaches the position i , and the first time it does is in state $(\underline{q}_i, \bar{q}_i)$.

We remark that for every cases 1 to 4 of the transition function, as long as the transition function is defined the run ρ' can continue. As we only visit configurations of T , as long as ρ is infinite, as assumed by Claim 10, so is ρ' .

Next, as \mathcal{A}' is reversible, it cannot loop, as it would require two different configuration to go to the same one to enter the loop, which would break co-determinism. So ρ' starts in T_i , does not stop nor loops, so since T_i is finite, ρ' has to end up leaving T_i . So ρ' reaches position i , while only moving along T_i . As $(\underline{q}_i, \bar{q}_i)$ is the only possible state where ρ' follows T_i while having its first token above ρ and the second below ρ , ρ' first reaches i in state $(\underline{q}_i, \bar{q}_i)$.

As this is true for any position i , and since T_i contains T_{i-1} , we can conclude the proof of Claim 10. \triangleleft

Based on this claim, we show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Let w be an infinite word accepted by \mathcal{A} , and ρ be the accepting run of \mathcal{A} on w . We showed that the configurations $\vdash u q v$ happen in the same order in ρ as configurations of the form $\vdash u (\underline{q}, \bar{q}) v$ in ρ' , the run on w of \mathcal{A}' . Moreover $c'((\underline{p}, \bar{p}), a) < c'(s, a)$ for all $a \in A$ and for all s of another form, and as $|\text{inf}(\rho)| > 0$ (because w is infinite and \mathcal{A} has a finite number of states), $\min\{t | t \in \text{inf}(\rho)\} = \min\{t | t \in \text{inf}(\rho')\}$. So $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

Let $\mathcal{T}' = (Q', A, \delta', (\underline{q}_0, \bar{q}_0), (c_i)_1^k, B, \lambda')$ be the gp2RT having \mathcal{A}' as underlying automaton, with

$$\lambda'(s, a) = \begin{cases} \lambda(q, a) & \text{if } s = (\underline{q}, \bar{q}) \\ \varepsilon & \text{if } s \text{ is of another form.} \end{cases}$$

Because we showed that states of the form (\underline{q}, \bar{q}) are met in the run of \mathcal{A}' on a given word in the same order as states q in the run of \mathcal{A} on the same word, the output of \mathcal{T}' is the same as the output of \mathcal{T} . So we have that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{T}' \rrbracket$.

Finally, $|Q'| = 4|Q|^2$, justifying the complexity. \blacktriangleleft

B Appendix for Section 6

In this section, we prove the reversibility of the transducer \mathcal{F} and the correctness of the construction given in the proof of Theorem 4.

Proof of Theorem 4. We begin by showing that \mathcal{F} is reversible, before showing that the formal construction from the earlier sketch of proof is correct.

Reversibility of \mathcal{F} . The transducer \mathcal{F} is clearly deterministic by construction. Let us prove that it is co-deterministic. To this end, let s_i be a state and σ a substitution. Looking at the transition function, its antecedent $\alpha^{-1}(s_i, \sigma)$ is either r_i if $\sigma(r)$ starts with vs for some word $v \in B^*$, or r_o if there is some register t that contains rvs for some word $v \in B^*$. Since we

only consider copyless substitutions, there is at most one register that contains s . The two options are then mutually exclusive, as one requires that s be the first register to appear, and the second requires that there is a register before s .

The proof for a state s_o is similar. The antecedent $\alpha^{-1}(s_o, \sigma)$ is either s_i if $\sigma(s)$ contains no register, or r_o if r is the last register appearing in $\sigma(s)$. Since these two are mutually exclusive, we get that \mathcal{F} is reversible. \blacktriangleleft

C Appendix for Section 7

The following lemma relies on Cayley's Formula to count the number of merging forests.

► **Lemma 11.** *Let $Q = Q^+ \uplus Q^-$ be of size $n > 0$ with $Q^+ \neq \emptyset$, χ of size $k \geq 0$ and $\ell > 0$. Then each element F of \mathcal{MF} has at most $2n - 2$ nodes, $2n - 2$ edges; and \mathcal{MF} itself is of size at most $\ell^{k(n-1)}(2n-1)^{2n-3}$.*

Proof. We first compute the maximal number of nodes and edges in a nonempty forest F of \mathcal{MF} . Note that, since F is nonempty, both Q^+ and Q^- must be nonempty. Let $\text{nbe}(t)$ and $\text{nbl}(t)$ be the number of edges and leaves of the tree t . If t has no unary nodes, then $\text{nbe}(t) \leq 2\text{nbl}(t) - 2$. Since in a forest $F \in \mathcal{MF}$, all unary nodes are roots, it follows that $\text{nbe}(t) \leq 2\text{nbl}(t) - 1$ for all trees $t \in F$. Also, the number of nodes of a tree is $\text{nb}(t) = 1 + \text{nbe}(t)$. We deduce that

$$\begin{aligned} \text{nbe}(F) &= \sum_{t \in F} \text{nbe}(t) \leq \sum_{t \in F} 2\text{nbl}(t) - 1 \leq 2|Q^-| - 1 \leq 2n - 3 \\ \text{nb}(F) &= \sum_{t \in F} \text{nb}(t) \leq \sum_{t \in F} 2\text{nbl}(t) \leq 2|Q^-| \leq 2n - 2. \end{aligned}$$

Now we compute the size of the set \mathcal{MF} . Cayley's formula [7] states that the number of non oriented trees with m differently labeled nodes is m^{m-2} . The difference here is that first we deal with forests with at most $2n - 2$ nodes, and secondly only the leaves and roots are labeled. The first point can be dealt with by adding a new node as the root of all trees of the forest, and new nodes if needed to get exactly $m = 2n - 1$ nodes in the tree. For the second point, we can label arbitrarily the remaining nodes. Finally, as each leaf is labeled by a χ -tuple of integers less than ℓ , each tree can appear in \mathcal{MF} up to $(\ell^k)^{|Q^-|}$ many times. The size of \mathcal{MF} is then smaller than $\ell^{k(n-1)}(2n-1)^{2n-3}$. \blacktriangleleft

Proofs for Theorem 4. We can now prove Theorem 4, beginning with the complexity.

Size of \mathcal{S} . Using Lemma 11, we get that $|Q'| \leq n(\ell^{k(n-1)}(2n-1)^{2n-3}) = \mathcal{O}(\ell^{kn}(2n)^{2n})$. Using carefully the registers, we only ever need at most $2n - 1$ registers.

Proof of correctness. First given a finite word w , we say that a right-right run $(x, y) \in Q^- \times Q^+$ of the gp2DT \mathcal{T} on w is useful if there exists an infinite word w' such that the run of \mathcal{T} on ww' is accepting and reaches x on position $|w|$.

We first prove that the state of the gpSST contains all the needed information, then prove that the registers can be used to produce the output. We prove by induction on the size of a word w that the state (q, F) of the constructed gpSST reached after reading w is such that (q_0, q) is a left-right run on w and F contains information about all useful runs on w . Moreover, the out register contains the production of the left-right run (q_0, q) on w and, given a path $\pi = u_0, \dots, u_n$ in F from a leaf u_0 labeled by x to a root u_n labeled by y , the production of the right-right run (x, y) is given by the concatenation of the registers $\xi_F((u_0, u_1)) \dots \xi_F((u_{n-1}, u_n))$.

First, if w is empty, then the initial state is (q_0, F_0) where F_0 describes the set of all right-right runs on \vdash . The register `out` is empty and each tree in the forest F_0 is reduced to a single edge containing the associated production, hence proving the initial case.

Now suppose that the statement holds for some word w and some state (q, F) and let $a \in A$ be a letter. We prove the statement for wa . Let $(p, F') = \alpha((q, F), a)$. If $p = \delta(q, a) \in Q^+$, then (q_0, p) is a left-right run on wa and $\beta((q, F), a)(\text{out}) = \text{out} \cdot \lambda(q, a)$, corresponding to the claim for the left-right run. Otherwise, let $r = \delta(q, a) \in Q^-$. The state p is then described in G as the state reached by the maximal path in G from the leaf u of F labeled by r , to the new node p_c . Using the induction hypothesis, F describes the useful right-right runs on w . Then, following the maximal path from u in G , we see the sequence of right-right runs on w and left-left runs on a , up to the last left-right transition on a leading to state p . This means that we have computed p such that (q_0, p) is the left-right run on wa . We also append to the register `out` all $\xi_G(e)$ for edges e in the path. By induction hypothesis, the registers contain the production of the useful right-right runs on w , and the added edges contains the local production, proving the claim for the left-right run.


We now prove that all useful runs of wa are in F' . Let $(x, y) \in Q^- \times Q^+$ be such a run. Then either $\delta(x, a) = y$ and this edge is added in G and remains in F' , or (x, y) is a sequence starting with a right-left transition over a , then useful right-right runs over w and left-left transitions over a , and finally a left-right transition over a . In the first case, the register $\xi_{F'}((x, y))$ takes the label of G , i.e. the local production $\lambda(x, a)$, satisfying the claim as the path is reduced to a single edge. In the second case, let u_0, \dots, u_n be the path from x to y in G . Each edge (u_i, u_{i+1}) is either a new edge whose label is a local production, or a single edge of F . The associated sequence of registers contains then all the output information of the (x, y) run. When reducing G to F' , as only non branching paths of G can be reduced to a single edge, there is no loss of information. Finally, notice that the edges deleted from G to obtain F' are the ones that are not part of a useful right-right run on wa , or are merging with the left-right run. These latter right-right runs are not useful for wa : they cannot occur anymore in an accepting run of \mathcal{T} since they would induce a loop on a finite prefix of the input. Hence all edges required for the path from x to y appear in F' . Consequently, there is no loss of run nor information, the path from x to y in F' exists and the associated sequence of registers contains the production of the run.

Finally, to prove that both transducers have the same domain, we remark that given the previous induction, if (q, F) is the state reached by \mathcal{S} after reading an input w , then upon reading a letter a , the color of the transition $\alpha((q, F), a) = (p, F')$ is the minimum of the color of all transitions used when extending the left-right run (q_0, q) of \mathcal{T} on w to the left-right run (q_0, p) on wa . Then given an infinite word u , \mathcal{S} has an infinite run on u if and only if \mathcal{T} does, and the minimum of all colors appearing infinitely often is the same on both runs. \blacktriangleleft


An Automata-Based Approach for Synchronizable Mailbox Communication

Romain Delpy 

LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Anca Muscholl 

LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Grégoire Sutre 

LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Abstract

We revisit finite-state communicating systems with round-based communication under mailbox semantics. Mailboxes correspond to one FIFO buffer per process (instead of one buffer per pair of processes in peer-to-peer systems). Round-based communication corresponds to sequences of rounds in which processes can first send messages, then only receive (and receives must be in the same round as their sends). A system is called synchronizable if every execution can be re-scheduled into an equivalent execution that is a sequence of rounds. Previous work mostly considered the setting where rounds have fixed size. Our main contribution shows that the problem whether a mailbox communication system complies with the round-based policy, with no size limitation on rounds, is PSPACE-complete. For this we use a novel automata-based approach, that also allows to determine the precise complexity (PSPACE) of several questions considered in previous literature.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Concurrent programming, Mailbox communication, Verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.22

Related Version *Full Version:* <https://arxiv.org/abs/2407.06968> [6]

Funding This work was (partially) supported by the grant ANR-23-CE48-0005 of the French National Research Agency ANR (project PaVeDyS).

1 Introduction

Message-passing is a key synchronization feature for concurrent programming and distributed systems. In this model, processes running asynchronously synchronize by exchanging messages over unbounded channels. The usual semantics is based on peer-to-peer communication, which is very popular for reasoning about telecommunication protocols. More recently, mailbox communication received increased attention because of its usage in multi-thread programming, as provided by languages like Rust or Erlang. Mailbox communication means that every process has a single incoming communication buffer on which incoming messages from other processes are multiplexed (a mailbox).

Message-passing programs are well-known to be challenging for formal verification since they can easily simulate Turing machines with unbounded channels. Some approximation techniques can help to recover decidability. Among the best known approaches are lossy channel systems [1, 9] and partial-order methods [14]. The latter tightly relate to (high-level) message sequence charts (HMSC), a communication formalism capturing multi-party session types [18, 16, 17]. An HMSC protocol is a graph with nodes labelled by communication scenarios, a.k.a. message sequence charts. Processes still evolve asynchronously, so that the division into nodes cannot be enforced by global synchronization. Such round-based communication is actually quite frequent in distributed computing, for example as building



© Romain Delpy, Anca Muscholl, and Grégoire Sutre;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 22; pp. 22:1–22:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

block in the Heard-Of model [5]. Often a distributed protocol consists of several rounds, where each round first has a phase where processes only send messages, then a phase where they only receive. We refer to such rounds as **sr**-rounds.

Recently **sr**-round-based communication and mailbox communication were considered together in [3]. It turned out that this combination is very attractive for formal verification. The paper [3] proposed a model where **sr**-rounds have fixed size, and showed that control-state reachability in this model becomes decidable (in PSPACE). The question whether a system complies with the **sr**-round model with given round size was shown to be decidable in [7]. It is also known how to decide if a system complies with the **sr**-round model when the round size is not known in advance [12]. All these properties motivate a genuine interest in the **sr**-round model on top of mailbox communication. A bit surprisingly, apart from control-state reachability, similar questions were shown to be undecidable for peer-to-peer communication [10].

In this paper we revisit the framework of [3] and propose an automata-based approach to deal with systems complying with the **sr**-round mailbox model (we refer to this property as *mb-synchronizability*). Importantly, we do not impose any size restriction on the rounds, as in previous works. This makes sense, because even when we can infer an upper bound on the size as in [12], this upper bound is exponential in the number of processes, so its practical use is somewhat limited. We establish that the complexity of all problems listed below is PSPACE-complete for *mb-synchronizable* systems:

- Global-state reachability (Theorem 3.6).
- Model-checking against a reasonable class of regular properties (Theorem 4.3).
- Check if a peer-to-peer system can be simulated as a mailbox system (modulo rescheduling executions, Theorem 4.8).

Our main result is that one can check in PSPACE if a system is *mb-synchronizable* (Theorem 5.16), the complexity being tight. An interesting byproduct of our results is that when we fix the number of processes all the problems above can be solved in PTIME (actually NLOGSPACE).

Comparison with related work. Our technique helps to establish the precise complexity of several problems considered in the papers mentioned above. To be precise, our definition of **sr**-round mailbox model (*mb-synchronizability*) slightly differs from the one used in [3, 7, 12] (but coincides with a variant introduced in [2]). The latter paper uses a partial-order variant of PDL (LCPDL) to show an EXPTIME upper bound for the synchronizability problem for their notion of synchronizability. Using MSO logic and special tree-width, the paper [2] also shows that checking if a system is synchronizable with fixed round size is decidable. Knowing if a round size exists is shown to be decidable with elementary complexity in [12], without exact bounds.

For convenience, technical terms and notations in the electronic version of this manuscript are hyper-linked to their definitions (cf. <https://ctan.org/pkg/knowledge>).

Proofs that are missing in the main text can be found in the full version of the paper [6].

2 Message-passing systems and synchronizability

Throughout the paper, \mathbb{P} denotes a finite non-empty set of *processes*, and \mathbb{M} denotes a finite non-empty set of *message contents*. We consider here peer-to-peer communication between distinct processes. Formally, the set of (communication) *channels* is the set Ch of all pairs $(p, q) \in \mathbb{P} \times \mathbb{P}$ such that $p \neq q$, and the set of (communication) *actions* is

$Act = \{p!q(m), q?p(m) \mid (p, q) \in Ch, m \in \mathbb{M}\}$. An action $p!q(m)$ denotes a *send* by p of message m to q and an action $q?p(m)$ denotes a *receive* by p of message m from q . In both cases, the process performing the action is p . Throughout the paper, we let S and R denote the sets of send actions and receive actions, formally, $S = \{p!q(m) \mid (p, q) \in Ch, m \in \mathbb{M}\}$ and $R = \{p?q(m) \mid (q, p) \in Ch, m \in \mathbb{M}\}$.

A communicating finite state machine [4] is a finite set of processes that exchange messages, each process being given as a finite LTS. Recall that a (finite) *labeled transition system*, *LTS* for short, is a quadruple (L, A, \rightarrow, i) where L is a (finite) set of *states*, A is a finite alphabet, $\rightarrow \subseteq L \times A \times L$ is a set of *transitions*, and $i \in L$ is an *initial* state. We will sometimes consider LTS without initial state. In the following definition, Act_p denotes the set of actions $a \in Act$ performed by p .

► **Definition 2.1** (Communicating Finite-State Machine). *A CFM is a tuple $\mathcal{A} = (\mathcal{A}_p)_{p \in \mathbb{P}}$, where each \mathcal{A}_p is a finite LTS $\mathcal{A}_p = (L_p, Act_p, \rightarrow_p, i_p)$. States in L_p are called local states. The size of \mathcal{A} is defined as $\sum_{p \in \mathbb{P}} (|L_p| + |\rightarrow_p|)$.*

In this paper, we mainly study and compare two semantics of communication: peer-to-peer and mailbox. These two semantics differ in the implementation of the communication network. In the *peer-to-peer semantics*, each channel (p, q) is implemented by a dedicated fifo buffer. This is the classical semantics for [communicating finite-state machines](#) [4]. In the *mailbox semantics*, each process q is equipped with a fifo buffer that acts as a *mailbox*: all messages towards q are enqueued in this buffer. Put differently, the channels (p, q) with same receiver q are multiplexed into a single buffer.

We define both semantics of [CFM](#) jointly, by viewing [channels](#) and [mailboxes](#) as (fifo) message [buffers](#):

► **Definition 2.2** (Process network). *A process network over \mathbb{P} is a pair $\mathcal{N} = (\mathbb{B}, \mathbf{bf})$ where \mathbb{B} is a finite set of fifo buffers and $\mathbf{bf} : Ch \rightarrow \mathbb{B}$ is a map that assigns a [buffer](#) to each [channel](#).*

The [peer-to-peer semantics](#) is induced by the [process network](#) $\mathbf{p2p} = (\mathbb{B}, \mathbf{bf})$ where $\mathbb{B} = Ch$ and \mathbf{bf} is the identity. Here, \mathbb{B} coincides with the set of communication [channels](#). The [mailbox semantics](#) is induced by the [process network](#) $\mathbf{mb} = (\mathbb{B}, \mathbf{bf})$ where $\mathbb{B} = \mathbb{P}$ and $\mathbf{bf}(p, q) = q$. Here, \mathbb{B} is a set of [mailboxes](#), one per process.

► **Remark 2.3.** For both [peer-to-peer semantics](#) and [mailbox semantics](#) we have that the [buffer](#) determines the recipient: $\mathbf{bf}(p, q) = \mathbf{bf}(p', q')$ implies $q = q'$. We call such [process networks](#) *many-to-one*.

Given a [CFM](#) and a [process network](#) we define the associated global transition system:

► **Definition 2.4** (Global transition system). *Let $\mathcal{A} = (\mathcal{A}_p)_{p \in \mathbb{P}}$ be a [CFM](#), and $\mathcal{N} = (\mathbb{B}, \mathbf{bf})$ be a [process network](#) over \mathbb{P} . The global transition system associated with \mathcal{A}, \mathcal{N} is the LTS $\mathcal{T}_{\mathcal{N}}(\mathcal{A}) = (C_{\mathcal{A}}, Act, \rightarrow_{\mathcal{A}}, c_{in})$ with set of configurations $C_{\mathcal{A}} = G \times ((Ch \times \mathbb{M})^*)^{\mathbb{B}}$ consisting of global states $G = \prod_{p \in \mathbb{P}} L_p$ (i.e., products of local states) and [buffer contents](#), with $((\ell_p)_{p \in \mathbb{P}}, (w_b)_{b \in \mathbb{B}}) \xrightarrow{\mathcal{A}} ((\ell'_p)_{p \in \mathbb{P}}, (w'_b)_{b \in \mathbb{B}})$ if*

- $\ell_p \xrightarrow{\mathcal{A}} \ell'_p$ and $\ell_q = \ell'_q$ for $q \neq p$, where p is the process performing a .
- *Send actions:* if $a = p!q(m)$ then $w'_b = w_b((p, q), m)$ and $w'_{b'} = w_{b'}$ for $b' \neq b$, where $b = \mathbf{bf}(p, q)$.
- *Receive actions:* if $a = p?q(m)$ then $((p, q), m)w'_b = w_b$ and $w'_{b'} = w_{b'}$ for $b' \neq b$, where $b = \mathbf{bf}(p, q)$.

The initial configuration is $c_{in} = ((i_p)_{p \in \mathbb{P}}, \varepsilon^{\mathbb{B}})$.

22:4 Synchronizable Mailbox Communication

An *execution* of $\mathcal{T}_{\mathcal{N}}(\mathcal{A})$ is a sequence $\rho = c_0 \xrightarrow{a_1} c_1 \cdots \xrightarrow{a_n} c_n$ with $c_i \in C_{\mathcal{A}}$ such that $c_{i-1} \xrightarrow{a_i}_{\mathcal{A}} c_i$ for every i . The sequence $a_1 \cdots a_n$ is the *label* of the **execution**. The **execution** is *initial* if $c_0 = c_{in}$.

► **Remark 2.5.** Note that in the definition above we added the channel name to the message content inserted in a buffer. This is to exclude **executions** like $p!q(m) q?r(m)$ with $p \neq r$. Without this addition such **executions** would be allowed in the mailbox semantics, which is clearly not intended.

► **Definition 2.6 (Trace).** A trace of a CFM \mathcal{A} over a *process network* \mathcal{N} is a sequence $u \in Act^*$ such that there exists an initial **execution** of $\mathcal{T}_{\mathcal{N}}(\mathcal{A})$ labelled by u . The set of all *traces* of \mathcal{A} is denoted by $Tr_{\mathcal{N}}(\mathcal{A})$.

As we will also need to consider infixes of **executions**, we introduce action sequences which are coherent w.r.t. the fifo behavior that we expect from a *process network*:

► **Definition 2.7 (Viable sequence).** Let $\mathcal{N} = (\mathbb{B}, \text{bf})$ be a *process network*. A sequence of actions $v \in Act^*$ is called \mathcal{N} -viable if for every *buffer* $\mathbf{b} \in \mathbb{B}$:

- for every prefix u of v , the number of receives from \mathbf{b} in u is less or equal the number of sends to \mathbf{b} in u ;
- for every k , if the k -th receive from \mathbf{b} in v has label $q?p(m)$ then the k -th send to \mathbf{b} in v has label $p!q(m)$.

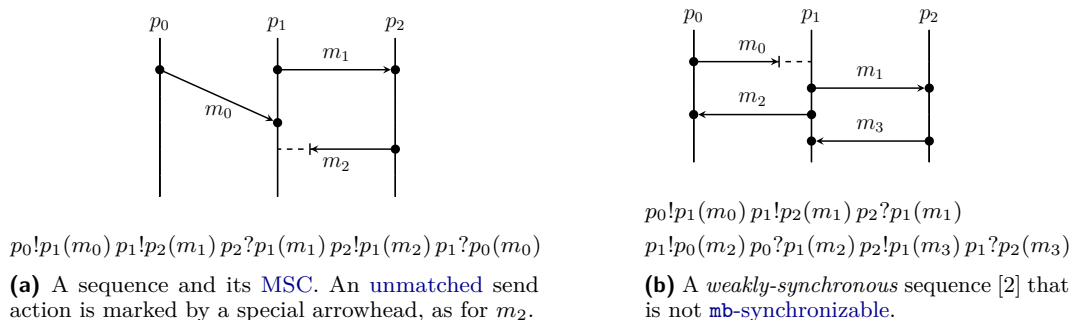
There is a strong connection between *traces* and *viable* sequences. For every sequence $u \in Act^*$, u is a *trace* of \mathcal{A} over \mathcal{N} iff u is \mathcal{N} -viable and u is recognized by $\prod_{p \in \mathbb{P}} \mathcal{A}_p$. Here, $\prod_{p \in \mathbb{P}} \mathcal{A}_p$ denotes the asynchronous product of the LTS \mathcal{A}_p , viewed as automata with every state final.

► **Remark 2.8.** It is easy to see that if a sequence is *mb-viable* then it is also *p2p-viable*. In fact, for every *process network* \mathcal{N} , we have that \mathcal{N} -viability implies *p2p-viability*. However, the converse is not true. For example, $p_0!p_1(m_0) p_2!p_1(m_1) p_1?p_2(m_1)$ is *p2p-viable*, but not *mb-viable* because m_1 is enqueued after m_0 in p_1 's mailbox, so it cannot be received first.

The classical *happens-before* relation [15], frequently used in reasoning about distributed systems, orders the actions of each process and every (*matched*) send action before its matching receive. The happens-before relation naturally associates a partial order with every *trace*, known as *message sequence chart*:

► **Definition 2.9 (Message Sequence Chart).** An MSC over \mathbb{P} is an *Act*-labeled partially ordered set $\mathcal{M} = (E, \leq_{hb}, \lambda)$ of events E , with $\lambda : E \rightarrow Act$ and $\leq_{hb} = (\leq_{\mathbb{P}} \cup \text{msg})^*$ the least partial order containing the relations $\leq_{\mathbb{P}}$ and *msg*, which are defined as:

1. For every process p , the set of events on p is totally ordered by $\leq_{\mathbb{P}}$, and $\leq_{\mathbb{P}}$ is the union of these total orders.
2. *msg* is the set of matching send/receive event pairs. In particular, $(e, f) \in \text{msg}$ implies $\lambda(e) = p!q(m)$ and $\lambda(f) = q?p(m)$ for some $p, q \in \mathbb{P}$ and $m \in \mathbb{M}$. Moreover, *msg* is a partial bijection between sends and receives such that every receive is paired with a (unique) send. A send is called *matched* if it is in the domain of *msg*, and *unmatched* otherwise.



■ **Figure 1** Two examples of MSCs.

The fifo behavior of message buffers implies that not every MSC arises as possible behavior. We formalise this for any process network $\mathcal{N} = (\mathbf{B}, \mathbf{bf})$ by defining a *buffer order*¹ $<_{\mathcal{N}}$ on sends to the same buffer. Let $e <_{\mathcal{N}} e'$ if e, e' are of type $p!q$ and $s!r$, resp., with $\mathbf{bf}(p, q) = \mathbf{bf}(s, r)$, and

- either e is **matched** and e' is **unmatched**,
- or $(e, f), (e', f') \in \text{msg}$ and $f <_{\mathbb{P}} f'$.

► **Definition 2.10** (Valid MSC). *Given a process network \mathcal{N} , an MSC $\mathcal{M} = (E, \leq_{\text{hb}}, \lambda)$ is called \mathcal{N} -valid if the relation $(<_{\text{hb}} \cup <_{\mathcal{N}})$ is acyclic.*

It is easy to see that an MSC is **p2p-valid** iff **matched** messages on any channel (p, q) never overtake and **unmatched** sends by p to q are $\leq_{\mathbb{P}}$ -ordered after the **matched** sends. An MSC is **mb-valid** iff for any sends $s <_{\text{hb}} s'$ to the same process, either they are both **matched** and their receives satisfy $r <_{\mathbb{P}} r'$, or s' is **unmatched**. Figure 1a shows an **mb-valid** MSC. An **mb-valid** MSC is the same as an MSC obtained from a trace that satisfies *causal delivery* in [3], and it is called *mailbox MSC* in [2].

If $u = u[1] \cdots u[n]$ is a **p2p-viable** sequence of actions then we can associate an MSC with u by setting $\text{msc}(u) = (E, \leq_{\text{hb}}, \lambda)$ with $E = \{e_1, \dots, e_n\}$, $\lambda(e_i) = u[i]$, and the orders defined as expected:

- $e_i \leq_{\mathbb{P}} e_j$ if $u[i]$ and $u[j]$ are performed by the same process and $i \leq j$.
- $(e_i, e_j) \in \text{msg}$ if there exists $k \geq 1$ and a buffer $\mathbf{b} \in \text{Ch}$ such that $u[i]$ is the k -th send to \mathbf{b} and $u[j]$ is the k -th receive from \mathbf{b} .

Note that $\text{msc}(u)$ only depends (up to isomorphism) on the projection of u on each process.

Caveat. Throughout the paper we switch between reasoning on \mathcal{N} -viable sequences (when we use automata) and their associated MSC (when we use partial orders). So when we refer to a position in a (**viable**) sequence u we often see it directly as an event of $\text{msc}(u)$, without further mentioning it.

► **Remark 2.11.** By definition, for any \mathcal{N} -viable sequence u the associated MSC $\text{msc}(u)$ is \mathcal{N} -valid. For the converse, if the process network is **many-to-one** and the MSC \mathcal{M} is \mathcal{N} -valid then every (labelled) linearization of the partial order $(<_{\text{hb}} \cup <_{\mathcal{N}})^*$ of \mathcal{M} is \mathcal{N} -viable. Indeed, all receives from the same buffer are totally ordered by $\leq_{\mathbb{P}}$ when the

¹ This definition of $<_{\mathcal{N}}$ is tailored for **many-to-one process networks**, but for simplicity we have chosen not to mention the restriction in the definition. Note that $<_{\mathcal{N}}$ is a strict partial order.

process network is **many-to-one**, and the corresponding sends are ordered in the same way because of the **buffer order**. For example, the sequence shown in Figure 1a is **mb-viable**, but $p_1!p_2(m_1) p_2?p_1(m_1) p_2!p_1(m_2) p_0!p_1(m_0) p_1?p_0(m_0)$ is not.

For a process network \mathcal{N} and a CFM \mathcal{A} we write $\text{msc}_{\mathcal{N}}(\mathcal{A}) = \{\text{msc}(u) \mid u \in \text{Tr}_{\mathcal{N}}(\mathcal{A})\}$ for the set of MSCs associated with initial executions of \mathcal{A} . By Remark 2.11, the set $\text{msc}_{\mathcal{N}}(\mathcal{A})$ consists only of \mathcal{N} -valid MSCs. The next definition introduces an equivalence relation \equiv on CFM traces that is ubiquitous in this paper. Two traces are equivalent up to commuting adjacent actions that are neither performed by the same process, nor a matching send/receive pair:

► **Definition 2.12** (Equivalence \equiv). *Two **p2p-viable** sequences $u, v \in \text{Act}^*$ are called equivalent if $\text{msc}(u) = \text{msc}(v)$ (up to isomorphism), and we write $u \equiv v$ in this case.*

► Remark 2.13. Two **p2p-viable** sequences are **equivalent** iff they have the same projection on each process.

► Remark 2.14. If $u, v \in \text{Act}^*$ are both \mathcal{N} -viable with $u \equiv v$, then $u \in \text{Tr}_{\mathcal{N}}(\mathcal{A})$ iff $v \in \text{Tr}_{\mathcal{N}}(\mathcal{A})$. However, \equiv does not preserve \mathcal{N} -viability, e.g. $p!q(m) r!q(m) q?p(m) \equiv r!q(m) p!q(m) q?p(m)$, but the left-hand side is **mb-viable** while the right-hand side is not.

For the rest of the section $\mathcal{N} = (\mathbf{B}, \text{bf})$ always refers to a **process network**. In order to be able to cope with partial executions we start by observing that **unmatched** sends to a **buffer** restrict the product of \mathcal{N} -viable sequences. Let u and v be two \mathcal{N} -viable sequences. The product $u *_{\mathcal{N}} v$ is defined if for every **buffer** $b \in \mathbf{B}$, if there is an **unmatched** send to b in u , then there is no receive from b in v . When it is defined, $u *_{\mathcal{N}} v$ is equal to uv . Note that the partial binary operation $*_{\mathcal{N}}$ is associative. Moreover, if $u_0 *_{\mathcal{N}} \dots u_i *_{\mathcal{N}} \dots u_j *_{\mathcal{N}} u_{j+1} \dots u_n$ is defined then $u_0 *_{\mathcal{N}} \dots u_i *_{\mathcal{N}} u_{j+1} \dots u_n$ is also defined, for every $i < j$. Note also that, when it is defined, the $*_{\mathcal{N}}$ -product of two \mathcal{N} -viable sequences is \mathcal{N} -viable.

► **Definition 2.15** (Exchanges, synchronizability).

1. An \mathcal{N} -exchange is any \mathcal{N} -viable sequence $w \in S^*R^*$.
2. An \mathcal{N} -viable sequence u is called \mathcal{N} -synchronous if it is a $*_{\mathcal{N}}$ -product of \mathcal{N} -exchanges. It is called \mathcal{N} -synchronizable if $u \equiv v$ for some \mathcal{N} -synchronous sequence v .
3. A CFM \mathcal{A} is \mathcal{N} -synchronizable if all its traces $u \in \text{Tr}_{\mathcal{N}}(\mathcal{A})$ are \mathcal{N} -synchronizable.

► Remark 2.16. The above definition of \mathcal{N} -synchronizability for $\mathcal{N} = \text{mb}$ differs from the one initially used by [3, 7] and later called *weak-synchronizability* in [2] (**mb-synchronizable** here coincides with *strongly synchronizable* in [2]). An **mb-viable** sequence of actions u is *weakly-synchronizable* if it is **equivalent** to a $*_{\text{p2p}}$ -product v of **mb-exchanges**. However, v is not required to be **mb-viable**. *Weak-synchronizability* yields more synchronizable traces, however some of them are spurious. In particular one cannot use the decompositions into exchanges from [3, 2] to check regular properties of executions, as we do in Section 4 later. Figure 1b shows an example distinguishing the definitions. The sequence there corresponds to a decomposition in exchanges according to [3, 2], but it is not **mb-viable**.

We end this section by a comparison between synchronizability for **peer-to-peer semantics** and **mailbox semantics**. These two notions are incomparable, in general. First, **mb-synchronizability** does not imply **p2p-synchronizability** simply because a system under **mb-semantics** has less executions than under **p2p-semantics**. Conversely, the following execution is **mb-viable** and **p2p-synchronizable**, but not **mb-synchronizable** (as we will see later the **unmatched** send makes it non-decomposable): $p!r(a) q!p(b) p?q(b) p!q(c) q?p(c) r!q(d) r?p(a)$. Finally, we note that **p2p-synchronizability** was shown to be undecidable in [2].

3 Reachability for mb-synchronizable systems

We start this section by showing that state reachability for **mb-synchronizable CFMs** is PSPACE-complete. The decidability (in exponential time) for **mb-synchronizable CFMs** can be already be inferred from [2] using the partial order logic LCPDL. The main point of this section is to introduce an automata-based approach to deal with **mb-synchronizable CFMs**. Although the set of **mb-synchronous traces** of a CFM is not regular in general, the projection of this set on (**marked**) send actions turns out to be regular. This crucial property is used later as a basic ingredient by our algorithm for deciding **mb-synchronizability**.

We start with an important observation saying that **mb-synchronizability** allows to focus on send actions. However, **unmatched** and **matched** sends need to be distinguished. So we introduce an extended alphabet $\bar{S} = \{\bar{s} \mid s \in S\}$. Sequences over $S \cup \bar{S}$ will be referred to as **ms-sequences**. For any **mb-viable** sequence u , we annotate every **unmatched** send $p!q(m)$ in u by $\overline{p!q(m)}$ and we denote by **marked**(u) the sequence obtained in this way. For example, for $u = p!q(m)p!r(m')r?p(m')$ we have **marked**(u) = $\overline{p!q(m)}p!r(m')r?p(m')$. The **ms-sequence** $ms(u)$ associated with an **mb-viable** sequence u is the projection of **marked**(u) on $S \cup \bar{S}$.

► **Lemma 3.1.**

1. For any **mb-exchanges** u, v with $ms(u) = ms(v)$, we have $u \equiv v$.
2. For any **mb-exchange** $u = vv'$ with $v \in S^*, v' \in R^*$, we define $\hat{u} = vv''$ with v'' obtained from v' by ordering the receives as their matching sends in v . Then \hat{u} is **mb-viable** and $u \equiv \hat{u}$.

Proof. For item 1, as $ms(u) = ms(v)$ and u, v are both **mb-viable**, we get that for each process p , the sequence of receives by p in u and v , resp., are the same. We derive from $u, v \in S^*R^*$ that u and v have the same projection on each process, and thus $u \equiv v$. For item 2 it is easy to check that \hat{u} is **mb-viable**, hence $u \equiv \hat{u}$ by item 1. ◀

► **Remark 3.2.** It is worth noting that Lemma 3.1 does not hold anymore under **p2p-semantics**. For example, the two **p2p-exchanges** $u = p_1!p_2(a)p_3!p_2(b)p_2?p_3(b)p_2?p_1(a)$ and $\hat{u} = p_1!p_2(a)p_3!p_2(b)p_2?p_1(a)p_2?p_3(b)$ have the same **marked** sequence, but they are not equivalent. This is the main reason why our decidability results don't carry over to the **p2p-semantics**.

Executable mb-exchanges

We now show how to check if an **ms-sequence** corresponds to an executable **mb-exchange** of a CFM \mathcal{A} . Since we use the same construction also for the model-checking problem in Section 4 we give a more general formulation below.

Given an **mb-viable** sequence u and two sets $D, D' \subseteq \mathbb{P}$, we write $D \overset{u}{\rightsquigarrow} D'$ if no process from D receives any message in u , and D' contains D and those processes q such that u has some **unmatched** send to q . We refer to processes in D, D' as *deaf* processes. It is routinely checked that, for every **mb-viable** sequences u_1, \dots, u_n , the product $u_1 *_{mb} \dots *_{mb} u_n$ is defined iff $D_0 \overset{u_1}{\rightsquigarrow} D_1 \dots \overset{u_n}{\rightsquigarrow} D_n$ for some sets D_0, \dots, D_n .

► **Definition 3.3** (*R-diamond*). Let $\mathcal{A} = (L, S \cup \bar{S} \cup R, \rightarrow_{\mathcal{A}})$ be an LTS. We say that \mathcal{A} is *R-diamond* if for all states $\ell, \ell' \in L$ and all receives $a, a' \in R$ performed by different processes, we have $\ell \xrightarrow{aa'}_{\mathcal{A}} \ell'$ iff $\ell \xrightarrow{a'a}_{\mathcal{A}} \ell'$.

For any states ℓ, ℓ' of \mathcal{A} , sets $D, D' \subseteq \mathbb{P}$ and **mb-viable** sequence u , we write $(\ell, D) \overset{u}{\rightsquigarrow}_{\mathcal{A}} (\ell', D')$ if $\ell \xrightarrow{\text{marked}(u)}_{\mathcal{A}} \ell'$ and $D \overset{u}{\rightsquigarrow} D'$. The next lemma shows how to adapt an *R-diamond* LTS to work on **ms-sequences** instead of **mb-synchronous** sequences (a similar idea appears in [12]):

► **Lemma 3.4.** *Assume that $\mathcal{A} = (L, S \cup \bar{S} \cup R, \rightarrow_{\mathcal{A}})$ is an *R-diamond LTS*. Then we can construct an LTS with ε -transitions $\mathcal{A}_{sync} = ((L \cup L^3) \times 2^{\mathbb{P}}, S \cup \bar{S}, \rightarrow_{sync})$ such that for any $v \in (S \cup \bar{S})^*$, states $\ell, \ell' \in L$, and sets $D, D' \subseteq \mathbb{P}$:*

$$(\ell, D) \xrightarrow{v}_{sync} (\ell', D') \quad \text{iff} \quad \exists u \text{ \textit{mb-synchronous} s.t. } v = ms(u) \text{ and } (\ell, D) \xrightarrow{u}_{\mathcal{A}} (\ell', D')$$

Proof. The LTS \mathcal{A}_{sync} has the following transitions, for any $\ell, \ell' \in L$, $D, D' \subseteq \mathbb{P}$, $a \in S \cup \bar{S}$:

$$\left\{ \begin{array}{ll} (\ell, D) \xrightarrow{\varepsilon}_{sync} (\ell, \hat{\ell}, D) & \text{for any } \hat{\ell} \in L \\ (\ell, \ell', \hat{\ell}, D) \xrightarrow{a}_{sync} (\ell_1, \ell'_1, \hat{\ell}, D) & \text{if } a = p!q(m), q \notin D, \ell \xrightarrow{a}_{\mathcal{A}} \ell_1, \ell' \xrightarrow{q?p(m)}_{\mathcal{A}} \ell'_1 \\ (\ell, \ell', \hat{\ell}, D) \xrightarrow{a}_{sync} (\ell_1, \ell', \hat{\ell}, D') & \text{if } a = \overline{p!q(m)}, \ell \xrightarrow{a}_{\mathcal{A}} \ell_1, D' = D \cup \{q\} \\ (\ell, \ell', \hat{\ell}, D) \xrightarrow{\varepsilon}_{sync} (\ell', D) & \text{if } \ell = \hat{\ell} \end{array} \right.$$

In other words, from a state $(\ell, D) \in L \times 2^{\mathbb{P}}$ the LTS \mathcal{A}_{sync} first guesses a “middle” state $\hat{\ell} \in L$ for the current *exchange*, as the state reached after the sends. Then it switches to state $(\ell, \hat{\ell}, \hat{\ell}, D)$. The first component and the second component track sends and their matching receives (if *matched*) in a “synchronous” fashion. The LTS \mathcal{A}_{sync} also guesses the end of the current *mb-exchange*, checking that the first component has reached the middle state $\hat{\ell}$ guessed originally. The claimed property of \mathcal{A}_{sync} follows from Lemma 3.1 (2) and from \mathcal{A} being *R-diamond*. ◀

Fix now a CFM \mathcal{A} . We abusively use the same notation $\xrightarrow{u}_{\mathcal{A}}$ as above for LTS: for any *global states* $g, g' \in G$ of \mathcal{A} , sets $D, D' \subseteq \mathbb{P}$ and *mb-viable* sequence u , we write $(g, D) \xrightarrow{u}_{\mathcal{A}} (g', D')$ if u labels an *execution* in $\mathcal{T}_{mb}(\mathcal{A})$ from the configuration $(g, \varepsilon^{\mathbb{B}})$ to some configuration $(g', (w_b)_{b \in \mathbb{B}})$, and $D \xrightarrow{u} D'$. We obtain from the previous lemma that:

► **Lemma 3.5.** *Let \mathcal{A} be a CFM, $g, g' \in G$ two *global states* of \mathcal{A} , and $D, D' \subseteq \mathbb{P}$ two sets of processes. One can construct automata \mathcal{B}, \mathcal{C} with $O(|G|^3 \times 2^{|\mathbb{P}|})$ states such that*

$$\begin{aligned} L(\mathcal{B}) &= \left\{ v \in (S \cup \bar{S})^* \mid \exists u \text{ \textit{mb-exchange} s.t. } v = ms(u) \text{ and } (g, D) \xrightarrow{u}_{\mathcal{A}} (g', D') \right\}, \\ L(\mathcal{C}) &= \left\{ v \in (S \cup \bar{S})^* \mid \exists u \text{ \textit{mb-synchronous} s.t. } v = ms(u) \text{ and } (g, D) \xrightarrow{u}_{\mathcal{A}} (g', D') \right\}. \end{aligned}$$

Proof. Assume that $\mathcal{A} = (\mathcal{A}_p)_{p \in \mathbb{P}}$. Let \mathcal{Q} denote the asynchronous product $\prod_{p \in \mathbb{P}} \bar{\mathcal{A}}_p$, where each $\bar{\mathcal{A}}_p$ is the LTS obtained from \mathcal{A}_p by adding a transition $\ell_p \xrightarrow{\bar{s}}_p \ell'_p$ for each transition $\ell_p \xrightarrow{s}_p \ell'_p$ with $s \in S$. Note that \mathcal{Q} is *R-diamond*. Moreover, it is routinely checked that, for every *mb-viable* sequence u , the relation $\xrightarrow{u}_{\mathcal{A}}$ coincides with the relation $\xrightarrow{u}_{\mathcal{Q}}$.

For \mathcal{C} we take the automaton \mathcal{Q}_{sync} constructed according to Lemma 3.4, and set the initial state to (g, D) and the final state to (g', D') . For \mathcal{B} , we need to tinker a bit with \mathcal{Q}_{sync} to ensure that we read only one *exchange*. So we remove all transitions from/to states in $L \times 2^{\mathbb{P}}$ except the transitions from (g, D) , which we set as initial, and the transitions to (g', D') , which we set as final. If $(g, D) = (g', D')$ then we make two different states for the initial and the final one. ◀

Using Lemma 3.5 we establish the upper bound of the global-state reachability problem for *mb-synchronizable CFMs* (the lower bound is straightforward). By global-state reachability we mean the existence of a reachable configuration with a specified global state. Decidability was shown in [7] for weak-synchronizability (correcting the proof in [3]) and assuming a uniform bound on the size of *exchanges*.

► **Theorem 3.6.** *The global-state reachability problem for **mb-synchronizable** CFMs is PSPACE-complete.*

Proof. Note first that if \mathcal{A} is a CFM and u, v two **mb-viable** sequences u, v with $u \equiv v$ then $c_{in} \xrightarrow{u}_{\mathcal{A}} c$ implies that $c_{in} \xrightarrow{v}_{\mathcal{A}} c'$ for some c' with the same global state as c . Since we assume that the CFM is **mb-synchronizable** we can choose v to be **mb-synchronous**. Thus we can use automaton \mathcal{C} from Lemma 3.5 to show the upper bound. This automaton can clearly be constructed on-the-fly in polynomial space.

For the lower bound we reduce from the problem of intersection of NFA. Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be NFA over the alphabet Σ . We use processes p_1, \dots, p_n where each p_i simulates \mathcal{A}_i . Process p_1 starts by guessing a letter a of Σ , making a transition on a and sending a to p_2 . Afterwards each process p_i receives a letter a from p_{i-1} , makes a transition on a , then sends a to p_{i+1} . Back again at p_1 , the procedure restarts. Figure 2 shows the principle.

Upon reaching a final state, p_1 can send message `accept` to p_2 and then stop. If p_i receives `accept` from p_{i-1} while being in a final state, it relays `accept` to p_{i+1} , and then stops.

One can see that every **trace** of the CFM is **mb-synchronizable**, as every message is in its own **exchange**. Moreover, the global-state $(\text{accept})_{p \in \mathbb{P}}$ is reachable if and only if the intersection of $\mathcal{A}_1, \dots, \mathcal{A}_n$ is non-empty. ◀

4 Model-checking regular properties

In this section we introduce a class of properties against which we can verify **mb-synchronizable** CFMs. We look for regular properties P over the alphabet $S \cup R \cup \bar{S}$, so we exploit the **marked sends** to refer (indirectly) to messages. The model-checking problem we consider is the following:

CFM-VS-REGULAR PROPERTY

INPUT: **mb-synchronizable** CFM \mathcal{A} , regular property $P \subseteq (S \cup \bar{S} \cup R)^*$.

OUTPUT: Yes if for every **mb-synchronous** trace $u \in Tr_{\text{mb}}(\mathcal{A})$ we have $\text{marked}(u) \in P$.

The properties we consider are regular, **R-closed** subsets of $(S \cup \bar{S} \cup R)^*$:

► **Definition 4.1** (*R-closed properties*). *Let \equiv_R be the reflexive-transitive closure of the relation consisting of all pairs $(uabv, ubav)$ with $u, v \in (S \cup \bar{S} \cup R)^*$, $a, b \in R$, and a, b performed by distinct processes. A property $P \subseteq (S \cup \bar{S} \cup R)^*$ is called **R-closed** if it is closed under \equiv_R (i.e., for any $u \equiv_R v$ we have $u \in P$ iff $v \in P$).*

As an example, we can consider a system with a central process c and a set of orbiting processes p_1, \dots, p_n . The central process gives tasks to the orbiting processes, and they send back their results. We can state a property expressing a round-based behavior for c : it sends tasks to orbiting processes, and if a process p_i does not send back to c in the next round, it will not participate in further rounds anymore. The opposite property consists of all sequences from $A^*S_c^*c!p_i(m)S_c^*R^+(\bigcup_{j \neq i} S_{p_j})^+R^+S_c^+R^*A^*p_i!c(m')A^*$ for some i and m, m' , and $A = S \cup \bar{S} \cup R$. As the above property is **R-closed**, its complement is too.

We will show that if the regular property is **R-closed** then the model-checking problem stated above is PSPACE-complete. Before that recall that both being **mb-viable** and being **mb-synchronous** (assuming **mb-viable**) are non regular properties. However, it is not necessary to be able to express the above, as we will apply the property to **mb-synchronous** traces of CFM. The next lemma is similar to Lemma 3.4:

► **Lemma 4.2.** *Let $P \subseteq (S \cup \bar{S} \cup R)^*$ be regular and R -closed. Then the set*

$$\text{Sync}(P) = \{v \in (S \cup \bar{S})^* \mid \exists u \text{ mb-synchronous s.t. } v = \text{ms}(u) \text{ and } \text{marked}(u) \in P\}$$

is regular. If P is given by an R -diamond NFA with n states, then we can construct an NFA for $\text{Sync}(P)$ with $O(n^3 \cdot 2^{|\mathbb{P}|})$ states.

Proof. Let P be given by an R -diamond NFA $\mathcal{P} = (L, S \cup \bar{S} \cup R, \rightarrow_{\mathcal{P}}, \ell_0, F)$ with n states. We may assume w.l.o.g. that \mathcal{P} contains no ε -transition. Consider the LTS with ε -transitions $\mathcal{P}_{\text{sync}}$ obtained from Lemma 3.4. Recall that this LTS has $O(n^3 \times 2^{|\mathbb{P}|})$ states. As NFA for $\text{Sync}(P)$, we take $\mathcal{P}_{\text{sync}}$, with (ℓ_0, \emptyset) as initial state, and $F \times 2^{\mathbb{P}}$ as final states. ◀

► **Theorem 4.3.** *The CFM-VS-REGULAR PROPERTY problem is PSPACE-complete if the property is R -closed. There exist properties that are not R -closed for which the problem is undecidable.*

Proof. For the upper bound, consider an **mb-synchronizable CFM** $\mathcal{A} = (\mathcal{A}_p)_{p \in \mathbb{P}}$ and an R -closed regular property $P \subseteq (S \cup \bar{S} \cup R)^*$ given by an NFA \mathcal{P} . Since P is R -closed, its complement P^{co} is also R -closed. As in the proof of Lemma 3.5, let \mathcal{Q} denote the asynchronous product $\prod_{p \in \mathbb{P}} \bar{\mathcal{A}}_p$, where each $\bar{\mathcal{A}}_p$ is the LTS obtained from \mathcal{A}_p by adding a transition $\ell_p \xrightarrow{\bar{s}} \ell'_p$ for each transition $\ell_p \xrightarrow{s} \ell'_p$ with $s \in S$. Note that \mathcal{Q} is R -diamond, so its language $Q = L(\mathcal{Q})$ is R -closed. We derive that $Q \cap P^{co}$ is R -closed. It is routinely checked that $(\mathcal{A}, \mathcal{P})$ is a positive instance of CFM-VS-REGULAR PROPERTY iff the set $\text{Sync}(Q \cap P^{co})$, as defined in Lemma 4.2, is empty. To derive the PSPACE upper bound from this lemma, we still need to provide an R -diamond NFA for $Q \cap P^{co}$. This R -diamond NFA is simply the synchronous product of \mathcal{Q} and the minimal automaton of P^{co} . The latter is R -diamond since P^{co} is R -closed, and it can be constructed on-the-fly in polynomial space from \mathcal{P} . Now it suffices to check emptiness of the NFA for $\text{Sync}(Q \cap P^{co})$ from Lemma 4.2. The lower bound is again straightforward.

For the undecidability of model-checking a property that is not R -closed we use a straightforward reduction from PCP. Let $(u_i, v_i)_{i=1 \dots k}$ be an instance of PCP over the binary alphabet $\{0, 1\}$. We can have three processes p, U, V and process p who sends, in rounds, some pair (u_i, v_i) to U and V , resp. That is, p sends u_i (v_i , resp.) letter by letter to U (V , resp.). The processes U and V do nothing except receiving whatever p sends to them.

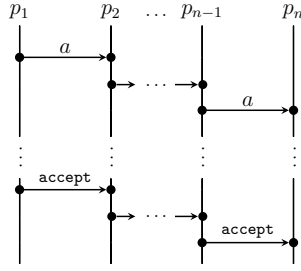
There is a solution to the given PCP instance iff there is a **trace** consisting of a single fully **matched mb-exchange** where U and V perform the same receives in lock-step. So we take as property P the regular language $P = (S \cup \bar{S} \cup R)^* \setminus P^{co}$ where $P^{co} = S^*\{U?p(0)V?p(0), U?p(1)V?p(1)\}^*$. ◀

Comparing p2p and mb semantics

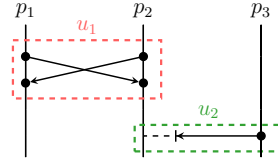
Given a protocol that was designed for p2p communication, it can be useful to know whether the protocol can be also deployed under mailbox communication. We call this property mailbox-similarity:

► **Definition 4.4** (Mailbox-similarity). *A p2p-viable sequence of actions u is called mailbox-similar if there exists some mb-viable sequence v such that $u \equiv v$. A CFM \mathcal{A} is called mailbox-similar if every trace from $\text{Tr}_{\text{p2p}}(\mathcal{A})$ is mailbox-similar.*

Equivalently, a CFM \mathcal{A} is mailbox-similar if every MSC from $\text{msc}_{\text{p2p}}(\mathcal{A})$ is mb-valid. Unsurprisingly, as it is often the case under p2p semantics, mailbox-similarity is undecidable without further restrictions:



■ **Figure 2** The MSC of a trace of the CFM for automata intersection.



■ **Figure 3** MSC of $u = p_2!p_1(m_1) p_1!p_2(m_2) p_1?p_2(m_1) p_2?p_1(m_2) p_3!p_2(m_3)$, with the two SCCs of its communication graph. Note that $1 \preceq_{\text{mb}}^u 2$, but neither $1 \preceq_{p_2p}^u 2$ nor $2 \preceq_{p_2p}^u 1$ holds.

► **Lemma 4.5.** *The question whether a given CFM is mailbox-similar is undecidable.*

In the remainder of this section, we show that mailbox-similarity becomes decidable if we assume that the CFM is **mb-synchronizable**. Recall that the latter means that every trace from $Tr_{\text{mb}}(\mathcal{A})$ is **mb-synchronizable**.

The next lemma shows how to check that two positions in an **mb-synchronous** sequence u are causally-ordered, i.e., there is some $(\prec_{\text{hb}} \cup \prec_{\text{mb}})$ -path between these positions (as usual, this refers to a path between associated events in $\text{msc}(u)$). We mark these positions by using a “tagged” alphabet $\Sigma = (S \cup \bar{S} \cup R) \times \{\circ, \bullet\}$.

► **Lemma 4.6.** *We can construct an R -diamond automaton \mathcal{D} with $O(|\mathbb{P}|)$ states over the alphabet Σ such that for every **mb-synchronous** sequence $u \in Act^*$ and every positions $i < j$ of u such that $u[i]$ and $u[j]$ are in S , there is a $(\prec_{\text{hb}} \cup \prec_{\text{mb}})$ -path from $u[i]$ to $u[j]$ iff \mathcal{D} accepts the word $\text{marked}(u)$ tagged by \bullet at i and j and by \circ elsewhere.*

Proof. Recall that $\preceq_{\text{hb}} = (\prec_{\mathbb{P}} \cup \text{msg})^*$ is the happens-before order. The automaton \mathcal{D} will guess a $(\prec_{\mathbb{P}} \cup \text{msg} \cup \prec_{\text{mb}})$ -path from $u[i]$ to $u[j]$. It will actually use only send actions of $\text{marked}(u)$, relying on the fact that u is **mb-synchronous**. That is, \mathcal{D} guesses a subsequence of positions $i_1 < \dots < i_t$ of u , with each $u[i_k] \in S$, as described in the following. Let $i_0 = i$ and $i_{t+1} = j$. We have three cases, and \mathcal{D} guesses in which case we are:

- $u[i_k], u[i_{k+1}]$ are performed by the same process p . After i_k the automaton \mathcal{D} remembers the pair $(\prec_{\mathbb{P}}, p)$ until it guesses i_{k+1} .
- $u[i_k], u[i_{k+1}]$ are both sends to the same process p , and $u[i_k]$ is **matched**. After i_k the automaton \mathcal{D} remembers (\prec_{mb}, p) until it guesses i_{k+1} .
- $u[i_k]$ is **matched**, its receive $u[h]$ is performed by the same process p as $u[i_{k+1}]$, and $h < i_{k+1}$. After i_k the automaton \mathcal{D} remembers the pair (msg, S, p) . After the next receive action, \mathcal{D} changes its state to (msg, R, p) until it guesses i_{k+1} . The assumption that u is **mb-synchronous** guarantees that the receive $u[h]$ **matched** with $u[i_k]$ has already occurred when \mathcal{D} guesses i_{k+1} .

By construction, if \mathcal{D} accepts $\text{marked}(u)$, then we have a $(\prec_{\text{hb}} \cup \prec_{\text{mb}})$ -path from $u[i]$ to $u[j]$, with $i < j$ the two positions tagged by \bullet in $\text{marked}(u)$.

For the left-to-right implication, assume that $u[i]$ and $u[j]$ are in S and that we have a $(\prec_{\text{hb}} \cup \prec_{\text{mb}})$ -path from $u[i]$ to $u[j]$. This path is a sequence $i = i_0 < i_1 < \dots < i_t < i_{t+1} = j$ of positions of u , such that each pair of consecutive indices is related by $\prec_{\mathbb{P}}$, \prec_{mb} or msg . Moreover, we may assume w.l.o.g. that there are no two consecutive $\prec_{\mathbb{P}}$ -arcs on this path. If the path contains only $\prec_{\mathbb{P}}$ and \prec_{mb} -arcs, then \mathcal{D} applies one of the first two rules above. Consider now a msg -arc $(u[i_k], u[i_{k+1}])$. As $u[i_{k+1}]$ is a receive, we get that $u[i_{k+1}] \prec_{\mathbb{P}} u[i_{k+2}]$.

Moreover, $u[i_{k+2}]$ is a send since there are no two consecutive $\langle_{\mathbb{P}}$ -arcs on the path. So \mathcal{D} can apply the third rule to go from i_k to i_{k+2} . We get that \mathcal{D} accepts the word $\text{marked}(u)$ tagged by \bullet at i and j and by \circ elsewhere. The number of states of \mathcal{D} is $4 * |\mathbb{P}| + 2$ (2 for initial/final state). \blacktriangleleft

► **Lemma 4.7.** *For any receive action $r \in R$, we can construct an R -diamond automaton \mathcal{P}_r with $O(|\mathbb{P}|)$ states over the alphabet $(S \cup \bar{S} \cup R)$ such that for every mb-synchronous sequence u , it holds that ur is p2p-viable and not mailbox-similar iff \mathcal{P}_r accepts $\text{marked}(u)$.*

Proof sketch. Consider a receive action $r = q?p(m)$. Let W_r denote the set of words $w \in \Sigma^*$ such that w contains exactly two positions $i < j$ tagged by \bullet , $w[i]$ is an unmatched send to q , $w[j]$ is p!q(m) , and no $w[h]$ with $h < j$ is an unmatched send from p to q . It is easily seen that W_r is recognized by an R -diamond NFA \mathcal{W}_r with three states. Let \mathcal{E}_r denote the synchronous product of \mathcal{W}_r and the R -diamond automaton \mathcal{D} from Lemma 4.6. The desired automaton \mathcal{P}_r is obtained from \mathcal{E}_r by untagging it, that is, by replacing each tagged action $(a, t) \in \Sigma$ by a . As \mathcal{E}_r is R -diamond, so is \mathcal{P}_r . By construction, \mathcal{P}_r satisfies the lemma condition. The details can be found in the full version of the paper [6]. \blacktriangleleft

We derive from the previous lemma that $\text{mailbox-similarity}$ can be solved in PSPACE for $\text{mb-synchronizable CFMs}$. The proof uses Lemma 4.2 and is similar to the proof of Theorem 4.3.

► **Theorem 4.8.** *The question whether a given $\text{mb-synchronizable CFM}$ is mailbox-similar is PSPACE-complete.*

5 Checking mb-synchronizability

In this section we show our main result, namely an algorithm to know if a CFM is mb-synchronizable . As a side result we obtain optimal complexity bounds for some problems considered in [7, 12].

The high-level schema of the algorithm is to look for a minimal witness for non- $\text{mb-synchronizability}$. This amounts to searching for an $\text{mb-synchronous trace}$ that violates $\text{mb-synchronizability}$ after adding one (receive) action. Of course, we need Theorem 3.6 to guarantee that the $\text{mb-synchronous trace}$ is executable. In addition, we have to detect the violation of $\text{mb-synchronizability}$, and for this we need to determine if an exchange is non-decomposable into smaller exchanges . Section 5.1 shows automata for non-decomposable exchanges , and in Section 5.2 we present the algorithm that finds minimal witnesses.

5.1 Automata for atomic exchanges

In this section we consider sequences of actions that cannot be split into smaller pieces without separating messages [11, 12]. We introduce these notions for arbitrary $\text{many-to-one process networks}$ \mathcal{N} . Later we will fix $\mathcal{N} = \text{mb}$ since reachability over synchronizable sequences is decidable in this setting.

► **Definition 5.1** (Atomic sequences). *An \mathcal{N} -viable sequence $u \in \text{Act}^*$ is \mathcal{N} -atomic (or atomic for short) if $u \equiv v *_{\mathcal{N}} w$ with v, w both \mathcal{N} -viable implies that one of v, w is empty.*

To check atomicity we can use a graph criterium introduced already in [13] (see also [11]), that is similar to the notion of conflict graph used in [3]:

► **Definition 5.2** (Communication graph). *Let u be an \mathcal{N} -viable sequence, and $\mathcal{M} = \text{msc}(u)$. The \mathcal{N} -communication graph of u is the directed graph $H_{\mathcal{N}}(u) = (V, E)$ where V is the set of all events of \mathcal{M} and the edges are defined by $(e, e') \in E$ if $e <_{\mathbb{P}} e'$ or $e <_{\mathcal{N}} e'$ or $\{(e, e'), (e', e)\} \cap \text{msg} \neq \emptyset$.*

The right part of Figure 4 shows (partly) the **communication graph** of the **MSC** in the left part. The cycle witnesses that the **MSC** is \mathcal{N} -atomic for $\mathcal{N} \in \{\text{mb}, \text{p2p}\}$, according to the next lemma.

► **Lemma 5.3.** *Let $u \in \text{Act}^*$ be a \mathcal{N} -viable sequence and $H_{\mathcal{N}}(u)$ the \mathcal{N} -communication graph of $\text{msc}(u)$. Then u is \mathcal{N} -atomic if and only if $H_{\mathcal{N}}(u)$ is strongly connected.*

From Lemma 5.3 we can infer a decomposition of any **trace** in **atomic** subsequences that is unique up to permuting adjacent **atomic** sequences that are not ordered in the sense of the next definition:

► **Definition 5.4** (Skeleton). *Let u be a \mathcal{N} -viable sequence with $\mathcal{M} = \text{msc}(u)$ and $H_{\mathcal{N}}(u)$ be the \mathcal{N} -communication graph of \mathcal{M} . Fix some arbitrary topological indexing $\{1, \dots, n\}$ of the SCCs of $H_{\mathcal{N}}(u)$. We define the skeleton of u as $\text{skel}(u) = (\{1, \dots, n\}, \preceq_{\mathcal{N}}^u)$, where $\preceq_{\mathcal{N}}^u$ is the partial order induced by setting $i \prec_{\mathcal{N}}^u j$ for $1 \leq i < j \leq n$ if there is some $<_{\mathbb{P}}$ -arc or some **mb**-arc in $H_{\mathcal{N}}(u)$ from the SCC with index i to the SCC with index j .*

► **Remark 5.5.** Assume that $u = u_1 *_{\mathcal{N}} \dots *_{\mathcal{N}} u_n$ where each u_i is \mathcal{N} -atomic and non-empty, and we index the SCCs according to the order of the u_i . Then we obtain $\text{skel}(u) = (\{1, \dots, n\}, \preceq_{\mathcal{N}}^u)$ with $i \prec_{\mathcal{N}}^u j$ if either both u_i and u_j contain some actions on the same process; or they both contain some send to the same **buffer**, with the one in u_i being **matched**. See Figure 3 for an example.

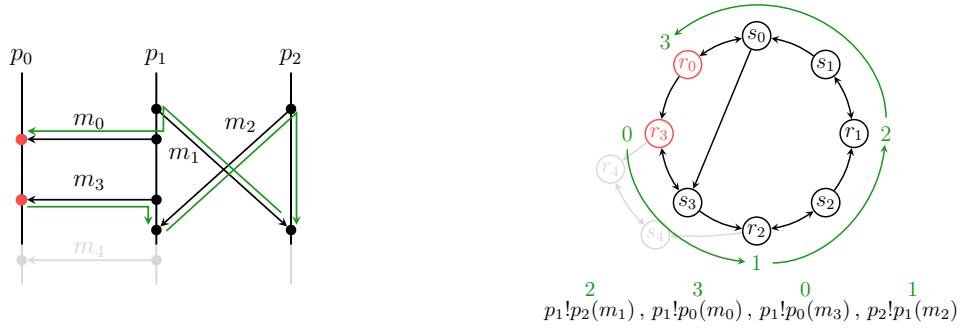
► **Lemma 5.6.** *Let u be an \mathcal{N} -viable sequence. Then there exist some \mathcal{N} -atomic non-empty sequences u_1, \dots, u_k such that $u \equiv u_1 *_{\mathcal{N}} \dots *_{\mathcal{N}} u_k$. Such a decomposition into \mathcal{N} -atomic non-empty sequences is unique up to the partial order $\preceq_{\mathcal{N}}^u$ of $\text{skel}(u)$.*

Throughout the remaining of this section we fix $\mathcal{N} = \text{mb}$. We will show now a simple, automaton-compatible condition to certify that an **ms-sequence** $v = \text{ms}(u)$ corresponds to an **mb-atomic exchange** u . First we note that, in order for the **communication graph** $H_{\text{mb}}(u)$ to be strongly connected, there must exist for every process p that is active in u some path from the last action of p to the first action of p (if there are at least two actions of p in u). A process p is called *active* in u if there is at least some action performed by p in u (resp., if $v = \text{ms}(u)$ contains either a send performed by p , or a **matched** send to p). We look for such a path for every active process and then we need to connect all such paths together.

Let $u \in \text{Act}^*$ be an **mb-exchange**. For some suitable integer n we define a labeling of $v = \text{ms}(u)$ as an injective mapping $\pi : \{0, \dots, n\} \rightarrow \{1, \dots, |v|\}$ where $\pi(i) = j$ means that position j of v is labeled by i . We say that π is a *well-labeling* of v (of size n) if, for every $0 \leq i < n$:

- either $\pi(i) < \pi(i+1)$ and, for some process p :
 - $v[\pi(i)]$ and $v[\pi(i+1)]$ are both sends by p , or
 - $v[\pi(i)]$ and $v[\pi(i+1)]$ are both sends to p , with $v[\pi(i)]$ **matched** (direct arc)
- or $v[\pi(i)]$ is a send by p and $v[\pi(i+1)]$ is a **matched** send to p (indirect arc).

An example of such labeling is shown in Figure 4. Informally, one can see the two types of arcs between positions of v as:



■ **Figure 4** A well-labeling of the *ms-sequence* bottom right, witnessing a path in the communication graph of the *MSC* left, from the last to the first event of process p_0 . The s_i and r_i vertices of the communication graph correspond respectively to the send and receive of message m_i .

- A **direct arc** between two sends corresponds to the **process order** $\leq_{\mathbb{P}}$ or the **mailbox order** \leq_{mb} in $\text{msc}(u)$. For example, we have a **direct arc** from position 2 to 3 in Figure 4.
- An **indirect arc** between two sends stems from composing edges of the **communication graph** $H_{\text{mb}}(u)$ that involve a receive event. An **indirect arc** is specific to **mb-exchanges**: in $H_{\text{mb}}(u)$ we can go from the event of $v[i]$ to the receive associated with the event of $v[j]$ (since u is an **mb-exchange** this receive is after $v[i]$), and then follow the message edge backwards to the event of $v[j]$. For example, we have an **indirect arc** from position 1 to 2 in Figure 4.

► **Lemma 5.7.** *Let u be an **mb-exchange** with $\mathcal{M} = \text{msc}(u)$, and $v = \text{ms}(u)$. There is a path in the **communication graph** $H_{\text{mb}}(u)$ from the event of \mathcal{M} corresponding to $v[i]$ to the event corresponding to $v[j]$ if and only if there is a **well-labeling** of v starting at i and ending at j .*

Proof. For the right-to-left direction, let π be a **well-labeling** of v starting at i and ending at j . As π is a **well-labeling**, there is a path in $H_{\text{mb}}(u)$ from the event corresponding to $v[\pi(k)]$ to the one of $v[\pi(k+1)]$, for every k in the domain of π . Each such path is either a direct edge, or consists of two edges, as explained before the statement of the lemma in the main body.

For the left-to-right direction, we suppose there is a path Π in $H_{\text{mb}}(u)$ from the event of $v[i]$ to the event of $v[j]$. We construct a labeling π of v that starts at i and ends at j , by labelling the positions of v that correspond to the events of Π with their respective rank on Π . Suppose that n positions are labeled and let $0 \leq k < n$. We show the existence of an arc from $\pi(k)$ to $\pi(k+1)$, which is either direct or indirect. There are three cases:

- There is no receive between the event of $v[\pi(k)]$ and the one of $v[\pi(k+1)]$ on Π . Thus $v[\pi(k)]$, $v[\pi(k+1)]$ are consecutive on Π and are either ordered by $<_{\mathbb{P}}$ or by $<_{\text{mb}}$. This gives a **direct arc** from $\pi(k)$ to $\pi(k+1)$.
- Between the event of $v[\pi(k)]$ and the one of $v[\pi(k+1)]$ we see on Π the receive matching $v[\pi(k)]$ before the receive matching $v[\pi(k+1)]$. Note that both receives must be on the same process (as all receives between $v[\pi(k)]$ and $v[\pi(k+1)]$), so they are ordered by $<_{\mathbb{P}}$. Thus, the events of $v[\pi(k)]$ and $v[\pi(k+1)]$, respectively, are ordered by $<_{\text{mb}}$. This gives a **direct arc** from $\pi(k)$ to $\pi(k+1)$.
- Between the event of $v[\pi(k)]$ and the one of $v[\pi(k+1)]$ we have on Π the receive matching the event of $v[\pi(k+1)]$ on the same process as the event of $v[\pi(k)]$. This gives an **indirect arc** from $\pi(k)$ to $\pi(k+1)$. ◀

► **Remark 5.8.** In Lemma 5.7, we only talk about send actions. If we are interested in a path in $H_{\text{mb}}(u)$ to a receive action, we just need to exhibit the path to its corresponding send action.

We can infer a bound on the size of **well-labelings**, using the pigeonhole principle on the **direct arcs** and **indirect arcs** going through each process.

► **Lemma 5.9.** *Let $u \in \text{Act}^*$ be an **mb-exchange** and $v = \text{ms}(u)$. If there is a path in the **communication graph** $H_{\text{mb}}(u)$ between $v[i]$ and $v[j]$ then there is a **well-labeling** of $\text{ms}(u)$ starting at position i and ending at position j of size at most $|\mathbb{P}|^2 + |\mathbb{P}|$.*

We construct now two kinds of automata, both working on **ms-sequences** $v = \text{ms}(u)$. Automaton \mathcal{B}_p will check for a process p that is active in u , that all actions performed by p are on a cycle in $H_{\text{mb}}(u)$. Automaton \mathcal{B}_{all} will check that all actions of active processes in u appear together on a cycle in $H_{\text{mb}}(u)$, by looking for a cycle going through all active processes at least once. Finally we take the product of all automata \mathcal{B}_p such that p is active and the automaton \mathcal{B}_{all} . The resulting automaton has $|\mathbb{P}|^{O(|\mathbb{P}|^3)}$ states and verifies the following property: for every **mb-exchange** u , it holds that u is **atomic** iff $\text{ms}(u)$ is accepted by the automaton. By taking the product of this last automaton with the automaton verifying that the **ms-sequence** corresponds to an **mb-exchange** (see Lemma 3.5), we immediately get:

► **Lemma 5.10.** *Let \mathcal{A} be a **CFM**, g, g' two **global states** of \mathcal{A} and $D, D' \subseteq \mathbb{P}$. One can construct an automaton \mathcal{B} with $O(|G|^3 \cdot |\mathbb{P}|^{O(|\mathbb{P}|^3)})$ states, such that*

$$L(\mathcal{B}) = \left\{ v \in (S \cup \bar{S})^* \mid \exists u \text{ atomic mb-exchange s.t. } v = \text{ms}(u) \text{ and } (g, D) \stackrel{u}{\rightsquigarrow}_{\mathcal{A}} (g', D') \right\}$$

5.2 Verifying **mb-synchronizability**

To check **mb-synchronizability** we look for an **mb-viable trace** that is not equivalent to a $*_{\text{mb}}$ -product of **mb-exchanges**. Such a *witness* u must contain some **atomic** factor v that is not equivalent to an **mb-exchange**. In other words, $u \equiv u' *_{\text{mb}} v *_{\text{mb}} u''$ for some u', u'' , with $v' \notin S^*R^*$ for every $v \equiv v'$. It is enough to reason on **atomic** factors, since for any **exchange** u where $u \equiv u_1 *_{\text{mb}} \dots *_{\text{mb}} u_n$ with each u_i **atomic**, all factors u_i are also **exchanges**. Note that an **atomic** v is not equivalent to an **mb-exchange** iff some process in v does a send after a receive.

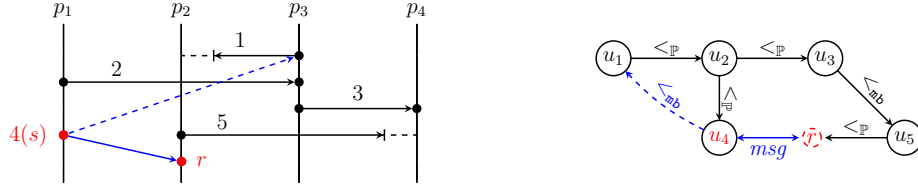
The next lemmas refer to the structure of *minimal witnesses* for non-**mb-synchronizability**.

► **Lemma 5.11.** *Let $u = vr$ be an **mb-viable** sequence with $r \in R$. There exist **mb-atomic** non-empty sequences v_1, \dots, v_n and indices $1 \leq i < j \leq n$ such that (1) $v \equiv v_1 *_{\text{mb}} \dots *_{\text{mb}} v_n$, and (2) $u \equiv v_1 *_{\text{mb}} \dots *_{\text{mb}} v_{i-1} *_{\text{mb}} w *_{\text{mb}} v_{j+1} *_{\text{mb}} \dots *_{\text{mb}} v_n$ with $w = (v_i *_{\text{mb}} \dots *_{\text{mb}} v_j)r$ being **mb-atomic**.*

► **Lemma 5.12.** *Let $u = vr$ be an **mb-viable** sequence with $r \in R$, such that v is not **mb-atomic**. We denote by s the send event **matched** with r in u , and by q the process of r . Then u is **mb-atomic** iff for every decomposition $v \equiv v_1 *_{\text{mb}} \dots *_{\text{mb}} v_n$ with v_i **mb-atomic** for all i : (1) v_1 contains s or some **unmatched** send to process q , and (2) v_n contains s or some action performed by process q .*

An example of such a decomposition is shown in Figure 5.

► **Lemma 5.13.** *Let $u = vr$ be **mb-viable** with $r \in R$ and v is **mb-synchronizable**. Let also s be the send matching r in u , and q the process doing r . Then u is not **mb-synchronizable** iff there exist $(v_i)_{i=1}^n$ with $v \equiv v_1 * \dots * v_n$, indices $1 \leq i_1 < \dots < i_k \leq n$, and $p \in \mathbb{P}$ s.t.:*



■ **Figure 5** The MSC of an atomic sequence. It is not **mb-synchronizable** by Lemma 5.13, each u_i consists of message i , the indices are $(1, 2, 3, 5)$, and $m = 2$.

1. Each v_i is **mb-atomic**.
2. For every $1 \leq j < k$ we have $i_j \prec_{\text{mb}}^v i_{j+1}$.
3. v_{i_1} contains s or some **unmatched** send to process q ; v_{i_k} contains s or some action performed by process q .
4. There exists $1 \leq m < k$ such that v_{i_m} contains a receive by p and $v_{i_{m+1}}$ a send by p .

Note that while we can guess **mb-synchronous** sequences without storing messages (Lemma 3.5), we need to be careful when guessing u in Lemma 5.13 so that it is **mb-viable**. E.g., by reversing message 2 in Figure 5 the sequence becomes non-**mb-viable**.

► **Lemma 5.14.** Let $u = vr$ be a **p2p-viable** sequence with $r \in R$ and v **mb-viable**. Let q be the process performing r . Then u is equivalent to an **mb-viable** sequence if and only if there is no non-empty $(\prec_{\text{hb}} \cup \prec_{\text{mb}})$ -path from $v[i]$ to $v[j]$ for some $i < j$ such that $v[i]$ is an **unmatched** send to q and $v[j]$ is the send matching r in u .

The next lemma shows how to check the existence of a $(\prec_{\text{hb}} \cup \prec_{\text{mb}})$ -path between two positions of an **ms-sequence**, using the automaton from Lemma 4.6.

► **Lemma 5.15.** One can construct an automaton \mathcal{D} with $O(|\mathbb{P}|)$ states over the alphabet $(S \cup \bar{S}) \times \{\circ, \bullet\} \cup \{\#\}$ with the following properties:

1. \mathcal{D} accepts only words from $(\Sigma^* \#)^*$ containing exactly two positions in $(S \cup \bar{S}) \times \{\bullet\}$.
2. For every $u = u_1 *_{\text{mb}} \dots *_{\text{mb}} u_n$ **mb-viable**, with each u_i an **exchange**, \mathcal{D} accepts tagged $v = \text{ms}(u_1) \# \dots \# \text{ms}(u_n)$ iff, there is a $(\prec_{\text{hb}} \cup \prec_{\text{mb}})$ -path from $u[i]$ to $u[j]$, where $i < j$ are the positions of u corresponding to the positions tagged by \bullet in v .

We have now all ingredients to show our main result. We use Lemma 5.13 to guess the witness sequence, **exchange** by **exchange**, and to be sure that the sequence is **mb-viable** we rely on Lemmas 5.14 and 5.15, complementing the automaton on-the-fly. The lower bound is obtained, as before, by reduction from the intersection emptiness problem for finite automata.

► **Theorem 5.16.** The question whether a **CFM** is **mb-synchronizable** is PSPACE-complete.

Proof. For the upper bound we use Lemma 5.13 to guess a minimal non-**mb-synchronizable** sequence $u = vr$. Recall that q is the process executing r , and s the matching send of r in u . First we rely on the automaton of Lemma 5.10 in order to guess the **atomic exchanges** v_i composing v on-the-fly. At the same time we guess the subsequence of indices $i_1 < \dots < i_k$ and the events that witness that $i_j \prec_{\text{mb}}^v i_{j+1}$ (cf. Definition 5.4).

We keep record of the current pair (g, D) , where g is a global state of the **CFM** and D a set of **deaf** processes, as we guess each v_i , to check that the sequence v labels an **execution**. When we process v_{i_k} , we remember its alphabet over $S \cup \bar{S}$ until we guess $v_{i_{k+1}}$, and check that $i_k \prec_{\text{mb}}^v i_{k+1}$ (cf. Remark 5.5). We also guess m as of item (5) in Lemma 5.13, and

check (5). After we have done v_n , we must have reached (g, D) such that the receive r can be done in state g . By verifying that u is **mb-viable** as described below, we know that s is **matched** with r .

We check that u is **mb-viable** with Lemma 5.15. From Lemma 5.14 we know that u is **mb-viable** iff there is no **unmatched** send s' to q s.t. there is a $(\leq_{\text{hb}} \cup <_{\text{mb}})$ -path from s' to s in v . We use the complement \mathcal{D}^{co} of \mathcal{D} , which is exponential in $|\mathbb{P}|$ but can be constructed on-the-fly in linear space. We make one copy $\mathcal{D}^{co}(p')$ of \mathcal{D}^{co} for every process $p' \neq q$. Each $\mathcal{D}^{co}(p')$ tags the first **unmatched** send of type $p'!q$ and s with \bullet . We make every $\mathcal{D}^{co}(p')$ read the tagged $\text{ms}(v_1)\#\dots\#\text{ms}(v_n)$ by adding the $\#$ after each **atomic mb-exchange** we read. Each $\mathcal{D}^{co}(p')$ should accept. This guarantees that no send of type $\bar{p}'!q$ has a $(\leq_{\text{hb}} \cup <_{\text{mb}})$ -path to s .

For the lower bound, we use the same reduction as in Theorem 3.6, and if we reach $(\text{accept})_{p \in \mathbb{P}}$, we use two other processes to do a non-**mb-synchronizable** gadget (see the full version of the paper [6]). This way, the **CFM** is **mb-synchronizable** if and only if the intersection of the automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ is empty. \blacktriangleleft

Theorem 5.16 yields two interesting corollaries. In the statements below we say that a **CFM** is k -**mb-synchronizable** if for every **trace** $u \in \text{Tr}_{\text{mb}}(\mathcal{A})$, we have $u \equiv u_1 * \dots * u_n$ for some **mb-exchanges** u_i where $|u_i| \leq k$. The next result has been shown decidable in [2] (with non-elementary complexity):

► **Theorem 5.17.** *Let k be an integer given in binary. The question whether a **CFM** is k -**mb-synchronizable** is PSPACE-complete. The lower bound already holds for k in unary.*

Proof. Using Theorem 5.16 we first check that the **CFM** is **mb-synchronizable**. Then we use the automaton \mathcal{C} from Lemma 3.5 to compute pairs (g, D) of global state and set of **deaf** processes that are reachable by some **mb-synchronous** sequence. Finally we check whether the automaton of Lemma 5.10 accepts only **exchanges** of size at most k . Since the size of our automata is exponential the test can be done in PSPACE. The lower bound can be obtained as in the proof of Theorem 5.16 (see Figure 2). \blacktriangleleft

For the second result and weak synchronizability, decidability was obtained in [12]. Our proof based on automata seems more direct and simpler than the one of [12]:

► **Theorem 5.18.** *The question whether for a given **CFM** \mathcal{A} there exists some k such that \mathcal{A} is k -**mb-synchronizable**, is PSPACE-complete.*

Proof. For the upper bound we proceed as in the previous proof. The difference is that at the end we check whether the automaton of Lemma 5.10 accepts an infinite language from a reachable pair (g, D) . The language of this automaton is infinite iff there is no k as stated by the theorem. The lower bound can be obtained as in the proof of Theorem 5.16. \blacktriangleleft

6 Conclusion

We have introduced a novel automata-based approach to reason about communication in the **sr-round mailbox** model. We showed that knowing whether a system complies with this model is PSPACE-complete. An interesting theoretical question is whether we can apply similar techniques to other types of communication. On the practical side it would be interesting to implement our algorithms and compare e.g. with existing tools like Soter [8] that targets safety properties for a relaxed model of Erlang. Our automata-based techniques may be easier to implement than previous approaches, and could even adapt to a dynamic setting.

References

- 1 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996. doi:10.1006/INCO.1996.0053.
- 2 Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes, and Amrita Suresh. A unifying framework for deciding synchronizability. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.14.
- 3 Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II 30*, pages 372–391. Springer, 2018.
- 4 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM (JACM)*, 30(2):323–342, 1983.
- 5 Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Comput.*, 22(1):49–71, 2009. doi:10.1007/S00446-009-0084-6.
- 6 Romain Delpy, Anca Muscholl, and Grégoire Sutre. An automata-based approach for synchronizable mailbox communication, 2024. arXiv:2407.06968.
- 7 Cinzia Di Giusto, Laetitia Laversa, and Etienne Lozes. On the k-synchronizability of systems. In *23rd International Conference on Foundations of Software Science and Computer Systems (FOSSACS 2020)*, volume 12077, pages 157–176. Springer, 2020.
- 8 Emanuele D’Osualdo, Jonathan Kochems, and C.-H. Luke Ong. Automatic verification of erlang-style concurrency. In Francesco Logozzo and Manuel Fähndrich, editors, *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 454–476. Springer, 2013. doi:10.1007/978-3-642-38856-9_24.
- 9 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.
- 10 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80(1-3):147–167, 2007.
- 11 Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. Infinite-state high-level mscs: Model-checking and realizability. *J. Comput. Syst. Sci.*, 72(4):617–647, 2006. doi:10.1016/J.JCSS.2005.09.007.
- 12 Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. Guessing the buffer bound for k-synchronizability. *Int. J. Found. Comput. Sci.*, 34(8):1051–1076, 2023. doi:10.1142/S0129054122430018.
- 13 Loïc Hélouët and Pierre Le Maigat. Decomposition of message sequence charts. In Edel Sherratt, editor, *SAM 2000, 2nd Workshop on SDL and MSC, Col de Porte, Grenoble, France, June 26-28, 2000*, pages 47–60. Verimag, IRISA, SDL Forum, 2000.
- 14 Dietrich Kuske and Anca Muscholl. Communicating automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 1147–1188. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/AUTOMATA-2/9.
- 15 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
- 16 Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 350–373. Springer, 2023. doi:10.1007/978-3-031-37709-9_17.

- 17 Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 35:1–35:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.35.
- 18 Felix Stutz. Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPICs*, pages 32:1–32:31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ECOOP.2023.32.

Regular Games with Imperfect Information Are Not That Regular

Laurent Doyen 

CNRS & LMF, ENS Paris-Saclay, France

Thomas Soullard 

LMF, ENS Paris-Saclay & CNRS, France

Abstract

We consider two-player games with imperfect information and the synthesis of a randomized strategy for one player that ensures the objective is satisfied almost-surely (i.e., with probability 1), regardless of the strategy of the other player. Imperfect information is modeled by an indistinguishability relation describing the pairs of histories that the first player cannot distinguish, a generalization of the traditional model with partial observations. The game is regular if it admits a regular function whose kernel commutes with the indistinguishability relation.

The synthesis of pure strategies that ensure all possible outcomes satisfy the objective is possible in regular games, by a generic reduction that holds for all objectives. While the solution for pure strategies extends to randomized strategies in the traditional model with partial observations (which is always regular), a similar reduction does not exist in the more general model. Despite that, we show that in regular games with Büchi objectives the synthesis problem is decidable for randomized strategies that ensure the outcome satisfies the objective almost-surely.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Probabilistic computation

Keywords and phrases Imperfect-information games, randomized strategies, synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.23

Related Version *Full Version*: <https://arxiv.org/abs/2403.20133>

1 Introduction

We consider the synthesis problem for two-player turn-based games with imperfect information, a model that has applications in several areas of computer-science, including the design of multi-agent systems [18], logics with uncertainty in planning and AI [5, 14], program synthesis [10], and automata theory [1]. The synthesis problem asks to decide the existence of (and if so, to construct) a strategy for one player that ensures a given objective is satisfied with the largest possible probability, regardless of the choices of the other player. We focus mostly on reachability objectives, and will also discuss ω -regular objectives [17].

Synthesis provides a natural formulation for the design of a reactive system that interacts with an unknown environment. The interactive nature of reactive systems is modeled by a two-player game between the system (the player) and an adversarial environment, and the limited access of the system to the current state of the game is modeled by imperfect information.

A simple example of a turn-based game with imperfect information is repeated matching pennies, where the environment (secretly) chooses head or tail (say, denoted by $x \in \{0, 1\}$), then the player chooses $y \in \{0, 1\}$ without seeing x , and wins if $y = x$. If $y \neq x$, the game repeats for one more round; the player loses if the game repeats forever. It is clear that there exists no pure (deterministic) strategy that is winning in repeated matching pennies, as for all sequences $\bar{x} = x_1 \dots x_i$ and $\bar{y} = y_1 \dots y_i$ that represent a history of the game (namely, the sequence of moves played in the first i rounds), given x_{i+1} (chosen by the



© Laurent Doyen and Thomas Soullard;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 23; pp. 23:1–23:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

environment), the choice of y_{i+1} (by the player) must be made independently of x_{i+1} , thus we may have $x_{i+1} = 1 - y_{i+1}$, showing that the game may repeat forever. That the player cannot ensure to win is counter-intuitive, and this is because pure (deterministic) strategies are considered. With randomized strategies, the player can win with probability 1, simply by choosing $y_i \in \{0, 1\}$ with uniform probability at every round. The power of randomization for decision making with imperfect information is well known and occurs in many situations [16, Section 1.3], including gene mutation in biology, penalty kick in sports, bluffing in card games, etc. It is also known that, even for simple objectives like reachability, optimal strategies may need memory [21].

In the sequel, we distinguish the synthesis of a pure strategy (which we call a sure winning strategy, as it aims to ensure all possible outcomes satisfy the objective), and the synthesis of a randomized strategy (which we call an almost-sure winning strategy, as it aims to ensure that the objective is satisfied with probability 1). Note that the synthesis problem for strategies that ensure the objective is satisfied with probability at least λ (given $0 < \lambda < 1$) is undecidable for probabilistic automata [20], a model that can be reduced to two-player turn-based games with imperfect information [11].

Given a finite alphabet Γ of moves, we denote by Γ^* the set of all histories in a game. The traditional model of imperfect information in games is a (regular) observation function defined on Γ^* that assigns to each history of the game a color that is visible to the player, while the history itself is not visible [22]. We call them partial-observation games, or games *à la Reif*. This model of games with imperfect information (even turn-based and non-stochastic) with randomized strategies generalizes many models such as concurrent games (e.g., matching pennies), Markov chains, Markov decision processes, and stochastic games [11, Theorem 5]. The synthesis of pure strategies in partial-observation games *à la Reif* with ω -regular objectives can be done using automata-based techniques analogous to the subset construction for finite automata [22, 21]. The synthesis of randomized (almost-sure winning) strategies can be done for reachability (and Büchi) objectives with a more involved technique based on the same approach of subset construction. The synthesis problem is EXPTIME-complete in both cases. Note that for ω -regular (and even for coBüchi) objectives, the synthesis problem with randomized strategies is undecidable [4].

A more abstract (and more general) model of imperfect information is given by a function $f : \Gamma^* \times \Gamma^* \rightarrow \{0, 1\}$ that specifies an indistinguishability relation [6, 7]: two histories τ, τ' are indistinguishable if $f(\tau, \tau') = 1$. In analogy to the partial-observation model, indistinguishable histories have the same length and their prefixes (of the same length) are also indistinguishable. Hence this function can be viewed as a set of pairs of finite words of the same length (over alphabet Γ), or alternatively as a language of finite words over the alphabet $\Gamma \times \Gamma$. Applications of this model include the synthesis of full-information protocols, which cannot be expressed by a partial-observation game *à la Reif* [7, Section 6].

A generic approach to the synthesis problem with this more general model is to construct a so-called rectangular morphism h defined on Γ^* with finite range [7]. Note that a morphism with finite range is a regular function (which can be defined by a finite-state automaton with output), and we say that a game is *regular* if such a rectangular morphism h exists. For instance, all partial-observation games *à la Reif* are regular as they admit a natural rectangular morphism, which is essentially the subset construction. The existence of a rectangular morphism guarantees that the information tree has a finite bisimulation quotient, and thus (pure) strategies can be transferred back and forth (by copying the action played in a bisimilar position of the other game), preserving the outcome of the game for arbitrary objectives [7]. Hence, for the synthesis of pure strategies, regular games with imperfect information can be reduced to finite games with perfect information.

In this paper, we consider the synthesis problem with randomized strategies, which is central in games with imperfect information (such as matching pennies). Given the existence of a finite bisimulation quotient in regular games, the fact that a solution based on a rectangular morphism (the subset construction) works for the synthesis of randomized strategies in reachability games *à la Reif* gives hope that synthesis in regular games is solvable along a similar path, by a reduction to a simpler equivalent game from which randomized strategies can be transferred back and forth, copying both the action and the probability of playing that action. The hope is reinforced by the existence of a bijection between the set of randomized strategies in a game *à la Reif* and the set of strategies in the simpler equivalent game, which relates strategies with same (probabilistic measure on the) outcome for arbitrary objectives [21, Theorem 4.2].

Surprisingly, we show that even a much weaker variant of such a reduction does not exist for regular games. First the hope for a bijective transfer of strategies is not reasonable because the information tree of regular games have in general unbounded branching [6], whereas the simpler equivalent games induced by (finite) rectangular morphism have a bounded-branching information tree. On the other hand, we show that even a non-bijective transfer of strategies, as general as it can reasonably be, may not exist for some regular game with a specific rectangular morphism (Section 4.3). Despite their nice structural properties, the existence of a finite bisimulation in their information tree, and their apparent similarity with games *à la Reif* where a bijection between randomized strategies is induced by a rectangular morphism, regular games are not as well-behaved as games *à la Reif*.

In this context, we present an algorithmic solution of the synthesis problem with randomized strategies for regular games with a reachability objective. The solution exploits the properties of rectangular morphisms to define a fixpoint computation with complexity quadratic in the size of the range of the rectangular morphism (which is of exponential size in the case of games *à la Reif* [21], and of non-elementary size in the case of full-information protocols [7]). Our algorithm shares the common features of the solutions of almost-sure reachability objectives in Markov decision processes [12], concurrent games [13], and games *à la Reif* [21], namely a nested fixpoint computation that iteratively constructs the almost-sure winning set by computing the set S_{i+1} of states from which the player can win with positive probability while ensuring to never leave the set S_i , where S_0 is the entire state space (Section 3). Our solution immediately extends to Büchi objectives, using reductions to reachability objectives from the literature.

In conclusion, this paper generalizes the positive (decidability) results about randomized strategies from games *à la Reif* to regular games (namely for reachability and Büchi objectives), whereas the reductions that worked in games *à la Reif* for arbitrary objectives no longer hold.

Omitted proofs are provided in the appendix.

2 Definitions

We recall basic definitions from logic and automata theory that will be useful in the rest of the paper, and then we discuss our model of games.

2.1 Basic notions

Regular functions. A *Moore machine* is a finite-state automaton with outputs, consisting of a finite input alphabet Γ , a finite output alphabet Σ , and a tuple $\mathcal{M} = \langle Q, q_\varepsilon, \delta, \lambda \rangle$ with a finite set Q of states (sometimes called the memory), an initial state $q_\varepsilon \in Q$, a (deterministic) transition function $\delta : Q \times \Gamma \rightarrow Q$, and an output function $\lambda : Q \rightarrow \Sigma$.

23:4 Regular Games with Imperfect Information Are Not That Regular

We extend the transition function to input words in the natural way, defining $\delta : Q \times \Gamma^* \rightarrow Q$ by $\delta(q, \varepsilon) = q$ for all states $q \in Q$, and inductively $\delta(q, \tau c) = \delta(\delta(q, \tau), c)$ for all histories $\tau \in \Gamma^*$ and moves $c \in \Gamma$.

The object of interest is the extension of the output function to a function $\lambda : \Gamma^* \rightarrow \Sigma$ defined by $\lambda(\tau) = \lambda(\delta(q_\varepsilon, \tau))$ for all histories $\tau \in \Gamma^*$. A function λ defined on Γ^* is *regular* if it is (the extension of) the output function of a Moore machine. The *cumulative extension* of λ is denoted by $\hat{\lambda} : \Gamma^* \rightarrow \Sigma^*$ and defined by $\hat{\lambda}(\varepsilon) = \varepsilon$ and inductively $\hat{\lambda}(\tau c) = \hat{\lambda}(\tau)\lambda(\tau c)$, thus $\hat{\lambda}(c_1 \dots c_k) = \lambda(c_1)\lambda(c_1 c_2) \dots \lambda(c_1 \dots c_k)$. Note that $\lambda = \text{last} \circ \hat{\lambda}$ where $\text{last}(c_1 \dots c_k) = c_k$. The function $\hat{\lambda}$ is further extended to infinite words as expected: $\hat{\lambda}(\pi) = \lambda(c_1)\lambda(c_1 c_2) \dots$ for all $\pi = c_1 c_2 \dots \in \Gamma^\omega$.

Fixpoint formulas. We briefly recall the interpretation of μ -calculus fixpoint formulas [8, 9] based on the Knaster-Tarski theorem. Given a monotonic function $\psi : 2^Q \rightarrow 2^Q$ (i.e., such that $X \subseteq Y$ implies $\psi(X) \subseteq \psi(Y)$), the expression $\nu Y. \psi(Y)$ is the (unique) greatest fixpoint of ψ , which can be computed as the limit of the sequence $(Y_i)_{i \in \mathbb{N}}$ defined by $Y_0 = Q$, and $Y_i = \psi(Y_{i-1})$ for all $i \geq 1$. Dually, the expression $\mu X. \psi(X)$ is the (unique) least fixpoint of ψ , and the limit of the sequence $(X_i)_{i \in \mathbb{N}}$ defined by $X_0 = \emptyset$, and $X_i = \psi(X_{i-1})$ for all $i \geq 1$. If $|Q| = n$, then it is not difficult to see that the limit of those sequences is reached after at most n iterations, $X_n = X_{n+1}$ and $Y_n = Y_{n+1}$.

On equivalence relations. Given an equivalence relation $\sim \subseteq S \times S$ and a set $T \subseteq S$, the *closure* of T under \sim is the set $[T]_\sim = \{t \in S \mid \exists t' \in T : t \sim t'\}$, and the *interior* of T under \sim is the set $\text{int}_\sim(T) = \{t \in S \mid [t]_\sim \subseteq T\}$. The closure and interior are monotone operators, that is if $T \subseteq U$, then $[T]_\sim \subseteq [U]_\sim$ and $\text{int}_\sim(T) \subseteq \text{int}_\sim(U)$. Note the duality $S \setminus \text{int}_\sim(T) = [S \setminus T]_\sim$, the complement of the interior of T is the closure of the complement of T . We say that T is *closed* under \sim if $T = [T]_\sim$ (or equivalently $T = \text{int}_\sim(T)$).

2.2 Games with imperfect information

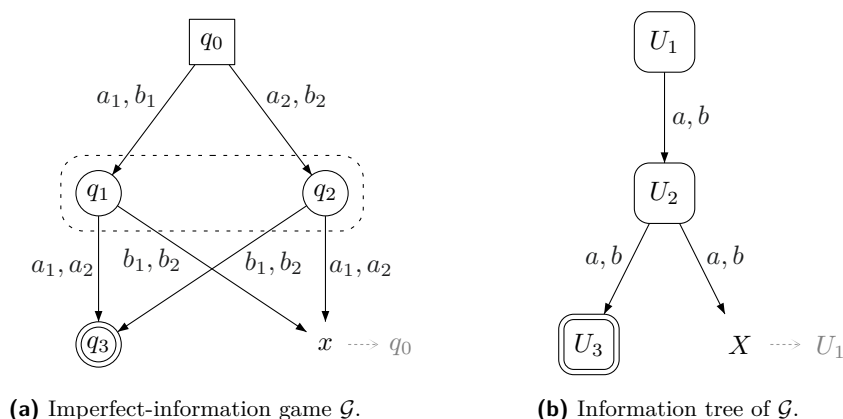
We consider an abstract model of games, given by a (nonempty) set A of actions, a set Γ of moves, and a surjective function $\text{act} : \Gamma \rightarrow A$. The game is played over infinitely many rounds (or steps). In each round, the player chooses an action $a \in A$, then the environment chooses a move $c \in \Gamma$ such that $\text{act}(c) = a$. We say that the move c is *supported* by the action a .

The outcome of the game is an infinite sequence $\pi \in \Gamma^\omega$ of moves of the environment, called a *play*, from which we can reconstruct the infinite sequence of actions of the player using the function act . A finite sequence of moves is called a *history*.

The winning condition is a set $W \subseteq \Gamma^\omega$ of plays that are winning for the player. We specify the winning condition by the combination of a coloring function $\lambda : \Gamma^* \rightarrow C$ for some finite alphabet C of colors, and a logical objective $L \subseteq C^\omega$. The coloring function is regular, and specified by a Moore machine. The induced winning condition is the set $W = \{\pi \in \Gamma^\omega \mid \hat{\lambda}(\pi) \in L\}$ of plays whose coloring satisfies the objective.

We consider the following classical objectives, for $C = \{0, 1\}$:

- the reachability objective is $\text{Reach} = 0^*1\{0, 1\}^\omega$ and for $x \in \mathbb{N}$ let $\text{Reach}^{\leq x} = \bigcup_{i \leq x} 0^i 1 \{0, 1\}^\omega$;
- the safety objective is $\text{Safe} = 0^\omega = \{0, 1\}^\omega \setminus \text{Reach}$;
- the Büchi objective is $\text{Büchi} = (0^*1)^\omega$;
- the coBüchi objective is $\text{coBüchi} = \{0, 1\}^* 0^\omega = \{0, 1\}^\omega \setminus \text{Büchi}$;



■ **Figure 1** A game with imperfect information representing matching pennies, and its information tree.

It is convenient to consider that the coloring function λ is part of a game instance, while the objective L can be specified independently of the game. This allows to quantify separately the game instances and the objectives. We recall that the traditional games played on graphs [17, 3] are an equivalent model, in particular they can be translated into our abstract model of game, for example by letting moves be the edges of the game graph and the corresponding action be the label of the edge, while the graph structure can be encoded in the winning condition [23]. When we refer to a game played on a graph, we assume that all move sequences that contain two consecutive moves (i.e., edges) $c_i = (q_1, q_2)$ and $c_{i+1} = (q_3, q_4)$ such that $q_2 \neq q_3$ belong to the winning condition (for the player).

Imperfect information is represented by an indistinguishability relation $\sim \in \Gamma^* \times \Gamma^*$ that specifies the pairs of histories that the player cannot distinguish. An indistinguishability relation is an equivalence that satisfies the following conditions [6], for all histories $\tau, \tau' \in \Gamma^*$ and moves $c, c' \in \Gamma$: (i) if $\tau \sim \tau'$, then $|\tau| = |\tau'|$ (indistinguishable histories have the same length); (ii) if $\tau c \sim \tau' c'$, then $\tau \sim \tau'$ (the relation is prefix-closed) and $\text{act}(c) = \text{act}(c')$ (the action is visible). Indistinguishability relations may be specified using two-tape automata, but the results of this paper hold for arbitrary indistinguishability relations.

Given a history $\tau \in \Gamma^*$ we call the equivalence class $[\tau]_{\sim} = \{\tau' \in \Gamma^* \mid \tau' \sim \tau\}$ containing τ an *information set*. A function λ defined on Γ^* is *information consistent* if λ is constant over every information set: $\lambda(\tau) = \lambda(\tau')$ for all indistinguishable histories $\tau \sim \tau'$. An equivalent requirement is that the cumulative extension $\hat{\lambda}$ is constant over every information set, since indistinguishability relations are prefix-closed. We always require the coloring function of a game to be information consistent (with respect to the indistinguishability relation of the game). For every action $a \in A$, we define a successor relation \rightarrow_a over information sets, where $[\tau]_{\sim} \rightarrow_a [\tau c]_{\sim}$ for all moves $c \in \Gamma$ such that $\text{act}(c) = a$. The relations \rightarrow_a are obviously acyclic, and induce a structure called the *information tree*. Note that there is a natural extension of the coloring function λ to information sets, defined by $\lambda([\tau]_{\sim}) = \lambda(\tau)$ for all histories $\tau \in \Gamma^*$, since λ is information-consistent.

Given the above definitions, we represent a game with imperfect information as a tuple $\mathcal{G} = \langle A, \text{act}, \sim, \lambda \rangle$, together with an objective $L \subseteq C^\omega$, where the (finite) alphabet of moves is Γ , and the (finite) alphabet of colors is C . The information tree of \mathcal{G} is denoted by $\mathcal{U}_{\sim}(\mathcal{G})$ (consisting of the information sets of \mathcal{G} , the successor relations \rightarrow_a , and the coloring

λ extended to information sets). The special case of perfect-information games corresponds the indistinguishability relation \sim being the identity (or equivalently by the information sets $[\tau]_{\sim} = \{\tau\}$ being singletons for all histories $\tau \in \Gamma^*$).

In figures, we present a very informal (but readable) description of games. The main components of a game that figures represent are the coloring function and the indistinguishability relation. Strictly speaking, the figures do not show a game but provide sufficient information to reconstruct the Moore machine defining the coloring function, and the two-tape automaton defining the indistinguishability relation. It should be possible to infer the color of a history (when its color is relevant), and the information set containing a given history. States are normally depicted as circles, but we may use boxes to emphasize that the action choice of the player is not relevant and only the move of the environment determines the successor state.

In the game \mathcal{G} of Figure 1a, representing matching pennies, the player has two actions, $A = \{a, b\}$, and the environment has two possible responses for each of them, thus four moves, $\Gamma = \{a_1, a_2, b_1, b_2\}$ where $\text{act}(a_i) = a$ and $\text{act}(b_i) = b$ for $i = 1, 2$. Intuitively, the adversarial environment chooses head (1) or tail (2) in the first round (at q_0) by choosing a response 1 or 2 (the action of the player is not relevant) that the player cannot observe (all histories of length 1 are indistinguishable, as suggested by the dashed line in Figure 1a). The player chooses head (action a) or tail (action b) in the next round, wins if the choice is matching the environment's choice, and replays the same game otherwise (x is a placeholder for the whole tree of Figure 1a).

The information tree of \mathcal{G} is shown in Figure 1b where $U_1 = [\varepsilon]_{\sim} = \{\varepsilon\}$ and $U_2 = [a_1]_{\sim} = \{a_1, a_2, b_1, b_2\} = \Gamma$. The information set U_3 contains eight of the sixteen histories of length 2. The node X is a placeholder for a tree that is isomorphic to the tree rooted at U_1 .

We recall that the nodes in information trees have a finite number of successors (there is at most $|\Gamma|^n$ information sets containing histories of length n), however the branching degree in a given information tree may be unbounded [6]. The partial-observation games *à la Reif* [22] are characterized by their information tree having bounded branching [6, Theorem 7].

2.3 Strategies and outcome

A *probability distribution* on a finite set S is a function $d : S \rightarrow [0, 1]$ such that $\sum_{s \in S} d(s) = 1$. A *Dirac distribution* assigns probability 1 to a (unique) element $s \in S$. We denote by $\mathcal{D}(S)$ the set of all probability distributions on S .

A strategy for the player is a function $\alpha : \Gamma^* \rightarrow \mathcal{D}(A)$ that is information consistent, thus $\alpha(\tau) = \alpha(\tau')$ for all indistinguishable histories $\tau \sim \tau'$. The strategy α is *pure* (or, deterministic) if $\alpha(\tau)$ is a Dirac distribution for all $\tau \in \Gamma^*$. A strategy for the environment is a function $\beta : \Gamma^* \times A \rightarrow \mathcal{D}(\Gamma)$ such that for all histories $\tau \in \Gamma^*$, actions $a \in A$, and moves $c \in \Gamma$, if the move c is played with positive probability $\beta(\tau, a)(c) > 0$, then it is supported by the action a , $\text{act}(c) = a$. The strategy β is *pure* (or, deterministic) if $\beta(\tau, a)$ is a Dirac distribution for all $\tau \in \Gamma^*$ and $a \in A$. We denote by $\mathcal{A}_{\mathcal{G}}$ (resp., by $\mathcal{B}_{\mathcal{G}}$) the set of all strategies for the player (resp., for the environment), and by $\mathcal{A}_{\mathcal{G}}^{\text{pure}}$ (resp., by $\mathcal{B}_{\mathcal{G}}^{\text{pure}}$) the set of all pure strategies for the player (resp., for the environment).

A pair of strategies (α, β) for the player and the environment induce a probability measure $\text{Pr}^{\alpha, \beta}$ on the sigma-algebra over the set of (infinite) plays. By Carathéodory's extension theorem, it is sufficient to define the probability measure over cylinder sets spanned by (finite) prefixes of plays. We define a family of probability measures $\text{Pr}_{\tau}^{\alpha, \beta}$ where $\tau \in \Gamma^*$ corresponds to starting the game after history τ . We set $\text{Pr}^{\alpha, \beta} = \text{Pr}_{\varepsilon}^{\alpha, \beta}$. Given $\rho \in \Gamma^*$, the measure of the cylinder set $\text{Cyl}(\rho) = \rho\Gamma^{\omega} = \{\pi \in \Gamma^{\omega} \mid \rho \text{ is a prefix of } \pi\}$ is defined as follows:

- if ρ is a prefix of τ , then

$$\Pr_{\tau}^{\alpha, \beta}(\text{Cyl}(\rho)) = 1;$$

- if $\rho = \tau c_1 \dots c_k$ where $c_1, \dots, c_k \in \Gamma$ for some $k \geq 1$, then

$$\Pr_{\tau}^{\alpha, \beta}(\text{Cyl}(\rho)) = \prod_{i=1}^k \alpha(\tau c_1 \dots c_{i-1})(\text{act}(c_i)) \cdot \beta(\tau c_1 \dots c_{i-1}, \text{act}(c_i))(c_i);$$

- all other cylinder sets have measure 0.

Given an objective $L \subseteq C^\omega$ defined on colors, assuming the game \mathcal{G} (and thus $\hat{\lambda}$) is clear from the context we take the freedom to denote by $\Pr^{\alpha, \beta}(L)$ the probability $\Pr^{\alpha, \beta}(\{\pi \in \Gamma^\omega \mid \hat{\lambda}(\pi) \in L\})$ that the objective L is satisfied by an outcome of the pair of strategies (α, β) in \mathcal{G} . A strategy α of the player is *almost-sure winning* from an information set $[\tau]_\sim$ (or simply *almost-sure winning* if $\tau = \varepsilon$) for objective L if $\Pr_{\tau'}^{\alpha, \beta}(L) = 1$ for all $\tau' \in [\tau]_\sim$ and all strategies $\beta \in \mathcal{B}_{\mathcal{G}}$ of the environment. We are interested in solving the synthesis problem, which is to decide, given a game \mathcal{G} and objective L , whether there exists an almost-sure winning strategy in \mathcal{G} for L .

Note that the definition of almost-sure winning is equivalent to a formulation where only *pure* strategies of the environment are considered, that is where we require $\Pr_{\tau'}^{\alpha, \beta}(L) = 1$ for all $\beta \in \mathcal{B}_{\mathcal{G}}^{\text{pure}}$ only, as once a strategy for the player is fixed, the environment plays in a (possibly infinite-state) Markov decision process, for which pure strategies are sufficient [19, 11].

We say that a history $\tau \in \Gamma^*$ is *compatible* with α and β if $\Pr^{\alpha, \beta}(\text{Cyl}(\tau)) > 0$. An *outcome* of the pair of strategies (α, β) is a play $\pi \in \Gamma^\omega$ all of whose prefixes are compatible with α and β . We note that if $\Pr^{\alpha, \beta}(\text{Cyl}(\tau)) > 0$, then $\Pr_{\tau}^{\alpha, \beta}(\text{Cyl}(\tau c_1 c_2 \dots c_n)) = \Pr^{\alpha, \beta}(\text{Cyl}(\tau c_1 c_2 \dots c_n) \mid \text{Cyl}(\tau))$.

3 Almost-Sure Reachability

Throughout this section, we consider games with a (fixed) reachability objective $\text{Reach} = 0^*1\{0, 1\}^\omega$ over color alphabet $C = \{0, 1\}$, thus a play is winning if it has a finite prefix with color 1 according to the coloring function λ of the game. By extension, we say that a history τ is winning if $\lambda(\tau) = 1$. We also assume without loss of generality that if $\lambda(\tau) = 1$, then $\lambda(\tau c) = 1$ for all $\tau \in \Gamma^*$ and all $c \in \Gamma$. In the rest of this section, we fix a game \mathcal{G} as defined in Section 2.2 and we show how to decide the existence of an almost-sure winning strategy in games with a reachability objective, relying on the existence of a rectangular morphism (defined below). Two interesting classes of games are known to admit rectangular morphisms, namely the partial-observation games *à la Reif* and the full-information protocols [7].

3.1 Regular games with imperfect information

A *rectangular morphism* for \mathcal{G} is a surjective function $h : \Gamma^* \rightarrow P$ (for some finite set P) such that for all $\tau, \tau' \in \Gamma^*$ and $c \in \Gamma$:

Rectangularity¹ if $h(\tau) = h(\tau')$, then $h([\tau]_\sim) = h([\tau']_\sim)$,

Morphism if $h(\tau) = h(\tau')$, then $h(\tau c) = h(\tau' c)$,

Refinement if $h(\tau) = h(\tau')$, then $\lambda(\tau) = \lambda(\tau')$.

¹ Rectangularity is equivalent to the kernel $H = \{(\tau, \tau') \mid h(\tau) = h(\tau')\}$ of h commuting with the indistinguishability relation, that is $H \circ \sim = \sim \circ H$.

Note that, as h is a finite-valued morphism, it can be represented by an automaton (whose output is not relevant) with input alphabet Γ and state space P . Games that admit a finite-valued rectangular morphism are called *regular*. Define the relation $\approx = \{(h(\tau), h(\tau')) \mid \tau \sim \tau'\}$ and recall a non-trivial fundamental result.

► **Lemma 1** ([7], Lemma 3 & Lemma 4). *The relation \approx is an equivalence and $h([\tau]_{\sim}) = [h(\tau)]_{\approx}$ for all $\tau \in \Gamma^*$.*

We call the elements of P *abstract states*. We extend the relation \approx to elements of $P \times A$ as follows: for all $p, p' \in P$ and $a, a' \in A$, let $(p, a) \approx (p', a')$ if $p \approx p'$ and $a = a'$.

Thanks to the morphism property, there exists a function $\delta^P : P \times \Gamma \rightarrow P$ such that $\delta^P(h(\tau), c) = h(\tau c)$ for all $\tau \in \Gamma^*$ and $c \in \Gamma$. Define the set $P_F = \{p \in P \mid \exists \tau \in \Gamma^* : h(\tau) = p \wedge \lambda(\tau) = 1\}$ of target (or final) states, which are the images by h of winning histories. Note that the coloring function is information-consistent and therefore the set P_F is closed under \approx by Lemma 1. Moreover, by our assumption on the coloring function the states in P_F form a sink set, that is $\delta^P(p, c) \in P_F$ for all states $p \in P_F$ and moves $c \in \Gamma$.

Rectangular morphisms are central to the solution of the synthesis problem for *pure* strategies [7], by showing that the following abstract game is equivalent to \mathcal{G} : starting from $p_0 = h(\varepsilon)$, the game is played in rounds where each round starts with a value p and the player chooses an action $a \in A$, the next round starts in p' chosen by the environment such that $p' \approx \delta^P(p, c)$ for some move $c \in \Gamma$ such that $\text{act}(c) = a$. The player wins if a value in P_F eventually occurs along a play. We may view the elements of P as positions of the player, and elements of $P \times A$ as positions of the environment. We further discuss this abstract game in Section 4.2.

An abstract state $p \in P$ is *existentially winning* if there exists a history $\tau \in \Gamma^*$ such that $h(\tau) = p$ and the player has an *almost-sure winning* strategy from $[\tau]_{\sim}$. We denote by P_{\exists} the set of all existentially winning states. Note that the set P_{\exists} is closed under \approx , since for all $p \in P_{\exists}$ and $p' \approx p$, by Lemma 1 there exists $\tau' \in [\tau]_{\sim}$, such that $h(\tau') = p'$ and the player has an *almost-sure winning* strategy from $[\tau']_{\sim} = [\tau]_{\sim}$.

An abstract state $p \in P$ is *universally winning* if for all histories $\tau \in \Gamma^*$ such that $h(\tau) = p$, the player has an *almost-sure winning* strategy from $[\tau]_{\sim}$. We denote by P_{\forall} the set of all universally winning states. Note that the set P_{\forall} is closed under \approx , by an argument analogous to the case of P_{\exists} . As we consider a reachability objective, it is easy to see that $P_F \subseteq P_{\forall} \subseteq P_{\exists}$ (the latter inclusion holds since h is surjective).

Note that in games of perfect information, the existentially and universally winning states coincide by definition since information sets are singletons, $P_{\forall} = P_{\exists}$. A corollary of our results is that $P_{\forall} = P_{\exists}$ also holds in games of imperfect information with a reachability objective.

3.2 Algorithm

We present an algorithm to decide if the player is almost-sure winning for reachability in a game of imperfect information, assuming we have a rectangular morphism.

Given a set $X \subseteq P \cup (P \times A)$, define the *predecessor* operator as follows:

$$\begin{aligned} \text{Pre}(X) &= \{p \in P \mid \exists a \in A : (p, a) \in X\} \\ &\cup \\ &\{(p, a) \in P \times A \mid \forall c : \text{act}(c) = a \implies \delta^P(p, c) \in X\} \end{aligned}$$

It is easy to verify that if $X_1 \subseteq X_2$, then $\text{Pre}(X_1) \subseteq \text{Pre}(X_2)$, that is $\text{Pre}(\cdot)$ is monotone. Intuitively, from a history τ such that $h(\tau) \in \text{Pre}(X)$ the player can choose an action a such that $(h(\tau), a) \in X$, and if $(h(\tau), a) \in \text{Pre}(X)$ then for all moves c chosen by the environment and supported by action a we have $h(\tau c) \in X$. In a game of perfect information, iterating the predecessor operator from the target set P_F until obtaining a fixpoint $X_* = \mu X. \text{Pre}(X) \cup P_F$ gives the (existentially and universally) winning states, that is $P_V = P_\exists = X_*$.

In a game of imperfect information with randomized strategies for the player, given $X_{i+1} = \text{Pre}(X_i)$, from a history τ such that $h(\tau) \in X_{i+1}$, we may consider playing all actions a such that $(h(\tau), a) \in X_i$ uniformly at random. However, an action played in a history τ is also played in every indistinguishable history $\tau' \sim \tau$. Therefore, we need to ensure that for all actions a played in τ (such that $(h(\tau), a) \in X_i$), playing a in τ' does not leave X_* . Hence for $p = h(\tau)$, even if $(p, a) \in X_*$, the action a should not be played from τ if there exists $p' \approx p$ such that $(p', a) \notin X_*$.

In our algorithm, we remove from X_* all elements (p, a) such that $(p', a) \notin X_*$ for some $p' \approx p$, that is we replace X_* by its interior. We define

$$Y_* = \nu Y. \mu X. \text{int}_{\approx}(Y) \cap (\text{Pre}(X) \cup P_F)$$

and we show that the sets Y_* , P_V , and P_\exists coincide. Note that the μ -calculus formula for Y_* is well defined since $\text{Pre}(\cdot)$ and $\text{int}_{\approx}(\cdot)$ are monotone operators, and that the fixpoint can be effectively computed since P is finite.

As the fixpoint satisfies $Y_* = \text{int}_{\approx}(Y_*) \cap (\text{Pre}(Y_*) \cup P_F)$, it follows that $Y_* = \text{int}_{\approx}(Y_*)$ is closed under \approx , and $Y_* \subseteq \text{Pre}(Y_*) \cup P_F$. Moreover it is easy to verify from the fixpoint iteration that $P_F \subseteq Y_*$ since P_F , which is closed under \approx , is always contained in the least fixpoint, and never removed from the greatest fixpoint.

► **Theorem 2.** *The abstract states in the fixpoint Y_* are the existentially winning states, which coincide with the universally winning states, $Y_* \cap P = P_V = P_\exists$.*

Theorem 2 entails the decidability of the existence of an almost-sure winning strategy in games with reachability objective: if there exists an almost-sure winning strategy (from ε), then $h(\varepsilon) \in P_\exists = Y_*$, and conversely if $h(\varepsilon) \in Y_*$ then $h(\varepsilon) \in P_V$ since $h(\varepsilon) \in P$, and there exists an almost-sure winning strategy from $[\varepsilon]_{\sim} = \{\varepsilon\}$.

► **Corollary 3.** *There exists an almost-sure winning strategy if and only if $h(\varepsilon) \in Y_*$.*

The result of Theorem 2 follows from the chain of inclusions $Y_* \cap P \subseteq P_V$ (Lemma 4), the already established $P_V \subseteq P_\exists$, and $P_\exists \subseteq Y_* \cap P$ (Lemma 5),

To show that $Y_* \cap P \subseteq P_V$ we construct a strategy for the player that is almost-sure winning from all τ (and from $[\tau]_{\sim}$) such that $h(\tau) \in Y_*$. The strategy plays uniformly at random all actions that ensure the successor τc of τ remains in Y_* , more precisely $h(\tau c) \in Y_*$. We show that at least one such action ensures progress towards reaching a target state in P_F , thus with probability at least $\frac{1}{|A|}$. The target is reachable in at most $|P|$ steps, which entails bounded probability (at least $\nu = \frac{1}{|A|^{|P|}}$) to reach P_F (from every state of Y_*) and since the strategy ensures that Y_* is never left, the probability of *not* reaching P_F is at most $(1 - \nu)^k$ for all $k \geq 0$, and as $(1 - \nu)^k \rightarrow 0$ when $k \rightarrow \infty$ the probability to eventually reach P_F is 1.

► **Lemma 4.** $Y_* \cap P \subseteq P_V$.

The last inclusion of the chain is proved by showing that there exists a fixpoint that contains P_\exists and since Y_* is defined as the greatest fixpoint, we have $P_\exists \subseteq Y_*$.

► **Lemma 5.** $P_\exists \subseteq Y_* \cap P$.

The fixpoint computation for Y_* can be implemented by a quadratic algorithm, with respect to the number $|P|$ of abstract states (assuming a constant number of actions).

Both stochastic games on graphs and Büchi objectives are subsumed by the results of this paper: in stochastic games, almost-sure Büchi reduces to almost-sure reachability [4, Lemma 8.3], and games with stochastic transitions can be simulated by (non-stochastic) games with imperfect information [11, Theorem 5]. These results can easily be lifted to the more general framework of games with imperfect information defined by indistinguishability relations. On the other hand, we recall that almost-sure coBüchi is undecidable, already in probabilistic automata with pure strategies [4], and also with randomized strategies [11, Corollary 2], but without the assumption that the coloring function is information consistent. For the special case of information-consistent coBüchi objectives, the decidability status of the synthesis problem for almost-sure winning is open (even in games *à la Reif*), to the best of our knowledge. Finally for safety objectives, almost-sure winning is equivalent to sure winning (which requires that all possible outcomes satisfy the objective), and the problem is equivalent to synthesis with pure strategies [7].

► **Theorem 6.** *Given a regular game \mathcal{G} with imperfect information and a rectangular morphism $h : \Gamma^* \rightarrow P$ for \mathcal{G} , the synthesis problem for almost-sure reachability and Büchi objectives can be solved in time $O(|P|^2)$.*

Since $|P|$ is already exponential in games *à la Reif* (for which the synthesis problem is EXPTIME-complete), the quadratic blow-up is not significant. For example, we get a $(k + 1)$ -EXPTIME complexity upper bound for the synthesis problem in the model of full-information protocols (FIP) with k observers, using the rectangular morphism of [7, Section 5.2]. The model of full-information protocols as presented in [7] features explicit communication actions that entail full disclosure of all available information of the sender. The game involves several players who may communicate, but only one active player who takes control actions, which reduces the synthesis problem to a game with imperfect information as considered here.

A matching $(k + 1)$ -EXPTIME lower bound can be obtained by a straightforward adaptation of the proof of [7, Theorem 3], which presents a reduction from the membership problem for alternating k -EXSPACE Turing machines to the synthesis problem of (pure) winning strategies. The reduction is such that in the constructed game, randomization does not help the player. If the player takes a chance in deviating from the faithful simulation of the Turing machine, a losing sink state is reached, thus with positive probability.

► **Theorem 7.** *The synthesis problem for FIP games with k observers (and almost-sure reachability objective) is $(k + 1)$ -EXPTIME-complete.*

The construction of an almost-sure winning strategy α can be done by following the first step in the proof of Lemma 4, which constructs α given the value of the fixpoint Y_* . It follows that the constructed strategy is a regular function that can be represented by the automaton underlying the rectangular morphism of the game, and thus memory of size $|P|$ is sufficient to define an almost-sure winning strategy.

4 Reductions

The results of Section 3 may suggest the existence of a (strong) correspondence, possibly an equivalence, between the game \mathcal{G} with regular indistinguishability relation and a simpler (finite-state) game \mathcal{H} (essentially the abstract game presented in Section 3.1) that would hold for arbitrary objectives.

While it is virtually impossible to establish the non-existence of a reasonably strong such correspondence, we show for a natural notion of game equivalence that such a correspondence does not hold in general.

4.1 Alternating probabilistic trace equivalence

Inspired by the notion of alternating refinement relations [2] in non-stochastic game graphs with perfect information (also called alternating transition systems), we consider the following definition of *game refinement*.

Given two games \mathcal{G} and \mathcal{H} , we say that \mathcal{G} *reduces* to \mathcal{H} (denoted by $\mathcal{G} \preceq \mathcal{H}$) if for all strategies $\alpha_{\mathcal{G}}$ of the player in \mathcal{G} , there exists a strategy $\alpha_{\mathcal{H}}$ of the player in \mathcal{H} , such that for all strategies $\beta_{\mathcal{H}}$ of the environment in \mathcal{H} , there exists a strategy $\beta_{\mathcal{G}}$ of the environment in \mathcal{G} , such that the probability measures $\Pr^{\alpha_{\mathcal{G}}, \beta_{\mathcal{G}}}$ in \mathcal{G} and $\Pr^{\alpha_{\mathcal{H}}, \beta_{\mathcal{H}}}$ in \mathcal{H} coincide (on all objectives $L \subseteq C^\omega$, that is $\Pr^{\alpha_{\mathcal{G}}, \beta_{\mathcal{G}}}(L) = \Pr^{\alpha_{\mathcal{H}}, \beta_{\mathcal{H}}}(L)$ for all L), in short:

$$\forall \alpha_{\mathcal{G}} \cdot \exists \alpha_{\mathcal{H}} \cdot \forall \beta_{\mathcal{H}} \cdot \exists \beta_{\mathcal{G}} \cdot \forall L \subseteq C^\omega : \Pr^{\alpha_{\mathcal{G}}, \beta_{\mathcal{G}}}(L) = \Pr^{\alpha_{\mathcal{H}}, \beta_{\mathcal{H}}}(L).$$

As expected, the quantifications over strategies of the player range over information-consistent randomized strategies (in their respective game), and the quantification over L ranges over measurable sets. Under these quantifications, the condition $\Pr^{\alpha_{\mathcal{G}}, \beta_{\mathcal{G}}}(L) = \Pr^{\alpha_{\mathcal{H}}, \beta_{\mathcal{H}}}(L)$ can be replaced by $\Pr^{\alpha_{\mathcal{G}}, \beta_{\mathcal{G}}}(L) \leq \Pr^{\alpha_{\mathcal{H}}, \beta_{\mathcal{H}}}(L)$ (consider the complement of L , which is also a measurable set), and then game refinement can be interpreted as the game \mathcal{H} being easier² than \mathcal{G} for the player because for every strategy of the player in \mathcal{G} there is a better² strategy for the player in \mathcal{H} (as reflected by the quantifier sequence $\forall \alpha_{\mathcal{G}} \cdot \exists \alpha_{\mathcal{H}}$), which means that the environment can always ensure a lower² value in \mathcal{G} (against $\alpha_{\mathcal{G}}$) than in \mathcal{H} (against $\alpha_{\mathcal{H}}$), as reflected by the quantifier sequence $\forall \beta_{\mathcal{H}} \cdot \exists \beta_{\mathcal{G}}$.

Note that in the special case of pure strategies (and non-stochastic games) where the probability measures assign probability 1 to a single play, game refinement boils down to

$$\forall \alpha_{\mathcal{G}} \in \mathcal{A}_{\mathcal{G}}^{\text{pure}} \cdot \exists \alpha_{\mathcal{H}} \in \mathcal{B}_{\mathcal{H}}^{\text{pure}} : \text{outcome}_{\mathcal{H}}(\alpha_{\mathcal{H}}) \subseteq \text{outcome}_{\mathcal{G}}(\alpha_{\mathcal{G}})$$

where $\text{outcome}_X(\alpha)$ denotes the set of plays compatible with α in game X . The condition $\text{outcome}_{\mathcal{H}}(\alpha_{\mathcal{H}}) \subseteq \text{outcome}_{\mathcal{G}}(\alpha_{\mathcal{G}})$ is equivalent to

$$\forall \beta_{\mathcal{H}} \in \mathcal{B}_{\mathcal{H}}^{\text{pure}} \cdot \exists \beta_{\mathcal{G}} \in \mathcal{B}_{\mathcal{G}}^{\text{pure}} : \text{outcome}_{\mathcal{H}}(\alpha_{\mathcal{H}}, \beta_{\mathcal{H}}) = \text{outcome}_{\mathcal{G}}(\alpha_{\mathcal{G}}, \beta_{\mathcal{G}}),$$

where $\text{outcome}_X(\alpha, \beta)$ denotes the set of plays compatible with α and β in game X . This is essentially the definition of alternating trace containment [2, Section 4]. Hence our definition of game refinement is a natural generalization of both alternating trace containment to a stochastic and imperfect-information setting, and of refinement for labeled Markov decision processes [15, Section 4] to games.

We say that two games \mathcal{G} and \mathcal{H} are *equivalent* (or inter-reducible) if $\mathcal{G} \preceq \mathcal{H}$ and $\mathcal{H} \preceq \mathcal{G}$. We sometimes refer to this equivalence as *alternating probabilistic trace equivalence*.

4.2 Bijection and bisimulation

We compare the approach in Section 3 and the technique for solving almost-sure reachability in games with partial observation *à la Reif* [21]. Recall that games with partial observation are a special case of games with regular indistinguishability relation.

² The words “easier”, “better”, “lower”, etc. are interpreted in a non-strict sense.

23:12 Regular Games with Imperfect Information Are Not That Regular

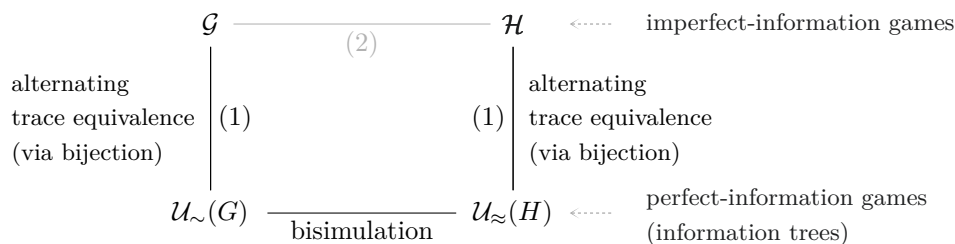
The solution of the synthesis problem with randomized strategies for games *à la Reif* established a bijection (denoted by h , but we call it \hat{h} to emphasize that it is a mapping between sequences) between the histories in the original game \mathcal{G} and in a game \mathcal{H} of partial observation played as follows: given a history ρ (initially ε), the player chooses an action $a \in A$, and the environment extends the history to $\rho' = \hat{h}(\tau c)$ where $\hat{h}(\tau) = \rho$ and c is a move (in \mathcal{G}) such that $\text{act}(c) = a$. Two histories are indistinguishable if all their respective prefixes (of the same length) are \approx -equivalent (according to the definition of \approx before Lemma 1). We call \mathcal{H} the *abstract game induced* from \mathcal{G} by \hat{h} , and by an abuse of notation, we denote by \approx the indistinguishability relation of \mathcal{H} .

The bijection \hat{h} naturally extends to a bijection between plays in \mathcal{G} and plays in \mathcal{H} , and further to a bijection between strategies in \mathcal{G} and strategies in \mathcal{H} that preserves probability measures. The existence of this bijection immediately ensures that the games \mathcal{G} and \mathcal{H} are equivalent (in the sense of alternating probabilistic trace equivalence). Another important consequence is that the information trees of \mathcal{G} and \mathcal{H} are isomorphic, a situation that is not guaranteed when \mathcal{G} is a game with regular indistinguishability relation and \mathcal{H} is a finite-state game (i.e., the set $h(\Gamma^*) = \{\text{last}(\hat{h}(\tau)) \mid \tau \in \Gamma^*\}$ is finite). Indeed, the branching degree of the information tree of \mathcal{G} may be unbounded [6], whereas the branching degree of the information tree of \mathcal{H} is necessarily bounded since \mathcal{H} is a finite game. It is therefore impossible to follow the same route as in games *à la Reif*, using bijections.

Since bijection is too strong for our general setting, a more realistic hope is to rely on the existence of a bisimulation between the information tree of \mathcal{G} and the information tree of \mathcal{H} [7, Theorem 1]. Note that bisimulation is insensitive to the branching degree. The bisimulation ensures that, under pure strategies for the player, the (perfect-information) games played on the information trees are equivalent (in the sense of alternating trace equivalent). The equivalence induced by the bisimulation is such that the transfer of strategies (the construction of $\alpha_{\mathcal{H}}$ given $\alpha_{\mathcal{G}}$) is of the form $\alpha_{\mathcal{H}}(\cdot) = \alpha_{\mathcal{G}}(\mu(\cdot))$ where μ is a mapping between bisimilar histories³ that is sequential, that is the image of a prefix of a history is a prefix of the image of that history. So, for pure strategies the action played by $\alpha_{\mathcal{H}}$ at a history is a copy of the action played by $\alpha_{\mathcal{G}}$ in a bisimilar history in \mathcal{G} . In the case of randomized strategies, the player chooses a distribution over actions, which we may view as attaching probabilities to actions, in a way that could be copied along with the action to transfer randomized strategies (on one hand from \mathcal{G} to \mathcal{H} , and on the other hand from \mathcal{H} to \mathcal{G}) and establish alternating probabilistic trace equivalence (containment in both directions) of the two games.

We recall the strong relationships between games with imperfect information and their induced abstract game (Figure 2). First, there is a natural bijection between an imperfect-information game \mathcal{G} and the perfect-information game played on its information tree $\mathcal{U}_{\sim}(\mathcal{G})$, which is a witness of alternating trace equivalence of the two games [7, Lemma 1]. Second, assuming the existence of a rectangular morphism h for \mathcal{G} , the information trees of \mathcal{G} and of \mathcal{H} (the abstract game induced from \mathcal{G} by h) are bisimilar. The bijections and the bisimulation (Figure 2) entail alternating trace equivalence of the games \mathcal{G} and \mathcal{H} and their information trees, and thus equivalence of the synthesis problems with pure strategies in \mathcal{G} and in \mathcal{H} [7, Lemma 2, Theorem 1].

³ In games played over information trees, the histories are sequences of information sets; we do not denote them by a symbol to avoid having to define them formally.



■ **Figure 2** Equivalences between the game \mathcal{G} , its induced abstract game \mathcal{H} , and their information trees.

4.3 Counterexamples

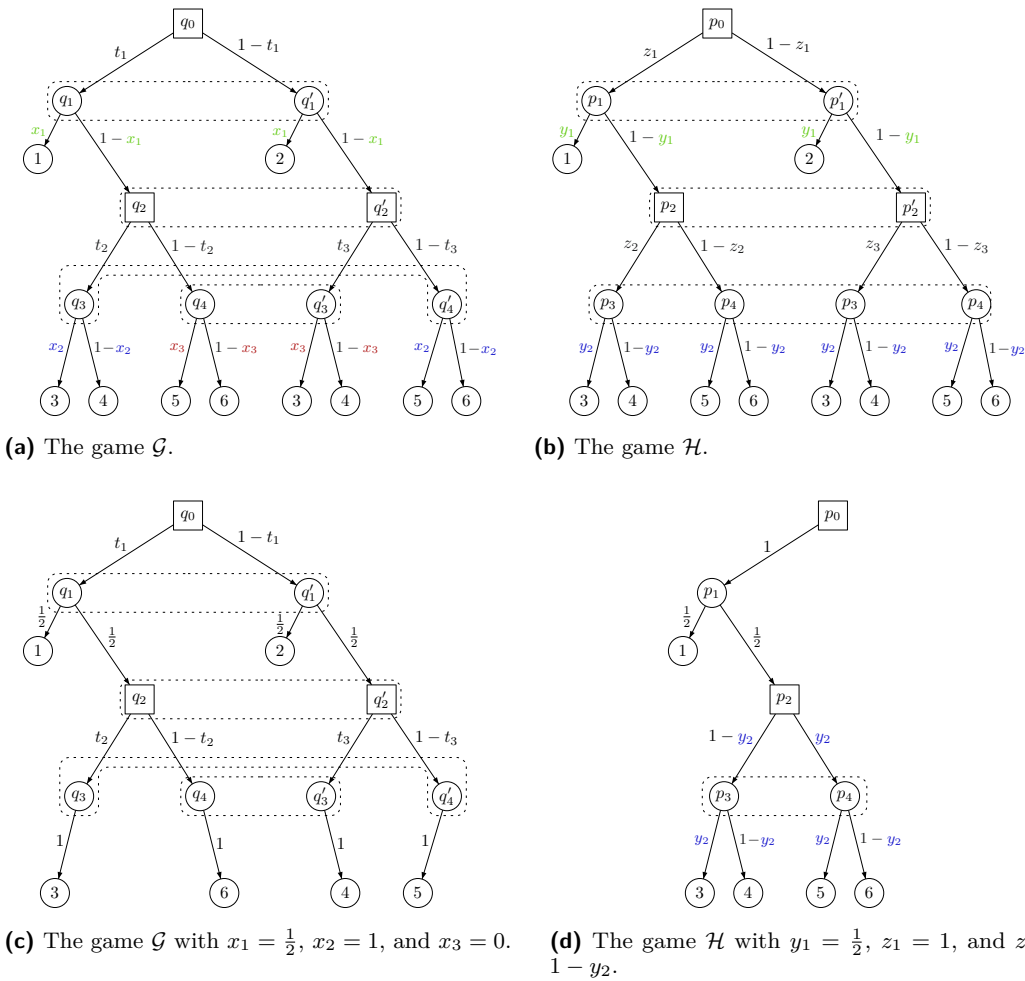
We present two examples showing the difficulty to adapt the results of alternating trace equivalence for games with pure strategies, to alternating probabilistic trace equivalence for games with randomized strategies.

First, we show that with randomized strategies, playing on the information tree of a game with imperfect information is not equivalent to playing in the original game, although this is true for pure strategies (links (1) of Figure 2).

The first example is matching pennies (the game \mathcal{G} in Figure 1a). Recall that the set of actions is $A = \{a, b\}$, the set of moves is $\Gamma = \{a_1, a_2, b_1, b_2\}$ with $\text{act}(a_i) = a$ and $\text{act}(b_i) = b$ for $i = 1, 2$. It is well known that the player cannot win matching pennies using a pure strategy, but wins almost-surely by choosing head and tail uniformly at random in every round. However, in the game played on the information tree of \mathcal{G} (Figure 1b), the player cannot win, even with positive probability. After the first round the information set is $U_2 = \Gamma$, which has two successors on both actions a and b . This is because U_2 contains both a history τ such that τa_i is winning and a history τ' such that $\tau' b_i$ is winning, hence U_3 is a successor of U_2 (on both actions a and b), and analogously X is a successor of U_2 (X is a placeholder for the whole tree of Figure 1b). Therefore, after any choice of action by the player in U_2 , the environment can always choose X and continue the game, ensuring that the reachability objective is never satisfied.

This first example illustrates a crucial difference between pure and randomized strategies. Considering a game \mathcal{G} (with imperfect information induced by an indistinguishability relation \sim) and the game played on its information tree $\mathcal{U}_{\sim}(\mathcal{G})$ (which is of perfect information), there is a natural bijection between (information-consistent) strategies in \mathcal{G} and strategies in $\mathcal{U}_{\sim}(\mathcal{G})$. The bijection exists for both pure and randomized strategies. However, only in the case of pure strategies, the bijection is a witness of alternating trace equivalence (Figure 2). Intuitively, this is because once a *pure* strategy for the player is fixed, the environment “knows” the specific action played in every history (which is the same within an information set), and therefore constructing a path in the original game \mathcal{G} compatible with the player strategy is equivalent to constructing a path in $\mathcal{U}_{\sim}(\mathcal{G})$ (informally, the information tree is a valid abstraction for “existence of paths”). In contrast, once a *randomized* strategy for the player is fixed, the environment only “knows” the specific distribution over actions played in every history, which leaves an element of surprise as to which action will be played when the strategy is executed. In this context, the information tree is no longer a valid abstraction, intuitively because it allows the environment to choose a history within the current information set *after* the randomly chosen action is known, as illustrated by matching pennies (Figure 1b). To be valid, the abstraction should force the environment to stick

23:14 Regular Games with Imperfect Information Are Not That Regular



■ **Figure 3** A game \mathcal{G} and the abstract game \mathcal{H} induced by a rectangular morphism, with randomized strategies encoded by the variables \bar{x}, \bar{t} (in \mathcal{G}) and \bar{y}, \bar{z} (in \mathcal{H}).

to a single choice of a history, before the action is drawn from the distribution chosen by the player. We may consider the abstract game \mathcal{H} induced from \mathcal{G} as a candidate for a richer abstraction, and establish a direct link (2) between \mathcal{G} and \mathcal{H} (Figure 2), given the link (1) breaks. Unfortunately, in our second example \mathcal{H} is not equivalent to \mathcal{G} for alternating probabilistic trace equivalence ($\mathcal{G} \not\sim \mathcal{H}$), showing the absence of such a direct link, which however exists in partial-observation games *à la Reif*, via the bijection \hat{h} mentioned in Section 4.2.

The second example (Figure 3a) shows a game \mathcal{G} and its abstract game \mathcal{H} induced by a rectangular morphism. To avoid tedious description of the game, we present the key features informally. The game \mathcal{G} has two actions for the player and two possible responses for the adversary. The actions and moves are not shown in Figure 3a, and states are depicted as circles when only the action of the player determines the successor, and as boxes when only the move of the environment determines the successor. All states have color 0 except the leaves, labeled by their color $1, 2, \dots, 6$. The leaves are sink states (with a self-loop on every move). The dashed lines show the indistinguishability classes. It is easy to check that the coloring function is information-consistent.

Figure 3b shows the abstract game \mathcal{H} , as a tree with the same shape as in Figure 3a for \mathcal{G} . The value $h(\tau)$ of a history can be read as the label of the node in Figure 3b corresponding to the path defined by τ in Figure 3a. For example, the histories (of length 3) in \mathcal{G} that correspond to the nodes q_3 and q'_3 are mapped by h to the abstract state p_3 . Figure 3b can be viewed as the unraveling of the morphism h . It is easy to verify that h is a rectangular refinement of the coloring function, and that \approx -equivalent abstract states are grouped within the dashed lines.

We show that \mathcal{G} does not reduce to \mathcal{H} according to alternating probabilistic trace containment ($\mathcal{G} \not\leq \mathcal{H}$). We describe all randomized strategies of the player and of the environment by specifying the probability to play an action or a move in each (relevant, i.e., non-leaf) history. We use variables $\bar{x} = x_1, x_2, x_3$ for $\alpha_{\mathcal{G}}$; $\bar{y} = y_1, y_2$ for $\alpha_{\mathcal{H}}$; $\bar{z} = z_1, z_2, z_3$ for $\beta_{\mathcal{H}}$; and $\bar{t} = t_1, t_2, t_3$ for $\beta_{\mathcal{G}}$. Note that the player may only use information-consistent strategies, hence the same variable is used from indistinguishable histories to describe $\alpha_{\mathcal{G}}$ and $\alpha_{\mathcal{H}}$. The claim $\mathcal{G} \not\leq \mathcal{H}$ can be written as

$$\psi \equiv \exists \bar{x} \cdot \forall \bar{y} \cdot \exists \bar{z} \cdot \forall \bar{t} : \bigvee_{c=1}^6 \varphi_c$$

where φ_c expresses that the mass of probability on color c differs in \mathcal{G} and in \mathcal{H} . For example,

$$\varphi_1 \equiv t_1 x_1 \neq z_1 y_1, \text{ and}$$

$$\varphi_3 \equiv t_1 t_2 x_2 (1 - x_1) + t_3 x_3 (1 - t_1) (1 - x_1) \neq z_1 z_2 y_2 (1 - y_1) + z_3 y_2 (1 - z_1) (1 - y_1).$$

To show that ψ holds, we fix $x_1 = \frac{1}{2}$, $x_2 = 1$, and $x_3 = 0$. Now, consider all possible values of \bar{y} and observe that if $y_1 \neq \frac{1}{2}$, then ψ holds since the probability mass in colors $\{1, 2\}$ is $\frac{1}{2}$ in \mathcal{G} and y_1 in \mathcal{H} (and thus $\varphi_1 \vee \varphi_2$ must hold regardless of the value of \bar{z} and \bar{t}). So, it remains to show that ψ holds for $y_1 = \frac{1}{2}$ (and for all values of y_2). Fix $z_1 = 1$ and $z_2 = 1 - y_2$ (the value of z_3 is arbitrary), as illustrated in Figure 3c and Figure 3d.

Consider all possible values of \bar{t} and observe that if $t_1 \neq 1$, then ψ holds since the probability mass in color 1 is $\frac{1}{2}$ in \mathcal{G} and t_1 in \mathcal{H} (thus φ_1 holds). With $t_1 = 1$, we show that $\varphi_4 \vee \varphi_5$ holds, that is:

$$0 \neq \frac{(1 - y_2)^2}{2} \vee 0 \neq \frac{(y_2)^2}{2}$$

which holds since either $y_2 \neq 0$ or $1 - y_2 \neq 0$.

We conclude that ψ holds and thus \mathcal{G} does not reduce to \mathcal{H} ($\mathcal{G} \not\leq \mathcal{H}$). Note that slightly stronger statements can be proved by a similar argument: first, the strategy $\beta_{\mathcal{H}}$ (represented by the variables \bar{z}) can be chosen pure, by setting $z_2 = 1$ if $y_2 = 0$ and $z_2 = 0$ if $y_2 = 1$ (and setting z_2 to an arbitrary value in $\{0, 1\}$ otherwise); second, the example does not even preserve almost-sure probabilities, in the following sense. Given a target set $T \subseteq C = \{1, \dots, 6\}$ of colors, let $\text{Reach}(T) = \bigcup_{k \in T} C^* k C^\omega$ be the reachability objective with target T . Consider the formula

$$\psi' \equiv \exists \bar{x} \cdot \forall \bar{y} \cdot \exists \bar{z} \cdot \forall \bar{t} : \bigvee_{T \subseteq C} \neg(\text{Pr}_{\mathcal{G}}^{\bar{x}, \bar{t}}(\text{Reach}(T)) = 1) \Leftrightarrow \text{Pr}_{\mathcal{H}}^{\bar{y}, \bar{z}}(\text{Reach}(T)) = 1$$

where $\text{Pr}_{\mathcal{G}}^{\bar{x}, \bar{t}}(\text{Reach}(T))$ is the probability that the objective $\text{Reach}(T)$ is satisfied in \mathcal{G} under strategies $\alpha_{\mathcal{G}}$ defined by the variables \bar{x} for the player and $\beta_{\mathcal{G}}$ defined by \bar{t} for the environment (and analogously for $\text{Pr}_{\mathcal{H}}^{\bar{y}, \bar{z}}(\text{Reach}(T))$ in \mathcal{H}). It is easy to check that the condition ψ' entails ψ .

To prove that ψ' holds, we fix $x_1 = \frac{1}{2}$, $x_2 = 1$, and $x_3 = 0$ as before (Figure 3c). Consider three possible cases for \bar{y} : (i) if $y_1 = 0$, then no matter the value of the other variables the objective $\text{Reach}(\{1, 2\})$ has probability $\frac{1}{2}$ in \mathcal{G} and probability 0 in \mathcal{H} , so we take $T = C \setminus \{1, 2\} = \{3, 4, 5, 6\}$; (ii) if $y_1 = 1$, we take $T = \{1, 2\}$ (the probability mass is $\frac{1}{2}$ in \mathcal{G} and 1 in \mathcal{H}); (iii) otherwise $0 < y_1 < 1$, and we take $z_1 = 1$ and $z_2 = 1 - y_2$ (illustrated in Figure 3d for $y_1 = \frac{1}{2}$), and consider all possible values of \bar{t} : if $t_1 \neq 1$ then $\text{Reach}(\{2\})$ has positive probability in \mathcal{G} and probability 0 in \mathcal{H} , so we take $T = C \setminus \{2\} = \{1, 3, 4, 5, 6\}$, and if $t_1 = 1$, then $\text{Reach}(\{4, 5\})$ has probability 0 in \mathcal{G} and positive probability in \mathcal{H} , so we take $T = C \setminus \{4, 5\} = \{1, 2, 3, 6\}$.

► **Theorem 8.** *There exists a regular game \mathcal{G} such that the abstract game \mathcal{H} induced from \mathcal{G} is not alternating probabilistic trace equivalent to \mathcal{G} .*

References

- 1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? *Inf. Comput.*, 282:104651, 2022. doi:10.1016/j.ic.2020.104651.
- 2 Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998. doi:10.1007/BFb0055622.
- 3 Krzysztof R. Apt and Erich Grädel, editors. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011. URL: <http://www.cambridge.org/gb/knowledge/isbn/item5760379>.
- 4 Christel Baier, Marcus Größer, and Nathalie Bertrand. Probabilistic ω -automata. *J. ACM*, 59(1):1, 2012. doi:10.1145/2108242.2108243.
- 5 Chitta Baral, Thomas Bolander, Hans van Ditmarsch, and Sheila A. McIlraith. Epistemic planning (Dagstuhl seminar 17231). *Dagstuhl Reports*, 7(6):1–47, 2017. doi:10.4230/DagRep.7.6.1.
- 6 Dietmar Berwanger and Laurent Doyen. Observation and distinction. representing information in infinite games. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPICs*, pages 48:1–48:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.STACS.2020.48.
- 7 Dietmar Berwanger, Laurent Doyen, and Thomas Soullard. Synthesising full-information protocols. *CoRR*, abs/2307.01063, 2023. doi:10.48550/arXiv.2307.01063.
- 8 Julian C. Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 721–756. North-Holland, 2007. doi:10.1016/s1570-2464(07)80015-2.
- 9 Julian C. Bradfield and Igor Walukiewicz. The mu-calculus and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 871–919. Springer, 2018. doi:10.1007/978-3-319-10575-8_26.
- 10 Pavol Cerný, Krishnendu Chatterjee, Thomas A. Henzinger, Arjun Radhakrishna, and Rohit Singh. Quantitative synthesis for concurrent programs. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2011. doi:10.1007/978-3-642-22110-1_20.
- 11 Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Thomas A. Henzinger. Randomness for free. *Inf. Comput.*, 245:3–16, 2015. doi:10.1016/j.ic.2015.06.003.

- 12 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995. doi:10.1145/210332.210339.
- 13 Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theor. Comput. Sci.*, 386(3):188–217, 2007. doi:10.1016/j.tcs.2007.07.008.
- 14 Stéphane Demri and Raul Fervari. Model-checking for ability-based logics with constrained plans. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 6305–6312. AAAI Press, 2023. doi:10.1609/aaai.v37i5.25776.
- 15 Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Equivalence of labeled markov chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008. doi:10.1142/S0129054108005814.
- 16 Robert Gibbons. *A primer in game theory*. Harvester Wheatsheaf, 1992.
- 17 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.
- 18 Dilian Gurov, Valentin Goranko, and Edvin Lundberg. Knowledge-based strategies for multi-agent teams playing against nature. *Artif. Intell.*, 309:103728, 2022. doi:10.1016/j.artint.2022.103728.
- 19 Donald A. Martin. The determinacy of blackwell games. *J. Symb. Log.*, 63(4):1565–1581, 1998. doi:10.2307/2586667.
- 20 A. Paz. *Introduction to probabilistic automata*. Academic Press, 1971.
- 21 Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Algorithms for omega-regular games with imperfect information. *Log. Methods Comput. Sci.*, 3(3), 2007. doi:10.2168/LMCS-3(3:4)2007.
- 22 John H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984. doi:10.1016/0022-0000(84)90034-5.
- 23 Wolfgang Thomas. On the synthesis of strategies in infinite games. In Ernst W. Mayr and Claude Puech, editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995. doi:10.1007/3-540-59042-0_57.

A Proof of Lemma 4

The abstract states computed by the fixpoint Y_* are all universally winning.

► **Lemma 4.** $Y_* \cap P \subseteq P_\forall$.

Proof. Given $p \in Y_* \cap P$, we show that $p \in P_\forall$. Let $A_p = \{a \in A \mid (p, a) \in Y_*\}$ and show that $A_p \neq \emptyset$. Since $Y_* \subseteq \text{Pre}(Y_*) \cup P_F$, either (i) $p \in P_F$ and then $A_p = A \neq \emptyset$ as $P_F \times A \subseteq \text{Pre}(P_F)$ (states in P_F form a sink set and P_F is closed under \approx); or (ii) $p \in \text{Pre}(Y_*)$ and by definition there exists $a \in A$ such that $(p, a) \in Y_*$, hence $A_p \neq \emptyset$.

Consider the strategy α for the player defined, for all $\tau \in \Gamma^*$, by $\alpha(\tau) = d_U(A_p)$ where $p = h(\tau)$ and $d_U(\cdot)$ is the uniform distribution. We show that α is information-consistent. Given $\tau' \sim \tau$, let $p = h(\tau)$ and $p' = h(\tau')$. Hence we have $p \approx p'$, and since Y_* is closed under \approx , if $(p, a) \in Y_*$ then $(p', a) \in Y_*$. It follows that $A_p = A_{p'}$ and thus $\alpha(\tau) = \alpha(\tau')$.

The strategy α ensures that Y_* is never left: if $h(\tau) \in Y_*$ and $a \in \text{Supp}(\alpha(\tau))$, then $h(\tau c) \in Y_*$ for all c such that $\text{act}(c) = a$. Indeed, for $p = h(\tau)$ we have $a \in A_p$ and $(p, a) \in Y_*$ thus $(p, a) \in \text{Pre}(Y_*)$ which imply $\delta^P(p, c) = h(\tau c) \in Y_*$.

23:18 Regular Games with Imperfect Information Are Not That Regular

We now show that α ensures from Y_* a target state is reached within the next $N = |P|$ steps with probability at least $\nu = \frac{1}{|A|^N}$, that is $\Pr_{\tau}^{\alpha, \beta}(\text{Reach}^{\leq |\tau|+N}) \geq \nu$ for all strategies β of the environment and for all $\tau \in \Gamma^*$ such that $h(\tau) \in Y_*$.

Since $Y_* = \mu X. \text{int}_{\approx}(Y_*) \cap (\text{Pre}(X) \cup P_F)$, the set Y_* is the limit of the (non-decreasing) sequence $(X_i)_{i \in \mathbb{N}}$ where $X_0 = \emptyset$ and $X_{i+1} = Y_* \cap (\text{Pre}(X_i) \cup P_F)$. The *rank* of an element $y \in Y_*$ is the least $i \geq 0$ such that $y \in X_i$ (i.e., such that $y \in X_i$ and $y \notin X_{i-1}$). By extension, the rank of a history τ such that $h(\tau) \in Y_*$ is the rank of $h(\tau)$. Note that states $p \in Y_* \cap P$ have odd rank, and the state-action pairs $(p, a) \in Y_* \cap (P \times A)$ have even rank. The smallest rank is 1 and corresponds to the set P_F of target states. Let N_* be the largest rank, which is bounded by $|Y_*|$.

Intuitively, from any history τ such that $h(\tau) \in Y_*$ the strategy α ensures, against all strategies of the environment, that the history τc after the next round has a lower rank (unless the rank of τ is 1, and thus τ is a winning history) with probability at least $\frac{1}{|A|}$. Let $r > 1$ be the rank of $h(\tau)$, then $h(\tau) \in \text{Pre}(X_{r-1})$ and thus there is an action $a \in A$ such that $(h(\tau), a) \in X_{r-1}$, thus $(h(\tau), a) \in \text{Pre}(X_{r-2})$ and $h(\tau c) \in X_{r-2}$ for all c with $\text{act}(c) = a$. It follows that $a \in A_p$ is played with probability at least $\frac{1}{|A|}$ and the intuitive claim follows.

Combined with the fact that the strategy α ensures Y_* is never left, this claim shows that from all τ such that $h(\tau) \in Y_*$, against all strategies β of the environment, the reachability objective is satisfied (i.e., the rank decreases to 1) within the next N_* rounds with probability at least $\nu = \frac{1}{|A|^{N_*}}$. Formally $\Pr_{\tau}^{\alpha, \beta}(\text{Reach}^{\leq |\tau|+N_*}) \geq \nu$ against all strategies. It follows that the probability of *not* reaching a target state within $k \cdot N_*$ rounds is at most $(1 - \nu)^k$ which tends to 0 as $k \rightarrow \infty$, thus $\Pr_{\tau}^{\alpha, \beta}(\text{Reach}) = 1$ by the squeeze theorem. We conclude that the player is almost-sure winning from $[\tau]_{\sim}$ since Y_* is closed under \approx , which implies that $p = h(\tau) \in P_{\forall}$. \blacktriangleleft

B Proof of Lemma 5

The fixpoint Y_* contains all existentially winning states.

► **Lemma 5.** $P_{\exists} \subseteq Y_* \cap P$.

Proof. As a preliminary, it will be useful to note that if a strategy α_{as} for the player is almost-sure winning from an information set $[\tau]_{\sim}$, then for all actions $a \in \text{Supp}(\alpha_{\text{as}}(\tau))$, for all moves c supported by a , $\text{act}(c) = a$, the strategy α_{as} is almost-sure winning from $[\tau c]_{\sim}$. An equivalent conclusion is that for all actions $a \in \text{Supp}(\alpha_{\text{as}}(\tau))$, the pair $(h(\tau), a)$ is in $\text{int}_{\approx}(\text{Pre}(P_{\exists}))$ (\star).

We now proceed with the proof of the lemma. First, $P_{\exists} \subseteq P$ by definition. To show that $P_{\exists} \subseteq Y_*$, since Y_* is defined as a greatest fixpoint, it is sufficient to show that $Y_{\exists} := P_{\exists} \cup \text{int}_{\approx}(\text{Pre}(P_{\exists}))$ is a fixpoint, that is $Y_{\exists} = \mu X. \text{int}_{\approx}(Y_{\exists}) \cap (\text{Pre}(X) \cup P_F)$. Since P_{\exists} is closed under \approx (i.e., $P_{\exists} = \text{int}_{\approx}(P_{\exists})$) so is Y_{\exists} , and we only need to show that $Y_{\exists} \subseteq \mu X. Y_{\exists} \cap (\text{Pre}(X) \cup P_F) =: X_*$ (the converse inclusion is trivial).

Given $y \in Y_{\exists}$, we show that $y \in X_*$, which concludes the proof. We consider two cases, either $y = p \in P_{\exists}$ or $y = (p, a) \in \text{int}_{\approx}(\text{Pre}(P_{\exists}))$:

(i) if $p \in P_{\exists}$, then there exists $\tau_p \in \Gamma^*$ such that $h(\tau_p) = p$ and the player is almost-sure winning from $[\tau_p]_{\sim}$ using a strategy α_{as} . Assume towards contradiction that $p \notin X_*$.

We construct a (pure) spoiling strategy β for the environment as follows. For all $\tau \in \Gamma^*$ such that $h(\tau) \in P_{\exists}$ and $h(\tau) \notin X_*$ (thus $h(\tau) \notin \text{Pre}(X_*)$), for all actions $a \in \text{Supp}(\alpha_{\text{as}}(\tau))$ played by α_{as} in τ , the pair $(h(\tau), a)$ is not in X_* , and further not in

$\text{Pre}(X_*)$ since $(h(\tau), a) \in \text{int}_{\approx}(\text{Pre}(P_{\exists}))$ by (\star) , thus there exists a move c supported by action a such that $h(\tau c) = \delta^P(h(\tau), c) \notin X_*$. Define $\beta(\tau, a) = c$, and define $\beta(\tau', a)$ arbitrarily for τ' such that $h(\tau') \in X_*$.

We have the following, for all histories $\tau \in \Gamma^*$ and for $a \in \text{Supp}(\alpha_{\text{as}}(\tau))$ and $c = \beta(\tau, a)$:

- if $h(\tau) \in P_{\exists}$, then $h(\tau c) \in P_{\exists}$ (since α_{as} is almost-sure winning),
- if $h(\tau) \in P_{\exists}$ and moreover $h(\tau) \notin X_*$, then $h(\tau c) \notin X_*$,
- $h(\tau_p) \in P_{\exists}$ and $h(\tau_p) \notin X_*$.

It follows by an inductive argument that against α , the constructed strategy β ensures from τ_p that X_* and thus P_F is never reached, hence $\Pr_{\tau_p}^{\alpha_{\text{as}}, \beta}(\text{Reach}) = 0$, which contradicts that α_{as} is almost-sure winning from τ_p . Hence $p \in X_*$.

- (ii) if $(p, a) \in \text{int}_{\approx}(\text{Pre}(P_{\exists}))$, then $\delta^P(p, c) \in P_{\exists}$ for all moves c supported by a . By an argument similar to case (i), we have $\delta^P(p, c) \in X_*$. It follows that $(p, a) \in \text{Pre}(X_*)$ and thus $(p, a) \in X_* = Y_{\exists} \cap (\text{Pre}(X_*) \cup P_F)$. ◀

Validity of Contextual Formulas

Javier Esparza  

Technical University of Munich, Germany

Rubén Rubio  

Universidad Complutense de Madrid, Spain

Abstract

Many well-known logical identities are naturally written as equivalences between *contextual formulas*. A simple example is the Boole-Shannon expansion $c[p] \equiv (p \wedge c[\text{true}]) \vee (\neg p \wedge c[\text{false}])$, where c denotes an arbitrary formula with possibly multiple occurrences of a “hole”, called a *context*, and $c[\varphi]$ denotes the result of “filling” all holes of c with the formula φ . Another example is the unfolding rule $\mu X.c[X] \equiv c[\mu X.c[X]]$ of the modal μ -calculus.

We consider the modal μ -calculus as overarching temporal logic and, as usual, reduce the problem whether $\varphi_1 \equiv \varphi_2$ holds for contextual formulas φ_1, φ_2 to the problem whether $\varphi_1 \leftrightarrow \varphi_2$ is valid. We show that the problem whether a contextual formula of the μ -calculus is valid for all contexts can be reduced to validity of ordinary formulas. Our first result constructs a *canonical context* such that a formula is valid for all contexts iff it is valid for this particular one. However, the ordinary formula is exponential in the nesting-depth of the context variables. In a second result we solve this problem, thus proving that validity of contextual formulas is EXP-complete, as for ordinary equivalences. We also prove that both results hold for CTL and LTL as well. We conclude the paper with some experimental results. In particular, we use our implementation to automatically prove the correctness of a set of six contextual equivalences of LTL recently introduced by Esparza et al. for the normalization of LTL formulas. While Esparza et al. need several pages of manual proof, our tool only needs milliseconds to do the job and to compute counterexamples for incorrect variants of the equivalences.

2012 ACM Subject Classification Theory of computation \rightarrow Modal and temporal logics

Keywords and phrases μ -calculus, temporal logic, contextual rules

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.24

Related Version *Extended Version*: [arXiv:2407.07759](https://arxiv.org/abs/2407.07759) [8]

Supplementary Material *Software (Source Code)*: <https://github.com/ningit/ctxform> [12]

Funding This work was partially supported by the Agencia Estatal de Investigación (AEI) under project PID2019-108528RB-C22.

1 Introduction

Some well-known identities useful for reasoning in different logics can only be easily formulated as *contextual identities*. One example is the Boole-Shannon expansion of propositional logic, which constitutes the foundation of Binary Decision Diagrams and many SAT-solving procedures [1]. It can be formulated as

$$c[p] \equiv (p \wedge c[\text{true}]) \vee (\neg p \wedge c[\text{false}]) \quad (1)$$

where, intuitively, c denotes a Boolean formula with “holes”, called a *context*, and $c[\varphi]$ denotes the result of “filling” every hole of the context c with the formula φ . For example, if $c := ([] \wedge p) \vee (q \rightarrow [])$ and $\varphi := p$, then $c[p] = (p \wedge p) \vee (q \rightarrow p)$. More precisely, c is a *context variable* ranging over contexts, and the equivalence sign \equiv denotes that for all possible assignments of contexts to c the ordinary formulas obtained on both sides of \equiv are equivalent.



© Javier Esparza and Rubén Rubio;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 24; pp. 24:1–24:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

24:2 Validity of Contextual Formulas

$$\begin{aligned}
c[\psi_1 \mathbf{U} \psi_2] \mathbf{W} \varphi &\equiv (\mathbf{GF} \psi_2 \wedge c[\psi_1 \mathbf{W} \psi_2] \mathbf{W} \varphi) \vee c[\psi_1 \mathbf{U} \psi_2] \mathbf{U} (\varphi \vee \mathbf{G} c[\text{false}]) \\
\varphi \mathbf{W} c[\psi_1 \mathbf{U} \psi_2] &\equiv \varphi \mathbf{U} c[\psi_1 \mathbf{U} \psi_2] \vee \mathbf{G} \varphi \\
c[\mathbf{GF} \psi] &\equiv (\mathbf{GF} \psi \wedge c[\text{true}]) \vee c[\text{false}] \\
c[\mathbf{FG} \psi] &\equiv (\mathbf{FG} \psi \wedge c[\text{true}]) \vee c[\text{false}] \\
\mathbf{GF} c[\psi_1 \mathbf{W} \psi_2] &\equiv \mathbf{GF} c[\psi_1 \mathbf{U} \psi_2] \vee (\mathbf{FG} \psi_1 \wedge \mathbf{GF} c[\text{true}]) \\
\mathbf{FG} c[\psi_1 \mathbf{U} \psi_2] &\equiv (\mathbf{GF} \psi_2 \wedge \mathbf{FG} c[\psi_1 \mathbf{W} \psi_2]) \vee \mathbf{FG} c[\text{false}]
\end{aligned}$$

■ **Figure 1** Rewrite system for the normalization of LTL formulas [9].

For linear-time temporal logic in negation normal form, a useful identity similar to the Boole-Shannon expansion is

$$c[\mathbf{GF} p] \equiv (\mathbf{GF} p \wedge c[\text{true}]) \vee c[\text{false}] \quad (2)$$

where c now ranges over formulas of LTL with holes. For example, the identity shows that $q \mathbf{U} (\mathbf{GF} p \wedge r)$ is equivalent to $\mathbf{GF} p \wedge q \mathbf{U} r \vee q \mathbf{U} \text{false}$, and so after simplifying equivalent to $\mathbf{GF} p \wedge q \mathbf{U} r$.¹ As a third example, the unfolding rule of the μ -calculus (the fundamental rule in Kozen's axiomatization of the logic [2])

$$\mu X. c[X] \equiv c[\mu X. c[X]]$$

whose formulation requires nested contexts. Further examples of contextual LTL identities are found in [9], where, together with Salomon Sickert, we propose a rewrite system to transform arbitrary LTL formulas into formulas of the syntactic fragment Δ_2 , with at most single-exponential blowup.² The rewrite system consists of the six identities (oriented from left to right) shown in Figure 1.

Remarkably, to the best of our knowledge the *automatic* verification of contextual equivalences like the ones above has not been studied yet. In particular, we do not know of any automatic verification procedure for any of the identities above. In [9] we had to prove manually that the left and right sides of each identity are equivalent for every context (Lemmas 5.7, 5.9, and 5.11 of [9]), a tedious and laborious task; for example, the proof of the first identity alone takes about $3/4$ of a page. This stands in sharp contrast to non-contextual equivalences, where an ordinary equivalence $\varphi_1 \equiv \varphi_2$ of LTL can be automatically verified by constructing a Büchi automaton for the formula $\neg(\varphi_1 \leftrightarrow \varphi_2)$ and checking its emptiness. So the question arises whether the equivalence problem for contextual formulas is decidable, and in particular whether the manual proofs of [9] can be replaced by an automated procedure. In this paper we give an affirmative answer.

Let σ be a mapping assigning contexts to all context variables of a formula φ , and let $\sigma(\varphi)$ denote the ordinary formula obtained by instantiating φ with σ . The equivalence, validity, and satisfiability problems for contextual formulas are:

¹ The restriction to formulas in negation normal form is necessary. For example, taking $c := \neg(p \wedge [\])$ does not yield a valid equivalence.

² Δ_2 contains the formulas in negation normal form such that every path of the syntax tree exhibits at most one alternation of the strong and weak until operators \mathbf{U} and \mathbf{W} . They have different uses, and in particular they are easier to translate into deterministic ω -automata [9].

1. *Equivalence* of φ_1 and φ_2 : does $\sigma(\varphi_1) \equiv \sigma(\varphi_2)$ hold for every σ ?
2. *Validity* of φ : is $\sigma(\varphi)$ valid (in the ordinary sense) for every σ ?
3. *Satisfiability* of φ : is $\sigma(\varphi)$ satisfiable (in the ordinary sense) for some σ ?

We choose the modal μ -calculus as overarching logic, and prove that these problems can be reduced to their counterparts for ordinary μ -calculus formulas. As corollaries, we also derive reductions for CTL and LTL. More precisely, we obtain the following two results.

First result. Given a contextual formula φ with possibly multiple occurrences of a context variable c , there exists a *canonical instantiation* κ_φ of c , also called the *canonical context*, such that φ is valid/satisfiable iff the ordinary formula $\kappa_\varphi(\varphi)$ is valid/satisfiable. Further, κ_φ can be easily computed from φ by means of a syntax-guided procedure.

To give a flavour of the idea behind the canonical instantiation consider the distributive law $\varphi_1 \wedge (\varphi_2 \vee \varphi_3) \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3)$ for ordinary Boolean formulas. It is well-known that such a law is correct iff it is correct for the special case in which $\varphi_1, \varphi_2, \varphi_3$ are distinct Boolean variables, say p_1, p_2, p_3 . In other words, the law is correct iff the Boolean formula $p_1 \wedge (p_2 \vee p_3) \leftrightarrow (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$ is valid. This result does not extend to contextual formulas. For example, consider the contextual equivalence (2), reformulated as the validity of the contextual formula

$$\varphi := c[\mathbf{GF} p] \leftrightarrow ((\mathbf{GF} p \wedge c[\text{true}]) \vee c[\text{false}]) \quad (3)$$

While φ is valid, the ordinary formula $\varphi^d := p_1 \leftrightarrow ((\mathbf{GF} p \wedge p_2) \vee p_3)$ obtained by replacing $c[\mathbf{GF} q]$, $c[\text{true}]$, and $c[\text{false}]$ by atomic propositions p_1, p_2, p_3 , respectively, is not. (We call φ^d the *decontextualization* of φ .) Loosely speaking, the replacement erases dependencies between $c[\mathbf{GF} q]$, $c[\text{true}]$, and $c[\text{false}]$. For example, since contexts are formulas in negation normal form, $c[\text{false}] \models c[\text{true}]$ or $c[\text{false}] \models c[\mathbf{GF} p]$ hold for every context c , but we do not have $p_3 \models p_2$ and $p_2 \models p_1$. To remedy this, we choose a context κ_φ that informally states:

At every moment in time, p_1 holds if the hole is filled with a formula globally entailing $\mathbf{GF} p$ and p_2 holds if it is filled with a formula globally entailing true and p_3 holds if it is filled with a formula globally entailing false.

The context is:

$$\kappa_\varphi := \mathbf{G} \left((\mathbf{G} ([] \rightarrow \mathbf{GF} p) \rightarrow p_1) \wedge (\mathbf{G} ([] \rightarrow \text{true}) \rightarrow p_2) \wedge (\mathbf{G} ([] \rightarrow \text{false}) \rightarrow p_3) \right) \quad (4)$$

Our result shows that κ_φ is a canonical context for φ . In other words, the contextual formula φ of (3) is valid iff the ordinary formula

$$\kappa_\varphi(\varphi) := \kappa_\varphi[\mathbf{GF} p] \leftrightarrow ((\mathbf{GF} p \wedge \kappa_\varphi[\text{true}]) \vee \kappa_\varphi[\text{false}])$$

obtained by setting $c := \kappa_\varphi$ in (3), is valid. After substituting according to (4) and simplifying, we obtain

$$\kappa_\varphi(\varphi) \equiv (\mathbf{G} (p_1 \wedge p_2) \wedge \mathbf{G} (\mathbf{GF} p \vee p_3)) \leftrightarrow (\mathbf{GF} p \wedge \mathbf{G} ((\mathbf{GF} p \rightarrow p_1) \wedge p_2) \vee \mathbf{G} (p_1 \wedge p_2 \wedge p_3)) \quad (5)$$

So (3) is valid iff the formula on the right-hand-side of (5) is valid, which is proved by SPOT 2.11 [4] in milliseconds.

Second result. Given a contextual formula φ , the ordinary formula $\kappa_\varphi(\varphi)$ has $O(|\varphi|^d)$ length, where d is the nesting depth of the context variables. Since $d \in O(n)$, the blowup is exponential. Our second result provides a polynomial reduction. Let $c[\psi_1], \dots, c[\psi_n]$ be the context expressions appearing in φ . Instead of finding a canonical instantiation, we focus on adding to the decontextualized formula φ^d information on the dependencies between $c[\psi_1], \dots, c[\psi_n]$. For every pair ψ_i, ψ_j , we add to φ^d the premise $\mathbf{G}(\mathbf{G}(\psi_i \rightarrow \psi_j) \rightarrow (p_i \rightarrow p_j))$. Intuitively, the premise “transforms” dependencies between ψ_i and ψ_j into dependencies between fresh atomic propositions p_i and p_j . For example, we obtain that (3) is valid iff the ordinary LTL formula

$$\mathbf{G} \left(\bigwedge_{i=1}^3 \bigwedge_{j=1}^3 (\mathbf{G}(\psi_i \rightarrow \psi_j) \rightarrow (p_i \rightarrow p_j)) \right) \rightarrow (p_1 \leftrightarrow ((\mathbf{GF} p \wedge p_2) \vee p_3))$$

is valid or, after simplification, iff

$$\mathbf{G} \left(\begin{array}{c} (\mathbf{FG} \neg p \rightarrow (p_1 \rightarrow p_3)) \wedge (\mathbf{GF} p \rightarrow (p_2 \rightarrow p_1)) \\ \wedge \\ (p_3 \rightarrow p_1) \wedge (p_1 \rightarrow p_2) \end{array} \right) \rightarrow (p_1 \leftrightarrow ((\mathbf{GF} p \wedge p_2) \vee p_3)) \quad (6)$$

is valid. Again, SPOT 2.11 proves that (6) is valid within milliseconds. Since the premise has polynomial size in the size of the original contextual formula, we obtain a reduction from contextual validity to ordinary validity with polynomial blowup. Observe, however, that the ordinary formula is not obtained by directly instantiating the context variable c .

Experiments. We have implemented our reductions and connected them to validity and satisfiability checkers for propositional logic (PySAT [10] and MiniSat [5]), LTL (SPOT 2.11 [4]), and CTL (CTL-SAT [11]). We provide some experimental results. In particular, we can prove the correctness of all the LTL identities of [9] within milliseconds.

Structure of the paper. Section 2 recalls the standard μ -calculus and presents its contextual extension. Section 3 studies the validity problem of contextual propositional formulas, as an appetizer for the main results on the μ -calculus in Section 4. These are extended to CTL and LTL in Sections 4.4 and 4.5. Experimental results are presented in Section 5, and Section 6 gives some conclusions.

2 The contextual μ -calculus

We briefly recall the syntax and semantics of the μ -calculus [2], and then introduce the syntax and semantics of the contextual μ -calculus.

The modal μ -calculus. The syntax of the modal μ -calculus over a set AP of atomic propositions and a set V of variables is

$$\varphi ::= p \mid \neg p \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle \cdot \rangle \varphi \mid [\cdot] \varphi \mid \mu X. \varphi \mid \nu X. \varphi \quad (7)$$

where $p \in AP$ and $X \in V$. The semantics is defined with respect to a Kripke structure and a valuation. A *Kripke structure* is a tuple $\mathcal{K} = (S, \rightarrow, I, AP, \ell)$, where S and I are sets of states and initial states, $\rightarrow \subseteq S \times S$ is the transition relation (where we assume that every state has at least one successor), and $\ell: S \rightarrow \mathcal{P}(S)$ assigns to each state a set of atomic

propositions. A *valuation* is a mapping $\eta: V \rightarrow \mathcal{P}(S)$. Given \mathcal{K} and η , the semantics assigns to each formula φ a set $\llbracket \varphi \rrbracket_\eta \subseteq S$, the set of states satisfying φ . A Kripke structure \mathcal{K} satisfies φ if $I \subseteq \llbracket \varphi \rrbracket_\eta$. The mapping $\llbracket \cdot \rrbracket_\eta$ is inductively defined by:

$$\begin{aligned} \llbracket p \rrbracket_\eta &= \{s \in S \mid p \in \ell(s)\} & \llbracket \langle \cdot \rangle \varphi \rrbracket_\eta &= \{s \in S \mid \exists s'. s \rightarrow s' \wedge s' \in \llbracket \varphi \rrbracket_\eta\} \\ \llbracket \neg p \rrbracket_\eta &= S \setminus \llbracket p \rrbracket_\eta & \llbracket [\cdot] \varphi \rrbracket_\eta &= \{s \in S \mid \forall s'. s \rightarrow s' \Rightarrow s' \in \llbracket \varphi \rrbracket_\eta\} \\ \llbracket X \rrbracket_\eta &= \eta(X) & \llbracket \mu X. \varphi \rrbracket_\eta &= \bigcap \{U \subseteq S \mid \llbracket \varphi \rrbracket_{\eta[X/U]} \subseteq U\} \\ \llbracket \varphi_1 \wedge \varphi_2 \rrbracket_\eta &= \llbracket \varphi_1 \rrbracket_\eta \cap \llbracket \varphi_2 \rrbracket_\eta & \llbracket \nu X. \varphi \rrbracket_\eta &= \bigcup \{U \subseteq S \mid U \subseteq \llbracket \varphi \rrbracket_{\eta[X/U]}\} \\ \llbracket \varphi_1 \vee \varphi_2 \rrbracket_\eta &= \llbracket \varphi_1 \rrbracket_\eta \cup \llbracket \varphi_2 \rrbracket_\eta \end{aligned}$$

Let $\text{FV}(\varphi) \subseteq V$ be the set of free variables, i.e. not bound by a fixpoint operator, in a formula φ . Observe that if φ is a closed formula (that is, $\text{FV}(\varphi) = \emptyset$), then $\llbracket \varphi \rrbracket_\eta$ depends only on \mathcal{K} , not on η , so we just write $\llbracket \varphi \rrbracket$. On the contrary, when dealing with multiple Kripke structures at the same time, we write $\llbracket \varphi \rrbracket_{\mathcal{K}, \eta}$ or $\llbracket \varphi \rrbracket_{\mathcal{K}}$ to avoid ambiguity. We say that \mathcal{K} satisfies φ , denoted $\mathcal{K} \models \varphi$, if $I \subseteq \llbracket \varphi \rrbracket_\eta$, that is, if every initial state satisfies φ . A closed formula φ is valid (satisfiable) if $\mathcal{K} \models \varphi$ for every (some) Kripke structure \mathcal{K} .

It is well-known that every formula of the μ -calculus is equivalent to a formula in which all occurrences of a variable are either bound or free, and every two distinct fixpoint subformulas have different variables.

The contextual modal μ -calculus. *Contextual formulas* are expressions over a set AP of atomic propositions, a set V of variables, and a set C of *context variables*. (The contextual formulas of the introduction only had one contextual variable, but in general they can have multiple and arbitrarily nested variables.) They are obtained by extending the syntax (7) with a new term:

$$\varphi ::= p \mid \neg p \mid X \mid \dots \mid \nu X. \varphi \mid c[\varphi]$$

where $c \in C$. For the semantics, we need to introduce contexts and their instantiations. A *context* is an expression over the syntax that extends (7) with *holes*:

$$\varphi ::= p \mid \neg p \mid X \mid \dots \mid \nu X. \varphi \mid [\]$$

We let \mathcal{C} denote the set of all contexts. An *instantiation* of the set C of context variables is a mapping $\sigma: C \rightarrow \mathcal{C}$. Given a contextual formula φ , we let $\sigma(\varphi)$ denote the ordinary formula obtained as follows: in the syntax tree of φ , proceeding bottom-up, repeatedly replace each expression $c[\psi]$ by the result of filling all holes of the context $\sigma(c)$ with ψ . Here is a formal inductive definition:

► **Definition 1** (instantiation). *Let \mathbb{F} and \mathbb{C} be the sets of ordinary and contextual formulas of the μ -calculus. An instantiation is a function $\sigma: C \rightarrow \mathcal{C}$ binding each context variable to a context. We lift an instantiation σ to a mapping $\bar{\sigma}_c: \mathbb{C} \rightarrow \mathbb{F}$ as follows:*

1. $\bar{\sigma}_c(p) = p$.
2. $\bar{\sigma}_c(c[\varphi]) = (\sigma(c))[[\] / \bar{\sigma}_c(\varphi)]$ (i.e., the result of substituting $\bar{\sigma}_c(\varphi)$ for $[\]$ in $\sigma(c)$).
3. $\bar{\sigma}_c(\varphi_1 \wedge \varphi_2) = \bar{\sigma}_c(\varphi_1) \wedge \bar{\sigma}_c(\varphi_2)$.
4. $\bar{\sigma}_c(\varphi_1 \vee \varphi_2) = \bar{\sigma}_c(\varphi_1) \vee \bar{\sigma}_c(\varphi_2)$.
5. $\bar{\sigma}_c(\langle \cdot \rangle \varphi) = \langle \cdot \rangle \bar{\sigma}_c(\varphi)$.
6. $\bar{\sigma}_c([\cdot] \varphi) = [\cdot] \bar{\sigma}_c(\varphi)$.
7. $\bar{\sigma}_c(\mu X. \varphi) = \mu X. \bar{\sigma}_c(\varphi)$.
8. $\bar{\sigma}_c(\nu X. \varphi) = \nu X. \bar{\sigma}_c(\varphi)$.

Abusing language, we overload σ and write $\sigma(\varphi)$ for $\bar{\sigma}(\varphi)$.

24:6 Validity of Contextual Formulas

► **Example 2.** Let $\varphi = p \mathbf{U} c_1 [(p \vee c_1[q]) \mathbf{W} c_2 [\neg p \vee q]]$. Further, let $\sigma(c_1) := \mathbf{G} []$ and $\sigma(c_2) := ([] \wedge q)$. We have $\sigma(\varphi) = p \mathbf{U} \mathbf{G} ((p \vee \mathbf{G} q) \mathbf{W} ((\neg p \vee q) \wedge q))$.

We can now extend the notions of validity and satisfaction from ordinary to contextual formulas.

► **Definition 3** (validity and satisfiability). *A closed contextual formula φ is valid if $\mathcal{K} \models \sigma(\varphi)$ for every instantiation σ and Kripke structure \mathcal{K} , and satisfiable if $\mathcal{K} \models \sigma(\varphi)$ for some instantiation σ and Kripke structure \mathcal{K} .*

3 Validity of contextual propositional formulas

As an appetizer, we study the validity and satisfiability problems for the propositional fragment of the modal μ -calculus, which allows us to introduce the main ideas in the simplest possible framework. The syntax of contextual propositional formulas over sets AP and C of propositional and contextual variables is

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid c[\varphi] \quad (8)$$

where $p \in AP$ and $c \in C$. The semantics is induced by the semantics of the modal μ -calculus, but we quickly recall it. Given a *valuation* $\beta: AP \rightarrow \{0, 1\}$, the semantics of an ordinary formula φ is the Boolean $\llbracket \varphi \rrbracket_\beta \in \{0, 1\}$, defined as usual, e.g. $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket_\beta = 1$ iff $\llbracket \varphi_1 \rrbracket_\beta = 1$ and $\llbracket \varphi_2 \rrbracket_\beta = 1$. Given a valuation β and an instantiation $\sigma: C \rightarrow \mathcal{C}$ of the context variables, the semantics of a contextual formula is the boolean $\llbracket \sigma(\varphi) \rrbracket_\beta$, where $\sigma(\varphi)$ is the formula obtained by instantiating each context variable c with the context $\sigma(c)$.

We will use the substitution lemma of propositional logic. Let \mathbb{F} and \mathbb{C} be the set of all ordinary and contextual propositional formulas, respectively.

► **Lemma 4** (substitution lemma). *For any $\varphi \in \mathbb{F}$, valuation $\beta: AP \rightarrow \{0, 1\}$, and substitution $\sigma: AP \rightarrow \mathbb{F}$, $\llbracket \sigma(\varphi) \rrbracket_\beta = \llbracket \varphi \rrbracket_{\beta'}$ where β' is given by $\beta'(p) := \llbracket \sigma(p) \rrbracket_\beta$ for every $p \in V$.*

Moreover, since formulas with syntax (8) are in negation normal form, we have the following monotonicity result.

► **Lemma 5** (monotonicity). *For any $\varphi, \psi, \psi' \in \mathbb{F}$, propositional variable p that does not appear negated in φ , and valuation $\beta: AP \rightarrow \{0, 1\}$, if $\llbracket \psi \rightarrow \psi' \rrbracket_\beta = 1$ then $\llbracket \varphi[p/\psi] \rightarrow \varphi[p/\psi'] \rrbracket_\beta = 1$.*

3.1 Canonical instantiations

Let us now prove that a contextual propositional formula $\varphi \in \mathbb{C}$ is valid (satisfiable) iff it is valid (satisfiable) for the following *canonical instantiation* κ_φ of its context variables.

► **Definition 6** (maximal context subformulas). *A context subformula of $\varphi \in \mathbb{C}$ is a subformula of φ of the form $c[\psi]$ for some $c \in C$ and $\psi \in \mathbb{C}$. The set of context subformulas of φ is denoted $CSub(\varphi)$. A context subformula is maximal if it is not a proper subformula of any other context subformula. The decontextualization of φ , denoted φ^d , is the result of replacing every maximal context subformula $c[\psi]$ of φ by a fresh propositional variable $p_{c[\psi]}$.*

► **Definition 7** (canonical instantiation of a contextual formula). *The canonical instantiation of $\varphi \in \mathbb{C}$, also called the canonical context, is the mapping $\kappa_\varphi: C \rightarrow \mathcal{C}$ that assigns to every context variable $c \in C$ the context*

$$\kappa_\varphi(c) := \bigwedge_{c[\psi] \in CSub(\varphi)} ([] \rightarrow \psi^d) \rightarrow p_{c[\psi]}$$

► **Example 8.** Let us illustrate Definition 7 on an example. Boole-Shannon’s expansion holds iff the contextual formula

$$\varphi := c[p] \leftrightarrow ((p \wedge c[\text{true}]) \vee (\neg p \wedge c[\text{false}]))$$

is valid. We have $CSub(\varphi) = \{c[p], c[\text{true}], c[\text{false}]\}$. All elements of $CSub(\varphi)$ are maximal. Since $p^d = p$, $\text{true}^d = \text{true}$, and $\text{false}^d = \text{false}$, the canonical instantiation is given by

$$\kappa_\varphi(c) = (([] \rightarrow p) \rightarrow p_{c[p]}) \wedge (([] \rightarrow \text{true}) \rightarrow p_{c[\text{true}]}) \wedge (([] \rightarrow \text{false}) \rightarrow p_{c[\text{false}]})$$

We need an auxiliary lemma.

► **Lemma 9.** *For any $\varphi \in \mathbb{C}$, instantiation σ , and valuation β , there is a valuation β' that coincides with β in every variable occurring in $\sigma(\varphi)$ and satisfies $\llbracket \sigma(\varphi) \rrbracket_\beta = \llbracket \kappa_\varphi(\varphi) \rrbracket_{\beta'}$.*

Proof sketch (full proof in [8]). Given φ , σ , and β , we define $\beta'(p_{c[\psi]}) = \llbracket \sigma(c[\psi]) \rrbracket_\beta$ for every $c[\psi] \in CSub(\varphi)$ and $\beta'(p) = \beta(p)$ otherwise. We first prove $\llbracket \sigma(\phi) \rrbracket_\beta = \llbracket \phi^d \rrbracket_{\beta'}$ as a direct application of the substitution lemma with $\gamma_\sigma(p_{c[\psi]}) = \sigma(c[\psi])$, which satisfies $\gamma_\sigma(\phi^d) = \sigma(\phi)$. Then, we prove $\llbracket \sigma(\phi) \rrbracket_\beta = \llbracket \kappa_\varphi(\phi) \rrbracket_{\beta'}$ by induction on ϕ , using the previous statement and some calculations on the expression of $\kappa_\varphi(c[\psi])$ using the monotonicity of contexts by Lemma 5. ◀

► **Proposition 10 (fundamental property of the canonical instantiation).** *A contextual formula $\varphi \in \mathbb{C}$ is valid (resp. satisfiable) iff the ordinary formula $\kappa_\varphi(\varphi) \in \mathbb{F}$ is valid (resp. satisfiable).*

Proof. For validity, if φ is valid, then $\sigma(\varphi)$ is valid for every instantiation σ , and so in particular $\kappa_\varphi(\varphi)$ is valid. For the other direction, assume $\kappa_\varphi(\varphi)$ is valid. We prove that $\sigma(\varphi)$ is also valid for any instantiation σ . Let β be a valuation. By Lemma 9 there is another valuation β' such that $\llbracket \sigma(\varphi) \rrbracket_\beta = \llbracket \kappa_\varphi(\varphi) \rrbracket_{\beta'}$. Moreover, we have $\llbracket \kappa_\varphi(\varphi) \rrbracket_{\beta'} = 1$ because $\kappa_\varphi(\varphi)$ is valid. So $\sigma(\varphi)$ is valid, because β is arbitrary.

Satisfiability is handled by a dual proof. If $\kappa_\varphi(\varphi)$ is satisfiable, then so is φ by definition. If φ is satisfiable, then there is an instantiation σ such that $\llbracket \sigma(\varphi) \rrbracket_\beta = 1$. Lemma 9 give us a valuation β' such that $\llbracket \kappa_\varphi(\varphi) \rrbracket_{\beta'} = \llbracket \sigma(\varphi) \rrbracket_\beta = 1$. ◀

► **Example 11.** Let φ and $\kappa_\varphi(c)$ be as in Example 8. By definition, we have $\kappa_\varphi(\varphi) := \kappa_\varphi(c[p]) \leftrightarrow ((p \wedge \kappa_\varphi(c[\text{true}])) \vee (\neg p \wedge \kappa_\varphi(c[\text{false}])))$. Simplification yields

$$\begin{aligned} \kappa_\varphi(\varphi) &\equiv && p_{c[p]} \wedge p_{c[\text{true}]} \wedge (\neg p \rightarrow p_{c[\text{false}]}) \\ &\leftrightarrow && (p \wedge (p \rightarrow p_{c[p]}) \wedge p_{c[\text{true}]}) \vee (\neg p \wedge p_{c[p]} \wedge p_{c[\text{true}]} \wedge p_{c[\text{false}]}) \end{aligned}$$

This formula is not valid, and so by Proposition 10 Boole-Shannon’s expansion is valid.

The following example shows that the ordinary formula $\kappa_\varphi(\varphi)$ may be exponentially larger than the contextual formula φ when φ contains nested contexts.

► **Example 12.** Consider the contextual formula $\varphi := c^n[q]$, where $c^0[\psi] := \psi$ and $c^n[\psi] := c[c^{n-1}[\psi]]$ for every formula ψ . The size of φ is $n + 4$. The canonical context is $\kappa_\varphi(c) = \bigwedge_{i=1}^n ([] \rightarrow p_{c^{i-1}[q]} \rightarrow p_{c^i[q]})$ with $p_{c^0[q]} = q$. Instantiating the “holes” of $\kappa_\varphi(c)$ with a formula ψ of size k yields the formula $\kappa_\varphi(c)[[]/\psi]$ of size $n(7+k) - 1 \geq nk$. Since $\kappa_\varphi(\varphi) = \kappa_\varphi(c^n[q]) = \kappa_\varphi(c)[[]/\kappa_\varphi(c^{n-1}[q])]$ by definition, the size of $\kappa_\varphi(\varphi)$ is at least $n! = (|\varphi| - 4)!$, and so exponential in the size of φ .

3.2 A polynomial reduction

As anticipated in the introduction, in order to avoid the exponential blowup illustrated by the previous example, we consider a second method that relies on finding an ordinary formula equivalent to the contextual formula. This will lead us to the complexity result of Corollary 14.

► **Proposition 13.** *A propositional contextual formula $\varphi \in \mathbb{C}$ is valid iff the ordinary propositional formula*

$$\varphi_e := \left(\bigwedge_{c[\psi_1], c[\psi_2] \in CSub(\varphi)} (\psi_1^d \rightarrow \psi_2^d) \rightarrow (p_{c[\psi_1]} \rightarrow p_{c[\psi_2]}) \right) \rightarrow \varphi^d$$

is valid.

Proof sketch (full proof in [8]). We follow here the same ideas of Section 3.1. (\Rightarrow) If φ_e is valid, for a given substitution σ and Kripke structure \mathcal{K} , we define the Kripke structure \mathcal{K}' of Lemma 9. After showing again that $\llbracket \phi^d \rrbracket_{\beta'} = \llbracket \sigma(\phi) \rrbracket_{\beta}$ for every subformula, we see that the condition of φ_e holds through a calculation, and then its conclusion yields $\llbracket \varphi^d \rrbracket_{\beta'} = \llbracket \sigma(\varphi) \rrbracket_{\beta} = 1$, so φ is valid. (\Leftarrow) If φ is valid, so is $\kappa_{\varphi}(\varphi)$ with a valuation β . We show φ_e holds under the same valuation. This is immediate if the premise does not hold. Otherwise, we can use the monotonicity encoded in the premise of φ_e to almost repeat the calculation on $\kappa_{\varphi}(c[\psi])$ in Lemma 9 and conclude $\llbracket \kappa_{\varphi}(\varphi) \rrbracket_{\beta} = \llbracket \varphi^d \rrbracket_{\beta'} = 1$. ◀

► **Corollary 14.** *The validity and satisfiability problems for contextual propositional formulas are co-NP-complete and NP-complete, respectively.*

Proof. Proposition 13 gives a polynomial reduction to validity of ordinary formulas. Indeed, $|\varphi^d| \leq |\varphi| + |\varphi|^3 \cdot (2|\varphi| + 5) \leq 8|\varphi|^4$. For satisfiability, it suffices to replace the top implication of φ_e by a conjunction. ◀

► **Remark 15.** In the propositional calculus, once we assign truth values to the atomic propositions every formula is equivalent to either true or false. Similarly, every context is equivalent to true, false, or $[\]$. Hence, an alternative method to check validity of a contextual propositional formula is to check the validity of all possible instantiations of the context variables with these three contexts. However, for n different context variables, this requires 3^n validity checks.

► **Remark 16.** Other examples of valid identities are $c[p \wedge q] \equiv c[p] \wedge c[q]$, $c[p \vee q] \equiv c[p] \vee c[q]$, and $c[p] \equiv c[c[p]]$. Example of valid entailments are $(p \leftrightarrow q) \models (c[p] \leftrightarrow c[q])$ and $(p \rightarrow q) \models (c[p] \rightarrow c[q])$; the entailments in the other direction are not valid, as witnessed by the instantiation $\sigma(c) := \text{false}$. Finally, $p \equiv c[p]$ is an example of an identity that is not valid in any direction. All these facts can be automatically checked using any of the methods described in the section.

4 Validity of contextual μ -calculus formulas

We extend the reductions of Section 3 to the contextual modal μ -calculus. In particular, this requires introducing a new definition of canonical instantiation and a new equivalent formula. The main difference with the propositional case is that contexts may now contain free variables (that is, variables that are bound outside the context). For example, in the unfolding rule we find the context $c[X]$, and X appears free in the argument of c . This

problem will be solved by replacing each free variable X by either the fixpoint subformula that binds it, or by a fresh atomic proposition p_X . We will also need to tweak decontextualizations. More precisely, the canonical instantiation will have the shape

$$\kappa_\varphi(c) := \bigwedge_{c[\psi] \in CSub(\varphi)} (\mathbf{AG}([\] \rightarrow \psi^*)) \rightarrow p_{c[\psi]}$$

where $\mathbf{AG} \psi$ is an abbreviation for $\nu X.([\]X \wedge \psi)$, and ψ^* is a slight generalization of ψ^d .

Throughout the section we let \mathbb{F} and \mathbb{C} denote the sets of all ordinary and contextual formulas of the contextual μ -calculus over sets AP , V , and C , of atomic propositions, variables, and context variables, respectively. Further, we assume w.l.o.g. that all occurrences of a variable in a formula are either bound or free, and that distinct fixpoint subformulas have distinct variables. The following notation is also used throughout:

► **Definition 17.** *Given a formula $\varphi \in \mathbb{C}$ and a bound variable X occurring in φ , we let $\alpha X.\varphi_X$ denote the unique fixpoint subformula of φ binding X .*

4.1 Variable and propositional substitutions

A key tool to obtain the results of Section 3 was the substitution lemma for propositional logic. On top of atomic propositions, the μ -calculus has also variables, and we need separate substitution lemmas for both of them. We start with the variable substitution lemma. In this case, we have a μ -calculus formula φ with some free variables $X \in FV(\varphi)$ and we want to replace them by closed formulas $\sigma(X) \in \mathbb{F}$. As usual, bound variables are not replaced by variable substitutions, i.e. $\sigma(\alpha X.\phi) = \alpha.\sigma|_{V \setminus \{X\}}(\phi)$. The following lemma is a direct translation of the substitution lemma for propositional logic.

► **Lemma 18** (variable substitution lemma). *For any Kripke structure with set of states S , valuation $\eta : V \rightarrow \mathcal{P}(S)$, and substitution $\sigma : V \rightarrow \mathbb{F}$ such that $\sigma(X)$ is either X or a closed formula for all $X \in V$, we have $\llbracket \sigma(\varphi) \rrbracket_\eta = \llbracket \varphi \rrbracket_{\eta'}$ where η' is defined by $\eta'(X) := \llbracket \sigma(X) \rrbracket_\eta$.*

Replacing atomic propositions is more subtle, since they can be mapped to non-ground μ -calculus formulas. While propositions have a fixed value, the semantics of their replacements may depend on the valuation, which could be the dynamic result of fixpoint calculations appearing in the formula. Consequently, the substitution lemma will not work for an arbitrary valuation like in Lemma 18, but only for those who *match* the values of the fixpoint variables of the formula.

► **Definition 19** (fixpoint valuation). *η is a fixpoint valuation of a formula $\varphi \in \mathbb{F}$ iff $\eta(X) = \llbracket \alpha X.\phi \rrbracket_\eta$ for every bound variable X of φ .*

► **Lemma 20** (propositional substitution lemma). *For any Kripke structure $\mathcal{K} = (S, \rightarrow, I, AP, \ell)$, formula $\varphi \in \mathbb{F}$, substitution $\sigma : AP \rightarrow \mathbb{F}$ such that $\sigma(p) = p$ if p appears negated in φ , and valuation $\eta : V \rightarrow \mathcal{P}(S)$ such that $\eta(X) = \llbracket \alpha X.\sigma(\phi_X) \rrbracket_\eta$ for every subformula $\alpha X.\phi_X$ of φ , we have $\llbracket \sigma(\varphi) \rrbracket_{\mathcal{K}, \eta} = \llbracket \varphi \rrbracket_{\mathcal{K}', \eta}$ where $\mathcal{K}' = (S, \rightarrow, AP, I, \ell')$ is the Kripke structure with $\ell'(p) = \llbracket \sigma(p) \rrbracket_\eta$ for every $p \in AP$.*

These two lemmas will be very helpful in the proof of the main theorems, where we turn subformulas and variables into atomic propositions to make formulas like $\kappa_\varphi(\varphi)$ and φ_e ordinary and closed, respectively. For example, consider $\mu X.c[X] = c[\mu X.c[X]]$. If we take ψ^d instead of ψ^* in the canonical instantiation, we obtain

$$\kappa_\varphi(c) = ((\mathbf{AG}([\] \rightarrow X)) \rightarrow p_{c[X]}) \wedge ((\mathbf{AG}([\] \rightarrow \mu X.p_{c[X]}) \rightarrow p_{c[\mu X.c[X]]}) \rightarrow p_{c[\mu X.c[X]]}) \quad (9)$$

24:10 Validity of Contextual Formulas

Since X is free in this context, which value should it take? The following lemma proves there is a unique fixpoint valuation $\hat{\eta}$ for each formula φ and Kripke structure \mathcal{K} . This will give the answer to this question.

► **Lemma 21** (existence of fixpoint valuation). *For every Kripke structure \mathcal{K} and closed formula $\varphi \in \mathbb{F}$, there is a unique fixpoint valuation $\hat{\eta}$, up to variables that do not appear in φ .*

Moreover, in (9), we will be interested in getting rid of the free occurrence of X to reduce the problem to validity of ordinary closed formulas. The following lemma claims that we can replace the free variables in the formula by the fixpoint subformula defining those variables, without changing the semantics of the formula. We call the substitution achieving this, which we show to be independent of any Kripke structure, the *fixpoint substitution* of φ .

► **Lemma 22** (fixpoint substitution). *For every closed formula $\varphi \in \mathbb{F}$, there is a (unique) variable substitution $\hat{\sigma} : X \rightarrow \mathbb{F}$ such that $\hat{\sigma}(X)$ is closed and $\hat{\sigma}(X) = \hat{\sigma}(\alpha X.\phi_X)$. Moreover, for every Kripke structure \mathcal{K} , every subformula ϕ of φ , and every fixpoint valuation η for φ , we have $\llbracket \sigma(\phi) \rrbracket = \llbracket \phi \rrbracket_\eta$.*

4.2 Canonical instantiation

We are now ready to define the canonical instantiation of a contextual formula.

► **Definition 23** (canonical instantiation of a contextual formula). *Let $\varphi \in \mathbb{C}$ be a contextual formula of the μ -calculus. Given a subformula ψ of φ , let ψ^* be the result of applying to ψ^d its fixpoint substitution. The canonical instantiation of φ is the mapping $\kappa_\varphi : \mathcal{C} \rightarrow \mathcal{C}$ defined by*

$$\kappa_\varphi(c) := \bigwedge_{c[\psi] \in CSub(\varphi)} (\mathbf{AG}([\] \rightarrow \psi^*)) \rightarrow p_{c[\psi]}$$

where $\mathbf{AG} \psi$ is an abbreviation for $\nu X.([\]X \wedge \psi)$ for some fresh variable X .

We prove that φ is valid (resp. satisfiable) iff $\kappa_\varphi(\varphi)$ is valid (satisfiable). We need two lemmas. The first one is the extension of Lemma 5 to the μ -calculus.

► **Lemma 24** (monotonicity). *For every $\varphi, \psi, \psi' \in \mathbb{F}$, where only φ may contain $[\]$, fixpoint valuation $\hat{\eta}$, and every $s \in S$, if $s \in \llbracket \mathbf{AG}(\psi \rightarrow \psi') \rrbracket_{\hat{\eta}}$, then $s \in \llbracket \varphi[[\]/\psi] \rrbracket_{\hat{\eta}}$ implies $s \in \llbracket \varphi[[\]/\psi'] \rrbracket_{\hat{\eta}}$.*

The second lemma is the key one.

► **Lemma 25.** *For every $\varphi \in \mathbb{F}$, instantiation σ , and Kripke structure $\mathcal{K} = (S, \rightarrow, I, AP, \ell)$, there is a Kripke structure $\mathcal{K}' = (S, \rightarrow, AP', I, \ell')$, where $AP \subseteq AP'$ and ℓ' extends ℓ , such that $\llbracket \sigma(\varphi) \rrbracket_{\mathcal{K}} = \llbracket \kappa_\varphi(\varphi) \rrbracket_{\mathcal{K}'}$.*

Proof sketch (full proof in [8]). The ideas of Lemma 9 are reproduced here, although with the additional complication of μ -calculus variables. Given φ , σ , and \mathcal{K} , we define $\mathcal{K}' := (S, \rightarrow, I, AP', \ell')$, $AP' := AP \cup \{p_{c[\psi]} \mid c[\psi] \in CSub(\varphi)\}$, $\ell'(p) = \ell(p)$ if $p \in AP$, and $\ell'(p_{c[\psi]}) = \llbracket \sigma(c[\psi]) \rrbracket_{\hat{\eta}}$ using the fixpoint valuation $\hat{\eta}$ of Lemma 21 for $\sigma(\varphi)$. First, we show $\llbracket \phi^* \rrbracket_{\hat{\eta}} = \llbracket \phi^d \rrbracket_{\hat{\eta}}$ for every subformula ϕ of φ using the variable substitution lemma (Lemma 18) with $\gamma_*(X) = \alpha X.\phi_X$. Then, like in the proposition case, we prove $\llbracket \sigma(\phi) \rrbracket_{\hat{\eta}} = \llbracket \phi^d \rrbracket_{\hat{\eta}}$ for any subformula ϕ using the propositional substitution lemma (Lemma 20) with $\gamma_\sigma(p_{c[\psi]}) = \sigma(c[\psi])$. Finally, we prove $\llbracket \sigma(\phi) \rrbracket_{\hat{\eta}} = \llbracket \kappa_\varphi(\phi) \rrbracket_{\hat{\eta}}$ by induction using some calculation (essentially the same in Lemma 9) by the monotonicity of Lemma 24 on the expression of $\kappa_\varphi(c)$ as well as the propositional substitution lemma. ◀

► **Theorem 26.** *For every $\varphi \in \mathbb{C}$, φ is valid (resp. satisfiable) iff $\kappa_\varphi(\varphi) \in \mathbb{F}$ is valid (satisfiable).*

Proof. For validity: (\Rightarrow) If φ is valid, then $\sigma(\varphi)$ is valid for every instantiation σ . In particular, $\kappa_\varphi(\varphi)$ is valid. (\Leftarrow) Assuming $\kappa_\varphi(\varphi)$ is valid and for any instantiation σ , we must prove that $\sigma(\varphi)$ is also valid. Let \mathcal{K} be a Kripke structure, Lemma 25 claims there is another Kripke structure \mathcal{K}' such that $\llbracket \sigma(\varphi) \rrbracket_{\mathcal{K}} = \llbracket \kappa_\varphi(\varphi) \rrbracket_{\mathcal{K}'}$, so $\mathcal{K} \models \sigma(\varphi)$ iff $\mathcal{K}' \models \kappa_\varphi(\varphi)$. Moreover, we have $\mathcal{K}' \models \kappa_\varphi(\varphi)$ because $\kappa_\varphi(\varphi)$ is valid, so $\sigma(\varphi)$ is valid too since \mathcal{K} is arbitrary.

For satisfiability, (\Leftarrow) If $\kappa_\varphi(\varphi)$ is satisfiable, then φ is satisfiable by definition. (\Rightarrow) If φ is satisfiable, then $\mathcal{K} \models \sigma(\varphi)$ for some σ and \mathcal{K} . Lemma 25 then ensures $\mathcal{K}' \models \kappa_\varphi(\varphi)$ for some \mathcal{K}' , so $\kappa_\varphi(\varphi)$ is satisfiable. ◀

As in the propositional case, the length of $\kappa_\varphi(\varphi)$ grows exponentially in the nesting depth of the context. However, in the μ -calculus it can also grow exponentially even for non-nested contexts. The reason is that applying the fixpoint substitution to a μ -formula can yield an exponentially larger formula.

► **Example 27.** Consider $\varphi = \mu X_1. \dots \mu X_n. X_1 \wedge \dots \wedge X_n$ for any $n \in \mathbb{N}$, whose size is $|\varphi| = 3n - 1$. It can be proven by induction that $|\sigma(X_k)| = 2^{k-1}(3n - 2) + 1$, so $|\sigma(X_n)| = 2^{n-1}(3n - 2) + 1 = 2^{\frac{1}{3}(|\varphi|-2)}(|\varphi| + 1) + 1 \in O(2^{|\varphi|})$.³

Both problems are solved in the next section by giving an alternative reduction to validity/satisfiability of ordinary μ -calculus formulas.

4.3 A polynomial reduction

Given a contextual formula φ of the μ -calculus, we construct an ordinary formula φ_e equivalent to φ of polynomial size in φ . Following the idea of Proposition 13, we replace all context occurrences in φ by fresh atomic propositions, and insert additional conditions to ensure that the values of these propositions are consistent with what they represent. However, in the μ -calculus, these conditions may introduce unbound variables, and the strategy to remove them in Theorem 26 involves the exponential blowup attested by Example 27. To solve this problem, we also replace the free μ -calculus variables in the context occurrences by fresh atomic propositions; further, we add additional clauses to ensure that they take a value consistent with the fixpoint calculation.

- **Definition 28** (equivalid formula). *For every contextual formula $\varphi \in \mathbb{C}$,*
- *let $F = \bigcup_{c[\psi] \in CSub(\varphi)} FV(\psi)$ be the free variables in all context arguments of φ ;*
 - *for every $X \in F$, let p_X be a fresh variable that does not occur in φ ; and*
 - *for every subformula ϕ of φ , let ϕ^+ be the result of replacing every free occurrence of X in the formula ϕ^d by p_X .*

We define the ordinary formula $\varphi_e \in \mathbb{F}$ as

$$\left(\bigwedge_{\substack{c[\psi_1], c[\psi_2] \\ \in CSub(\varphi)}} \mathbf{AG} (\mathbf{AG} (\psi_1^+ \rightarrow \psi_2^+) \rightarrow (p_{c[\psi_1]} \rightarrow p_{c[\psi_2]})) \wedge \bigwedge_{X \in F} \mathbf{AG} (p_X \leftrightarrow \alpha X. \phi_X^+) \right) \rightarrow \varphi^+$$

We say \mathcal{K}' is an extension of \mathcal{K} if $\mathcal{K}' = (S, \{\rightarrow\}, I, AP', \ell')$, $AP \subseteq AP'$, and $\ell'|_{AP} = \ell$.

³ The difficulty in the previous example are fixpoint formulas with free variables. Otherwise, $|\sigma(X)| = |\sigma(\alpha X. \varphi_X)| = |\alpha X. \varphi_X| \leq |\varphi|$.

24:12 Validity of Contextual Formulas

► **Proposition 29.** *For every contextual formula $\varphi \in \mathbb{C}$, instantiation σ , and Kripke structure $\mathcal{K} = (S, \rightarrow, I, AP, \ell)$,*

1. *there is an extension \mathcal{K}' of \mathcal{K} such that $\mathcal{K} \models \sigma(\varphi)$ iff $\mathcal{K}' \models \varphi_e$.*
2. *for any extension \mathcal{K}' of \mathcal{K} such that \mathcal{K}' is a model for the premise of φ_e , $\mathcal{K}' \models \kappa_\varphi(\varphi)$ iff $\mathcal{K}' \models \varphi_e$.*

Proof sketch (full proof in [8]). We follow the main strategy of Proposition 13. (1) Using the valuation $\hat{\eta}$ given by Lemma 21 for $\sigma(\varphi)$, we define the Kripke structure \mathcal{K}' of Lemma 25 with some more variables $\ell'(p_X) = \llbracket \alpha X.\varphi_X \rrbracket_{\mathcal{K}, \hat{\eta}}$. Again, for every subformula, we prove $\llbracket \phi^d \rrbracket_{\mathcal{K}', \hat{\eta}} = \llbracket \phi^+ \rrbracket_{\mathcal{K}', \hat{\eta}}$, $\llbracket \phi^d \rrbracket_{\mathcal{K}', \hat{\eta}} = \llbracket \sigma(\phi) \rrbracket_{\mathcal{K}, \hat{\eta}}$ invoking the appropriate substitution lemmas. This let us prove that the premise of φ_e holds because of the monotonicity of contexts reflected in Lemma 24 and the matching definitions of $\hat{\eta}$ and $\ell'(p_X)$. Hence, φ_e is equivalent to its conclusion, and we have proven $\llbracket \varphi^d \rrbracket_{\mathcal{K}', \hat{\eta}} = \llbracket \sigma(\varphi) \rrbracket_{\mathcal{K}, \hat{\eta}}$, which implies the statement.

(2) Let $\varphi_e = \varphi_p \rightarrow \varphi^d$, we now assume $\mathcal{K}' \models \varphi_p$ and must prove $\mathcal{K}' \models \kappa_\varphi(\varphi)$ iff $\mathcal{K}' \models \varphi_e$, or equivalently iff $\mathcal{K}' \models \varphi^+$. Using the fixpoint valuation $\hat{\eta}$ of $\kappa_\varphi(\varphi)$, we inductively extract from the premise that $\ell'(p_X) = \hat{\eta}(X)$ and $\ell'(p_{c[\psi]}) = \llbracket \kappa_\varphi(c[\psi]) \rrbracket_{\hat{\eta}}$ with the usual calculations and substitutions. Then, we conclude that the right argument of the top implication in φ_e satisfies $\llbracket \varphi^+ \rrbracket = \llbracket \sigma(\varphi) \rrbracket$, which implies the statement. ◀

► **Theorem 30.** *A contextual μ -calculus formula $\varphi \in \mathbb{C}$ is valid iff $\varphi_e \in \mathbb{F}$ is valid.*

Proof. (\Leftarrow) φ is valid if $\sigma(\varphi)$ is valid for every instantiation σ . Let \mathcal{K} be any Kripke structure, $\mathcal{K} \models \sigma(\varphi)$ must hold. However, Proposition 29 ensures there exists \mathcal{K}' such that $\mathcal{K} \models \sigma(\varphi)$ if $\mathcal{K} \models \varphi_e$. Since φ_e is valid, we are done. (\Rightarrow) If φ is valid, so is $\kappa_\varphi(\varphi)$. We should prove that φ_e is valid, which means $\mathcal{K} \models \varphi_e$ for all \mathcal{K} . If the premise of φ_e does not hold, $\mathcal{K} \models \varphi_e$ trivially. Otherwise, Proposition 29 reduce the problem to $\mathcal{K} \models \kappa_\varphi(\varphi)$, which holds by hypothesis. ◀

► **Corollary 31.** *The validity and satisfiability problems for contextual μ -calculus formulas are EXPTIME-complete.*

Proof. The validity and satisfiability problems for ordinary formulas of the μ -calculus are EXPTIME-complete [2]. Theorem 30 gives a polynomial reduction from contextual to ordinary validity. Indeed, a rough bound is $|\varphi_e| \leq |\varphi|^3 \cdot (2|\varphi| + 5) + |\varphi| \cdot (|\varphi| + 2) + |\varphi| \leq 11|\varphi|^4$. For satisfiability, we can again replace the top implication of φ_e by a conjunction. ◀

4.4 Validity of contextual CTL formulas

We assume that the reader is familiar with the syntax and semantics of CTL (see e.g. [3]), and only fix a few notations. The syntax of contextual CTL over a set AP of atomic propositions is:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid c[\varphi] \mid \mathbf{A}(\varphi \mathbf{U} \varphi) \mid \mathbf{A}(\varphi \mathbf{W} \varphi) \mid \mathbf{E}(\varphi \mathbf{U} \varphi) \mid \mathbf{E}(\varphi \mathbf{W} \varphi)$$

where $p \in AP$, $c \in C$, and \mathbf{U} , \mathbf{W} are the strong until and weak until operators. As for the μ -calculus, the syntax of contextual CTL-formulas adds a term $c[\varphi]$, and the syntax of CTL contexts adds the hole term $[]$.

Given a Kripke structure $\mathcal{K} = (S, \rightarrow, I, AP, \ell)$ and a mapping $\ell: S \rightarrow \mathcal{P}(S)$, the semantics assigns to each ordinary formula φ the set $\llbracket \varphi \rrbracket \subseteq S$ of states satisfying φ . For example, $\llbracket \mathbf{E}(\varphi_1 \mathbf{W} \varphi_2) \rrbracket$ is the set of states s_0 such that some infinite path $s_0 s_1 s_2 \dots$ of the Kripke

structure satisfies either $s_i \in \llbracket \varphi_1 \rrbracket$ for every $i \in \mathbb{N}$, or $s_k \in \llbracket \varphi_2 \rrbracket$ and $s_0, \dots, s_{k-1} \in \llbracket \varphi_1 \rrbracket$ for some $k \geq 1$. We extend the semantics to contexts and contextual formulas as for the μ -calculus.

We proceed to solve the validity problem for contextual CTL using the syntax-guided translation from CTL to the μ -calculus [6, Pag. 1066]. The translation assigns to each CTL-formula φ a closed formula φ^μ of the μ -calculus such that $\llbracket \varphi \rrbracket = \llbracket \varphi^\mu \rrbracket$ holds for every Kripke structure \mathcal{K} and mapping ℓ .

► **Definition 32** (CTL to μ -calculus translation). *For any context or contextual CTL formula φ we inductively define the μ -calculus formula φ^μ by*

1. $p^\mu = p$
2. $(\neg p)^\mu = \neg p$
3. $([\])^\mu = [\]$
4. $(c[\psi])^\mu = c[\psi^\mu]$
5. $(\varphi_1 \wedge \varphi_2)^\mu = \varphi_1^\mu \wedge \varphi_2^\mu$
6. $(\varphi_1 \vee \varphi_2)^\mu = \varphi_1^\mu \vee \varphi_2^\mu$
7. $\mathbf{A}(\varphi_1 \mathbf{U} \varphi_2)^\mu = \mu X.([\] X \wedge \varphi_1^\mu) \vee \varphi_2^\mu$.
8. $\mathbf{A}(\varphi_1 \mathbf{W} \varphi_2)^\mu = \nu X.([\] X \wedge \varphi_1^\mu) \vee \varphi_2^\mu$.
9. $\mathbf{E}(\varphi_1 \mathbf{U} \varphi_2)^\mu = \mu X.(\langle \cdot \rangle X \wedge \varphi_1^\mu) \vee \varphi_2^\mu$.
10. $\mathbf{E}(\varphi_1 \mathbf{W} \varphi_2)^\mu = \nu X.(\langle \cdot \rangle X \wedge \varphi_1^\mu) \vee \varphi_2^\mu$.

The translations of **AF**, **AG**, **EF**, and **EG** follow by instantiating (7-10) appropriately.

The proof of the following corollary can be found in [8]. Intuitively, it is a consequence of the fact that the canonical instantiation of Definition 23 is a formula of the CTL-fragment of the μ -calculus.

► **Corollary 33.** *For any contextual CTL formula φ , let $\kappa_\varphi : C \rightarrow \text{CTL}$ be the instantiation of contexts defined by*

$$\kappa_\varphi(c) := \bigwedge_{c[\psi] \in CSub(\varphi)} (\mathbf{AG}([\] \rightarrow \psi^d)) \rightarrow p_{c[\psi]}$$

Then, the following statements are equivalent:

1. φ is valid,
2. $\kappa_\varphi(\varphi)$ is valid, and
3. $\varphi_e := \left(\bigwedge_{c[\psi_1], c[\psi_2] \in CSub(\varphi)} \mathbf{AG}(\mathbf{AG}(\psi_1^d \rightarrow \psi_2^d) \rightarrow (p_{c[\psi_1]} \rightarrow p_{c[\psi_2]})) \right) \rightarrow \varphi^d$ is valid.

Proof sketch (full proof in [8]). A straightforward induction shows that $(\sigma(\varphi))^\mu = \sigma^\mu(\varphi^\mu)$ for any instantiation σ . Moreover, $\kappa_\varphi^\mu = \kappa_{\varphi^\mu}$ and $\varphi_e^\mu = (\varphi^\mu)_e$. Hence, going back and forth between CTL and the μ -calculus, we can translate Theorems 26 and 30 to CTL. ◀

► **Corollary 34.** *The validity problem for contextual CTL formulas is EXPTIME-complete.*

Proof. The validity problem for CTL is known to be EXPTIME-complete [7], it is a specific case of the contextual validity problem, and Corollary 33 gives a polynomial reduction from the latter to the former. ◀

► **Example 35.** Let us use item (2) of Corollary 33 to show that the Boole-Shannon expansion is not valid in CTL. As in Example 11, let $\varphi := c[p] \leftrightarrow (p \wedge c[\text{true}]) \vee (\neg p \wedge c[\text{false}])$. By definition, $\kappa_\varphi(\varphi) := \kappa_\varphi(c[p]) \leftrightarrow ((p \wedge \kappa_\varphi(c[\text{true}])) \vee (\neg p \wedge \kappa_\varphi(c[\text{false}])))$. Simplification yields

$$\begin{aligned} \kappa_\varphi(\varphi) &\equiv p_{c[p]} \wedge p_{c[\text{true}]} \wedge (\mathbf{AG} \neg p \rightarrow p_{c[\text{false}]}) \\ &\quad \leftrightarrow \\ & (p \wedge (\mathbf{AG} p \rightarrow p_{c[p]}) \wedge p_{c[\text{true}]}) \vee (\neg p \wedge p_{c[p]} \wedge p_{c[\text{true}]} \wedge p_{c[\text{false}]}) \end{aligned}$$

24:14 Validity of Contextual Formulas

This formula is not valid. For example, take any Kripke structure with a state s satisfying p and $p_{c[\text{true}]}$, but neither $p_{c[p]}$ nor $\mathbf{AG} p$. Then s satisfies the right-hand side of the bi-implication, because it satisfies the left disjunct, but not the left-hand side, because it does not satisfy $p_{c[p]}$. By Corollary 33, the Boole-Shannon expansion is not valid.

Either item (2) or (3) of Corollary 33 can be used to check, for instance, that the substitution rules $\mathbf{AG}(a \leftrightarrow b) \models \mathbf{AG}(c[a] \leftrightarrow c[b])$, and $\mathbf{AG}(a \rightarrow b) \models \mathbf{AG}(c[a] \rightarrow c[b])$ do hold.

4.5 Validity of contextual LTL formulas

The syntax of LTL is obtained by dropping \mathbf{E} and \mathbf{A} from the syntax of CTL. Formulas are interpreted over infinite sequences of atomic propositions [3], and a state s_0 of a Kripke structure satisfies a formula φ if every infinite path $s_0 s_1 s_2 \dots$ of the Kripke structure satisfies φ . The Kripke structure itself satisfies φ if all its initial states satisfy φ .

Unlike for CTL, there is no syntax-guided translation from LTL to μ -calculus. However, there is one for *lassos*, finite Kripke structures in which every state has exactly one infinite path rooted at it.

► **Definition 36.** A Kripke structure $\mathcal{K} = (S, \rightarrow, I, AP, \ell)$ is a lasso if S is finite and for every $s \in S$ there is exactly one state $s' \in S$ such that $s \rightarrow s'$.

Consider the variation of the translation φ^μ from CTL where \mathbf{X} , \mathbf{U} , \mathbf{G} , and \mathbf{F} are translated as \mathbf{AX} , \mathbf{AU} , \mathbf{AG} , and \mathbf{AF} . For example, we define $(\varphi_1 \mathbf{U} \varphi_2)^\mu := \mu X.([\] X \wedge \varphi_1^\mu) \vee \varphi_2^\mu$. We have:

► **Lemma 37.** An LTL formula is valid iff it holds over all lassos. Further, for every lasso \mathcal{K} and every formula φ of LTL, $\mathcal{K} \models_{LTL} \varphi$ iff $\mathcal{K} \models \varphi^\mu$.

The results of Theorems 26 and 30 can be extended to LTL similarly to the CTL case. However, since φ^μ and φ are only guaranteed to be equivalent in lassos, some care should be taken to always use them.

► **Corollary 38.** For any LTL formula φ , let $\kappa_\varphi : C \rightarrow LTL$ be the instantiation of contexts defined by

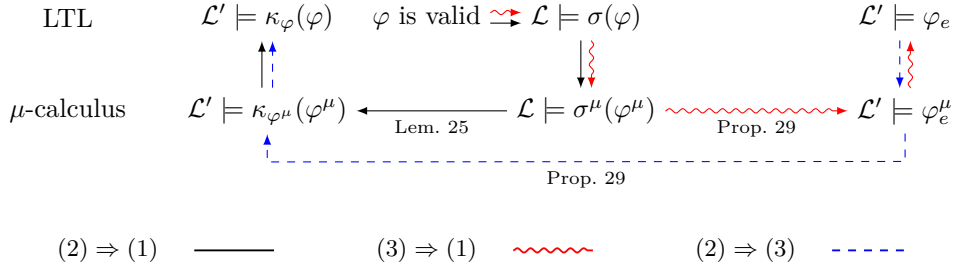
$$\kappa_\varphi(c) := \bigwedge_{c[\psi] \in CSub(\varphi)} (\mathbf{G}([\] \rightarrow \psi^d)) \rightarrow p_{c[\psi]}$$

Then, the following statements are equivalent

1. φ is valid,
2. $\kappa_\varphi(\varphi)$ is valid, and
3. $\varphi_e := \left(\bigwedge_{c[\psi_1], c[\psi_2] \in CSub(\varphi)} \mathbf{G}(\mathbf{G}(\psi_1^d \rightarrow \psi_2^d) \rightarrow (p_{c[\psi_1]} \rightarrow p_{c[\psi_2]})) \right) \rightarrow \varphi^d$ is valid.

Proof sketch (full proof in [8]). Like for CTL, $(\sigma(\varphi))^\mu = \sigma^\mu(\varphi^\mu)$, $\kappa_\varphi^\mu = \kappa_{\varphi^\mu}$, and $\varphi_e^\mu = (\varphi^\mu)_e$. Each implication of the equivalence can be derived as depicted in Figure 2. Notice that, while $\sigma(\varphi)$ and $\sigma^\mu(\varphi^\mu)$ are not equivalent in general, they are when evaluated on a lasso by Lemma 37. This allows going back and forth between LTL and the μ -calculus, and Lemma 25 and Proposition 29 complete the proof. ◀

Using the procedure just described, we can check that the rules in Figure 1 are valid, that the Boole-Shannon expansion does not hold in LTL, and one-side implications like $c[\mathbf{G}p] \models c[p]$, $\mathbf{G}(a \leftrightarrow b) \models \mathbf{G}(\varphi[a] \leftrightarrow \varphi[b])$ and $\mathbf{G}(a \rightarrow b) \models \mathbf{G}(\varphi[a] \rightarrow \varphi[b])$.



■ **Figure 2** Proof summary of Corollary 38 (arrow is problem reduction).

5 Experiments

We have implemented the methods of Propositions 10 and 13 and Corollaries 33 and 38 in a prototype that takes two contextual formulas as input and tells whether they are equivalent, one implies another, or they are incomparable.⁴ The prototype is written in Python and calls external tools for checking validity of ordinary formulas: MiniSat [5] through PySAT [10] for propositional logic, SPOT 2.11 [4] for LTL, and CTL-SAT [11] for CTL. No tool has been found for deciding μ -calculus satisfiability, so this logic is currently not supported.

Most formulas of Figure 1 are solved in less than 10 milliseconds by the first and second methods. The hardest formula is the second one: using the equivalid formula, the largest automaton that appears in the process has 130 states and it is solved in 16.21 ms; with the canonical instantiation, the numbers are 36 and 20.22 ms. We have also applied small mutations to the rules of Figure 1 to yield other identities that may or may not hold (see Appendix C of [8] for a list). Solving them takes roughly the same time and memory as the original ones. For CTL, the behavior even with small formulas is much worse because of the worse performance of CTL-SAT. The canonical context method takes 20 minutes to solve $c[a \wedge b] \equiv c[a] \wedge c[b]$, while the method by the equivalid formula runs out of memory with that example and requires 22 minutes for the Boole-Shannon expansion. Hence, we have not continued with further benchmarks on CTL. The first three rows of Table 1 show the time, peak memory usage (in megabytes), and number of states of the automata (for LTL) required for checking the aforementioned examples. The experiments have been run under Linux in an Intel Xeon Silver 4216 machine limited to 8 Gb of RAM. Memory usage is as reported by Linux cgroups' memory controller.

In addition to these natural formulas, we have tried with some artificial ones with greater sizes and nested contexts to challenge the performance of the algorithm. We have considered two repetitive expansions of the rules in Figure 1:

1. There is a dual of the first rule in Figure 1 that removes a **W**-node below a **U**-node:

$$\varphi \mathbf{U} c[\psi_1 \mathbf{W} \psi_2] \equiv \varphi \mathbf{U} c[\psi_1 \mathbf{U} \psi_2] \vee (\mathbf{F} \mathbf{G} \varphi \wedge (\varphi \wedge \mathbf{F} c[\text{true}]) \mathbf{W} c[\psi_1 \mathbf{W} \psi_2]).$$

Then, we can build formulas like $c_1[\psi_1 \mathbf{U} \psi_2] \mathbf{W} \varphi$ ($n = 1$), $c_1[\psi_1 \mathbf{U} c_2[\psi_2 \mathbf{W} \psi_3]] \mathbf{W} \varphi$ ($n = 2$), $c_1[\psi_1 \mathbf{U} c_2[\psi_2 \mathbf{W} c_3[\psi_3 \mathbf{U} \psi_4]]] \mathbf{W} \varphi$ ($n = 3$), and so on, and apply the first rule of the rewrite system and its dual to obtain the normalized right-hand side. Table 1 shows that the first method does not finish within an hour for $n = 2$, and the second reaches this time limit for $n = 3$. For $n = 2$ the second method checks the emptiness of an automaton of 1560 states (and another of 720 states for the other side of the implication).

⁴ The prototype and its source code are publicly available at <https://github.com/ningit/ctxform>.

■ **Table 1** Compared performance with the challenging examples (memory in Mb).

Example		Method 1			Method 2		
		Time	Memory	States	Time	Memory	States
Shannon	Bool	8.18 ms	3.67		15.08 ms	3.67	
	LTL	5 ms	2.62	9	8.07 ms	2.62	12
Rules [9] (max)		23.88 ms	5.24	36	16.21 ms	4.71	130
Mutated (max)		44.12 ms	5.72	48	31.68 ms	4.98	130
(1)	0	1.66 ms	1.05	4	1.53 ms	1.05	4
	1	17.40 ms	4.92	36	15.62 ms	4.46	130
	2	timeout			3:25 min	413.83	1560
(2)	1	36.19 ms	5.24	45	22.40 ms	5.24	80
	2	623.07 ms	26.96	168	117.79 ms	10.49	160
	3	5:21 min	1245.95	1140	26.04 s	100.25	220

2. The third and fourth rules of Figure 1 can also be nested. We can consider $c_0[\mathbf{FG} c_1[p]]$ ($n = 1$), $c_0[\mathbf{FG} c_1[\mathbf{GF} c_2[p]]]$ ($n = 2$), $c_0[\mathbf{FG} c_1[\mathbf{GF} c_2[\mathbf{FG} c_3[p]]]]$ ($n = 3$), and so on. We also take $c_0 = c_1 = \dots = c_n$ to make the problem harder. As shown in Table 1, we can solve up to $n = 3$ within the memory constraints.

6 Conclusions

We have presented two different methods to decide the validity and satisfiability of contextual formulas in propositional logic, LTL, CTL, and the μ -calculus. Moreover, we have shown that these problems have the same complexity for contextual and ordinary formulas. Interesting contextual equivalences can now be checked automatically. In particular, we have replaced the manual proofs of the several LTL simplification rules in [9] to a few milliseconds of automated check.



While we have limited our exposition to formulas in negation normal form, and hence to monotonic contexts, the results for propositional logic, CTL, and LTL can be generalized to the unrestricted syntax of the corresponding logics and to arbitrary contexts. Some clues are given in Appendix B of [8].

References

- 1 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. doi:10.3233/FAIA336.
- 2 Julian C. Bradfield and Igor Walukiewicz. The mu-calculus and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 871–919. Springer, 2018. doi:10.1007/978-3-319-10575-8_26.
- 3 Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. Introduction to model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 1–26. Springer, 2018. doi:10.1007/978-3-319-10575-8_1.
- 4 Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 2022. doi:10.1007/978-3-031-13188-2_9.

- 5 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.
- 6 E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier and MIT Press, 1990. doi:10.1016/B978-0-444-88074-1.50021-4.
- 7 E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985. doi:10.1016/0022-0000(85)90001-7.
- 8 Javier Esparza and Rubén Rubio. Validity of contextual formulas (extended version). *arXiv*, 2024. doi:10.48550/arXiv.2407.07759.
- 9 Javier Esparza, Rubén Rubio, and Salomon Sickert. Efficient normalization of linear temporal logic. *J. ACM*, 71(2):16:1–16:42, 2024. doi:10.1145/3651152.
- 10 Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018. doi:10.1007/978-3-319-94144-8_26.
- 11 Nicola Prezza. CTL (Computation Tree Logic) SAT solver, 2014. URL: <https://github.com/nicolaprezza/CTLSAT>.
- 12 Rubén Rubio. Equivalence checker for contextual formulas. Software (visited on 2024-07-25). URL: <https://github.com/ningit/ctxform>.

A Unifying Categorical View of Nondeterministic Iteration and Tests

Sergey Goncharov  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Tarmo Uustalu  

Reykjavik University, Iceland, and Tallinn University of Technology, Estonia

Abstract

We study Kleene iteration in the categorical context. A celebrated completeness result by Kozen introduced Kleene algebra (with tests) as a ubiquitous tool for lightweight reasoning about program equivalence, and yet, numerous variants of it came along afterwards to answer the demand for more refined flavors of semantics, such as stateful, concurrent, exceptional, hybrid, branching time, etc. We detach *Kleene iteration* from *Kleene algebra* and analyze it from the categorical perspective. The notion, we arrive at is that of *Kleene-iteration category* (with coproducts and tests), which we show to be general and robust in the sense of compatibility with programming language features, such as exceptions, store, concurrent behaviour, etc. We attest the proposed notion w.r.t. various yardsticks, most importantly, by characterizing the free model as a certain category of (nondeterministic) *rational trees*.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Axiomatic semantics

Keywords and phrases Kleene iteration, Elgot iteration, Kleene algebra, coalgebraic resumptions

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.25

Related Version *Full Version*: <https://arxiv.org/abs/2407.08688>

Funding *Sergey Goncharov*: German Research Foundation (DFG) project 501369690, Icelandic Research Fund project 228684-052

Tarmo Uustalu: Icelandic Research Fund project 228684-052

1 Introduction

Axiomatizing notions of iteration both algebraically and categorically is a well-established topic in computer science where two schools of thought can be distinguished rather crisply: the first one is based on the inherently nondeterministic *Kleene iteration*, stemming from the seminal work of Stephen Kleene [22] and deeply rooted in automata and formal language theory; the second one stems from another seminal work – by Calvin Elgot [12] – and is based on another notion of iteration, we now call *Elgot iteration*. The most well-known instance of Kleene iteration is the one that is accommodated in the algebra of regular expressions where a^* represents n -fold compositions $a \cdot \dots \cdot a$ and n nondeterministically ranges over all naturals. More abstractly, Kleene iteration is an operation of the following type:

$$\frac{p: X \rightarrow X}{p^*: X \rightarrow X}$$

Intuitively, we think of p as a program whose inputs and outputs range over X , and of p^* as a result of composing p nondeterministically many times with itself. Elgot iteration, in contrast, is agnostic to nondeterminism, but crucially relies on the categorical notion of binary coproduct, and thus can only be implemented in categorical or type-theoretic setting.



© Sergey Goncharov and Tarmo Uustalu;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 25; pp. 25:1–25:22

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Concretely, the typing rule for Elgot iteration is

$$\frac{p: X \rightarrow Y + X}{p^\dagger: X \rightarrow Y} \quad (\dagger)$$

That is, given a program that receives an input from X , and can output either to Y or to X , p^\dagger self-composes p precisely as long as p outputs to X .

A profound exploration of both versions of iteration and their axiomatizations in the categorical context, more precisely, in the context of *Lavwere theories*, has been done by Bloom and Ésik in a series of papers and subsumed in their monograph [6]. One outcome of this work is that in the context of Lavwere theories, in presence of nondeterminism, Kleene iteration and Elgot iteration are essentially equivalent – the ensuing theory was dubbed *iteration grove theory* [5]. The existing analysis still does not cover certain aspects, which we expressly address in our present work, most importantly, the following.

- Lavwere theories are only very special categories, while iteration is a common ingredient of semantic frameworks, which often involve it directly via an ambient category with coproducts, and not via the associated Lavwere theory.
- Previous results on the equivalence of Elgot iteration and Kleene iteration do not address the connection between control mechanisms involved in both paradigms: Elgot iteration fully relies on coproducts for making decisions whether to continue or to end the loop, while Kleene iteration for the same purpose uses an additional mechanism of *tests* [24], which are specified axiomatically and thus yield a higher degree of flexibility.
- A key feature of Kleene iteration of Kleene algebra, are the quasi-equational laws, which can be recast [16] to a form of the versatile and powerful *uniformity principle* (e.g. [36]). The latter is parameterized by a class of well-behaved elements, which in Kleene algebra coincide with the algebra’s entire carrier. However, in many situations, this class has to be restricted, which calls for axiomatizing it, analogously to tests.

Here, we seek a fundamental, general and robust categorical notion of Kleene iteration, which addresses these issues, is in accord with Elgot iteration and the corresponding established laws for it (Elgot iteration operators that satisfy these laws are called *Conway operators*). In doing so, we depart from the laws of Kleene algebra, and relax them significantly. Answering the question how to do this precisely and in a principled way is the main insight of our work.

Let us dwell briefly on the closely related issues of generality and robustness. The laws of Kleene algebra, as originally axiomatized by Kozen [23], capture a very concrete style of semantics, mirrored in the corresponding free model, which is the algebra of regular events, i.e. the algebra of regular sets of strings over a finite alphabet of symbols, with iteration rendered as a least fixpoint. Equations validated by this model are thus shared by the whole class of Kleene algebras. By regarding Kleene algebra terms as programs, the interpretation over the free model can be viewed as *finite trace semantics* of linear-time nondeterminism. A standard example of properly more fine-grained – branching-time – nondeterminism is (bisimulation-based) process algebra, which fails the Kleene algebra’s law of distributivity from the left:

$$p ; (q + r) = p ; q + p ; r. \quad (1)$$

Similarly, if we wanted to allow our programs to raise exceptions, the laws of Kleene algebra would undesirably force all exceptions to be equal:

$$\text{raise } e_1 = \text{raise } e_1 ; 0 = 0 = \text{raise } e_2 ; 0 = \text{raise } e_2.$$

Here, we combine the law $p; 0 = 0$ of Kleene algebra with the equation $\text{raise}_i; p = \text{raise}_i$ that alludes to the common programming knowledge that raising an exception exits the program instantly and discards any subsequent fragment p . The resulting equality $\text{raise } e_1 = \text{raise } e_2$ states that raising exception e_1 is indistinguishable from raising exception e_2 .

We can interpret these and similar examples as evidence that the axioms of Kleene algebra are not sufficiently robust under extensions by programming language features. More precisely, Kleene algebras can be scaled up to *Kleene monads* [17], and thus reconciled with Moggi's approach to computational effects [30]. An important ingredient of this approach are *monad transformers*, which allow for combining effects in a principled way. For example, one uses the *exception monad transformer* to canonically add exception raising to a given monad. The above indicates that Kleene monads are not robust under this transformer.

Finally, even if we accept all iteration-free implications of Kleene algebra, these will not jointly entail the following identity:

$$1^* = 1, \tag{2}$$

which is however entailed by the Kleene algebra axioms. One setting where (2) is undesirable is domain theory, which insists on distinguishing *deadlock* from *divergence*, in particular, (2) is failed by interpreting programs over the *Plotkin powerdomain* [33]. Intuitively, (2) states that, if a loop *may* be exited, it *will* eventually be exited, while failure of (2) would mean that the left program *may* diverge, while the right program *must* converge, and this need not be the same. Let us call the corresponding variant of Kleene algebra, failing (2), *may-diverge Kleene algebra*. However, it is not a priori clear how the axioms of may-diverge Kleene algebras must look like, given that (2) is not a Kleene algebra axiom, but a consequence of the assumption that Kleene iteration is a least fixpoint. Hence, in may-diverge Kleene algebras Kleene iteration is not a least fixpoint (w.r.t. the order, induced by $+$).

The notion we develop and present here is that of *Kleene-iteration category (with tests)* ($KiC(T)$). It is designed to address the above issues and to provide a uniform general and robust framework for Kleene iteration in a category. We argue in various ways that $KiC(T)$ is in a certain sense the most basic practical notion of Kleene iteration, most importantly by characterizing its free model, as a certain category of (nondeterministic) rational trees.

Related work. The (finite or ω -complete) partially additive categories (PACs) by Arbib and Manes [2] and the PACs with effects of Cho [8] are similar in spirit to KiC s in that they combine structured homsets and coproducts, but significantly more special; in particular they support relational, sets of traces and similar semantics, but not branching time semantics. A PAC is a category with coproducts enriched in partial commutative monoids (PMC). The PMC structure of homsets and the coproducts are connected by axioms that make the PMC structure unique. In an ω -complete PAC, these axioms also ensure the presence of an Elgot iteration operator, which is computed as a least fixpoint. A PAC with effects comes with a designated effect algebra object; this defines a wide subcategory of total morphisms, with coproducts inherited from the whole category. Effectuses [21] achieve the same as PACs with effects, but starting with a category of total morphisms and then adding partial morphisms. Cockett [9] recently proposed a notion of iteration in a category, based on restriction categories, and analogous to Elgot iteration (\dagger), but avoiding binary coproducts in favor of a suitably axiomatized notion of disjointness for morphisms.

In the strand of Kleene algebra, various proposals were made with utilitarian motivations to weaken or modify the Kleene algebra laws, and thus to cope with process algebra [14], branching behaviour [31], probability [27], statefulness [20], graded semantics [15], without

however aiming to identify the conceptual core of Kleene iteration, which is our objective here. A recent move within this tendency is to eliminate nondeterminism altogether, with *guarded Kleene algebras* [37], which replace nondeterministic choice and iteration with conditionals and while-loops. This is somewhat related to our analysis of tests and iteration via while-loops, but largely orthogonal to our main objective to stick to Kleene iteration as nondeterministic operator in the original sense. Our aim to reconcile Kleene algebra, (co)products and Elgot iteration is rather close to that of Kozen and Mamouras [25].

Our characterization of the free KiCT in a way reframes the original Kozen’s characterization of the free Kleene algebra [23]. We are not generalizing this result though, essentially because we work in categories with coproducts, while a true generalization would only be achieved via categories without any extra structure (noting that algebras are single-object categories). This distinction becomes particularly important in the context of branching time semantics, which we also cover by allowing a controlled use of programs that fail distributivity from the left (1). An axiomatization for such semantics has been proposed by Milner [29] and was shown to be complete only recently [19]. Again, we are not generalizing this result, since the definability issues, known to be the main obstruction for completeness arguments there, are not effective in presence of coproducts.

Plan of the paper. We review minimal notations and conventions from category theory in Section 2. We then introduce idempotent grove and Kleene-Kozen categories in Section 3 to start off. In Section 4, we formally compare two control mechanisms in categories: decisions and tests. In Sections 5, 6, we establish equivalent presentations of nondeterministic iteration as Kleene iteration, as Elgot iteration and as while-iteration. In Section 7 we construct a free model for our notion of iteration, and then come to conclusions in Section 8.

2 Notations and Conventions

We assume familiarity with the basics of category theory [26, 3]. In a category \mathbf{C} , $|\mathbf{C}|$ will denote the class of objects and $\mathbf{C}(X, Y)$ will denote the set of morphisms from X to Y . The judgement $f: X \rightarrow Y$ will be regarded as an equivalent to $f \in \mathbf{C}(X, Y)$ if \mathbf{C} is clear from the context. We tend to omit indexes at natural transformations for readability. A subcategory \mathbf{D} of \mathbf{C} is called *wide* if $|\mathbf{C}| = |\mathbf{D}|$. We will use diagrammatic composition $;$ of morphisms throughout, i.e. given $f: X \rightarrow Y$ and $g: Y \rightarrow Z$, $f; g: X \rightarrow Z$. We will denote by 1_X , or simply 1 the identity morphism on X .

Coproducts. In this paper, by calling \mathbf{C} “a category with coproducts” we will always mean that \mathbf{C} has *selected binary coproducts*, i.e. that a bi-functor $\oplus: \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$ exists such that $X \oplus Y$ is a coproduct of X and Y . In such a category, we write $\text{in}_0: X \rightarrow X \oplus Y$ and $\text{in}_1: Y \rightarrow X \oplus Y$ for the left and right coproduct injections correspondingly. We will occasionally condense $\text{in}_i; \text{in}_j$ to in_{ij} for the sake of succinctness.

Monads. A monad \mathbf{T} on \mathbf{C} is determined by a *Kleisli triple* $(T, \eta, (-)^\sharp)$, consisting of a map $T: |\mathbf{C}| \rightarrow |\mathbf{C}|$, a family of morphisms $(\eta_X: X \rightarrow TX)_{X \in |\mathbf{C}|}$ and *Kleisli lifting* sending each $f: X \rightarrow TY$ to $f^\sharp: TX \rightarrow TY$ and obeying *monad laws*:

$$\eta^\sharp = 1, \quad \eta; f^\sharp = f, \quad (g; f^\sharp)^\sharp = g^\sharp; f^\sharp.$$

It follows that T extends to a functor, η extends to a natural transformation – *unit*, $\mu = 1^\sharp: TTX \rightarrow TX$ extends to a natural transformation – *multiplication*, and that (T, η, μ) is a monad in the standard sense [26]. We will generally use blackboard capitals (such as \mathbf{T}) to refer to monads and the corresponding Roman letters (such as T) to refer to their functor parts. Morphisms of the form $f: X \rightarrow TY$ are called *Kleisli morphisms* and form the *Kleisli category* $\mathbf{C}_{\mathbf{T}}$ of \mathbf{T} under *Kleisli composition* $f, g \mapsto f; g^\sharp$ with identity η . If \mathbf{C} has binary coproducts then so does $\mathbf{C}_{\mathbf{T}}$: the coproduct injections are Kleisli morphisms of the form $\text{in}_0; \eta: X \rightarrow T(X \oplus Y)$, $\text{in}_1; \eta: Y \rightarrow T(X \oplus Y)$.

Coalgebras. Given an endofunctor $F: \mathbf{C} \rightarrow \mathbf{C}$, a pair $(X \in |\mathbf{C}|, c: X \rightarrow FX)$ is called an F -coalgebra. Coalgebras form a category under the following notion of morphism: $h: X \rightarrow X'$ is a morphism from (X, c) to (X', c') if $h; c' = c; Fh$. A terminal object in this category is called a *final coalgebra*. We reserve the notation $(\nu F, \text{out})$ for a selected final coalgebra if it exists. A well-known fact (Lambek's lemma) is that out is an isomorphism.

For a coalgebra $(X, c: X \rightarrow FX)$ on \mathbf{Set} a relation $\mathcal{B} \subseteq X \times X$ is a (*coalgebraic*) *bisimulation* if it extends to a coalgebra $(\mathcal{B}, b: \mathcal{B} \rightarrow F\mathcal{B})$, such that the left and the right projections from \mathcal{B} to X are coalgebra morphisms; $x \in X$ and $y \in X$ are *bisimilar* if $x \mathcal{B} y$ for some bisimulation \mathcal{B} ; the coalgebra $(X, c: X \rightarrow FX)$ is *strongly extensional* [38] if bisimilarity entails equality. Final coalgebras are the primary example of strongly extensional coalgebras.

3 Idempotent Grove and Kleene-Kozen Categories

A monoid is precisely a single-object category. Various algebraic structures extending monoids can be generalized to categories along this basic observation (e.g. a group is a single-object groupoid, a quantale is a single-object quantaloid, etc.). In this section, we consider two classes of categories for nondeterminism and Kleene iteration, which demonstrate our principled categorical approach of working with algebraic structures.

► **Definition 1** (Idempotent Grove Category, cf. [4, 5]). *Let us call a category \mathbf{C} an idempotent grove category if the hom-sets of $\mathbf{C}(X, Y)$ are equipped with the structure $(0, +)$ of bounded join-semilattice such that, for all $p \in \mathbf{C}(Y, Z)$ and $q, r \in \mathbf{C}(X, Y)$,*

$$0; p = 0, \quad (q + r); p = q; p + r; p. \quad (3)$$

In such a category, we call a morphism $p \in \mathbf{C}(X, Y)$ linear if it satisfies, for all $q, r \in \mathbf{C}(Y, Z)$,

$$p; 0 = 0, \quad p; (q + r) = p; q + p; r. \quad (4)$$

An idempotent grove category with coproducts is an idempotent grove category with selected binary coproducts and with in_0 and in_1 linear.

Given $p, q \in \mathbf{C}(X, Y)$, let $p \leq q$ if $p + q = q$. This yields a partial order with 0 as the bottom element, and morphism composition is monotone on the left, while linear morphisms are additionally monotone on the right. The class of all linear morphisms of an idempotent grove category thus forms a sub-category enriched in bounded join-semilattices (equivalently: commutative and idempotent monoids) – thus, an idempotent grove category where all morphisms are linear is an enriched category. However, we are interested in categories where not all morphisms are linear. An instructive example is as follows.

► **Example 2** (Synchronization Trees). Let A be some non-empty fixed set of labels, and let $TX = \nu\gamma. \mathcal{P}_{\omega_1}(X \oplus A \times \gamma)$ where \mathcal{P}_{ω_1} is the countable powerset functor. By generalities [39], T extends to a monad \mathbf{T} on \mathbf{Set} . The elements of TX can be characterized as *countably-branching strongly extensional synchronization trees with exit labels in X* . Synchronization trees have originally been introduced by Milner [28] as denotations of process algebra terms, and subsequently generalized to infinite branching and to explicit exit labels (e.g. [1]). A generic element $t \in TX$ can be more explicitly represented using the following syntax:

$$t = \sum_{i \in I} a_i \cdot t_i + \sum_{i \in J} x_i$$

where I and J are at most countable, the a_i range over A , the t_i range over TX , and x_i range over X . The involved summation operators $\sum_{i \in I}$ are considered modulo countable versions of associativity, commutativity and idempotence, and $0 = \sum_{i \in \emptyset} t_i$ and $t_1 + t_2 = \sum_{i \in \{1,2\}} t_i$.

Recall that strong extensionality means that bisimilar elements are equal [38]. The Kleisli category of \mathbf{T} is idempotent grove with 0 and $+$ inherited from \mathcal{P}_{ω_1} and ensuring (3) automatically. It is easy to see that linear morphisms are precisely those that do not involve actions.

A straightforward way to add a Kleene iteration operator to a category is as follows.

► **Definition 3** (Kleene-Kozen Category [16]). *An idempotent grove category \mathbf{C} is a Kleene-Kozen category if all morphisms of \mathbf{C} are linear and there is a Kleene iteration operator $(-)^*: \mathbf{C}(X, X) \rightarrow \mathbf{C}(X, X)$ such that, for any $p: X \rightarrow X$, $q: Y \rightarrow X$ and $r: X \rightarrow Z$, the morphism q ; p^* is the least (pre-)fixpoint of $q + (-)$; p and the morphism p^* ; r is the least (pre-)fixpoint of $r + p$; $(-)$.*

It is known [16] that Kleene algebra is precisely a single-object Kleene-Kozen category.

In idempotent grove categories with coproducts, the following property is a direct consequence of linearity of in_0 , in_1 , and will be used extensively throughout.

► **Proposition 4.** *In idempotent grove categories with coproducts, $[p, q] + [p', q'] = [p + p', q + q']$.*

4 Decisions and Tests in Category

We proceed to compare two mechanisms for modeling control in categories: *decisions* and *tests*. The first one is inherently categorical, and requires coproducts. The second one needs no coproducts, but requires nondeterminism. The latter one is directly inspired by tests of the Kleene algebra with tests [24]. We will show that tests and decisions are in a suitable sense equivalent, when it comes to modeling control that satisfies Boolean algebra laws.

► **Definition 5** (Decisions [10, 16]). *In a category \mathbf{C} with binary coproducts, we call morphisms from $\mathbf{C}(X, X \oplus X)$ decisions.*

We consider the following operations on decisions, modeling truth values and logical connectives: $\text{tt} = \text{in}_1$ (true), $\text{ff} = \text{in}_0$ (false), $\sim d = d$; $[\text{in}_1, \text{in}_0]$ (negation), $d \parallel e = d$; $[e, \text{in}_1]$ (disjunction), $d \&\& e = d$; $[\text{in}_0, e]$ (conjunction). Even without constraining decisions in any way, certain logical properties can be established, e.g. (not necessarily commutative or idempotent) monoidal structures (ff, \parallel) , $(\text{tt}, \&\&)$, involutivity of \sim , “de Morgan laws” $\sim(d \parallel e) = \sim d \&\& \sim e$, $\sim(d \&\& e) = \sim d \parallel \sim e$, and the laws $\text{tt} \parallel d = \text{tt}$, $\text{ff} \&\& d = \text{ff}$.

Given $d \in \mathbf{C}(X, X \oplus X)$ and $p, q \in \mathbf{C}(X, Y)$, let

$$\text{if } d \text{ then } p \text{ else } q = d; [q, p]. \tag{5}$$

► **Definition 6 (Tests).** Given an idempotent grove category \mathbf{C} , we call a family of linear morphisms $\mathbf{C}^? = (\mathbf{C}^?(X) \subseteq \mathbf{C}(X, X))_{X \in |\mathbf{C}|}$ tests if every $\mathbf{C}^?(X)$ forms a Boolean algebra under $;$ as conjunction and $+$ as disjunction.

It follows that $1 \in \mathbf{C}^?(X)$ and $0 \in \mathbf{C}^?(X)$ correspondingly are the top and bottom elements of $\mathbf{C}^?(X)$. Given $b \in \mathbf{C}^?(X)$, $p, q \in \mathbf{C}(X, Y)$, let

$$\text{if } b \text{ then } p \text{ else } q = b ; p + \bar{b} ; q. \quad (6)$$

In an idempotent grove category \mathbf{C} with coproducts and tests $\mathbf{C}^?$, let $?: \mathbf{C}(X, X \oplus X) \rightarrow \mathbf{C}^?(X)$ be the morphism $d? = d ; [0, 1]$.

► **Proposition 7.** Let \mathbf{C} be an idempotent grove category with coproducts. If a decision d is linear, then, for all p and q , we have $\text{if } d \text{ then } p \text{ else } q = \text{if } d? \text{ then } p \text{ else } q$.

Let us say that a pair $(b, c) \in \mathbf{C}(X, X) \times \mathbf{C}(X, X)$ satisfies (the law of) contradiction if $b ; c = 0$, and that it satisfies (the law of) excluded middle if $b + c = 1$. The following characterization is instructive.

► **Proposition 8.** Given an idempotent grove category \mathbf{C} , a family of linear morphisms $\mathbf{C}^? = (\mathbf{C}^?(X) \subseteq \mathbf{C}(X, X))_{X \in |\mathbf{C}|}$ forms tests for \mathbf{C} iff, for every $b \in \mathbf{C}^?$, there is $\bar{b} \in \mathbf{C}^?$ such that (b, \bar{b}) satisfies contradiction and excluded middle.

Note that the smallest choice of tests in \mathbf{C} is $\mathbf{C}^?(X) = \{0, 1\}$. We proceed to characterize the smallest possible choice of tests, sufficient for modeling control.

► **Definition 9 (Expressive Tests).** We call the tests $\mathbf{C}^?$ expressive if every $\mathbf{C}^?(X)$ contains $[\text{in}_0, 0]$ whenever $X = X_1 \oplus X_2$.

► **Lemma 10.** The smallest expressive family of tests always exists and is obtained by closing tests of the form $0, 1, [\text{in}_0, 0]$, and $[0, \text{in}_1]$ under $+$ and $;$.

In the sequel, we will use the notation $\top, \perp, \wedge, \vee$ for tests, synonymously to $1, 0, ;, +$ to emphasize their logical character.

► **Lemma 11.** Let $\mathbf{C}^?$ be tests in an idempotent grove category \mathbf{C} with binary coproducts.

1. The morphisms $\diamond : \mathbf{C}^?(X) \rightarrow \mathbf{C}(X, X \oplus X)$, $?: \mathbf{C}(X, X \oplus X) \rightarrow \mathbf{C}^?(X)$ defined by $\diamond b = \bar{b} ; \text{in}_0 + b ; \text{in}_1$, $d? = d ; [0, 1]$ form a retraction.
2. Every morphism d in the image of \diamond is linear. Moreover, we have $d ; \nabla = 1$, $d = d \&\& d$, and $d = d \parallel d$.
3. For all e and d in the image of \diamond , it holds that $(e \parallel d)? = e? \vee d?$, $(e \&\& d)? = e? \wedge d?$, and $(\sim d)? = \bar{d}?$.

Lemma 11 indicates that in presence of coproducts and with linear coproduct injections, instead of Boolean algebras on subsets of $\mathbf{C}(X, X)$, one can equivalently work with Boolean algebras on subsets of $\mathbf{C}(X, X \oplus X)$.

We conclude this section by an illustration that varying tests, in particular, going beyond smallest expressive tests is practically advantageous.

► **Example 12.** Consider the nondeterministic state monad \mathbf{T} with $TX = \mathcal{P}(S \times X)^S$ on \mathbf{Set} , where S is a fixed global store, which the programs, represented by Kleisli morphisms of \mathbf{T} are allowed to read and modify. Morphisms of the Kleisli category $\mathbf{Set}_{\mathbf{T}}$ are equivalently (by uncurrying) maps of the form $p: S \times X \rightarrow \mathcal{P}(S \times Y)$, meaning that $\mathbf{Set}_{\mathbf{T}}$ is equivalent to a full subcategory of $\mathbf{Set}_{\mathcal{P}}$, from which $\mathbf{Set}_{\mathbf{T}}$ inherits the structure of an idempotent grove

$$\begin{array}{l}
 \text{in}_0; [p, q] = p \quad \text{in}_1; [p, q] = q \quad [\text{in}_0, \text{in}_1] = 1 \quad [p, q]; r = [p; r, q; r] \\
 0 + p = p \quad p + p = p \quad p + q = q + p \quad (p + q) + r = p + (q + r) \\
 0; p = 0 \quad (q + r); p = q; p + r; p \quad u; 0 = 0 \quad u; (p + q) = u; p + u; q \\
 \text{(*-Fix)} \quad p^* = 1 + p; p^* \quad \text{(*-Sum)} \quad (p + q)^* = p^*; (q; p^*)^* \quad \text{(*-Uni)} \quad \frac{u; p = q; u}{u; p^* = q^*; u}
 \end{array}$$

■ **Figure 1** Axioms of KiCs, including binary coproducts (p, q, r range over \mathbf{C} , u ranges over $\bar{\mathbf{C}}$).

category. The tests identified in Lemma 10 are those maps $b: S \times X \rightarrow \mathcal{P}(S \times X)$ that are determined by decompositions $X = X_1 \oplus X_2$, in particular, they can neither read nor modify the store. In practice, only the second is regarded as undesirable (and indeed would break commutativity of tests), while reading is typically allowed. This leads to a more permissive notion of tests, as those that are determined by the decompositions $S \times X = X_1 \oplus X_2$.

5 Kleene Iteration, Categorically

We now can introduce our central definition by extending idempotent grove categories with a selected class of linear morphisms, called *tame morphisms*, and with Kleene iteration. Crucially, we assume the ambient category \mathbf{C} to have coproducts as a necessary ingredient. Finding a general definition, not relying on coproducts, presently remains open.

- **Definition 13** (KiC(T)). *We call a tuple $(\mathbf{C}, \bar{\mathbf{C}})$ a Kleene-iteration category (KiC) if*
1. \mathbf{C} is an idempotent grove category with coproducts;
 2. $\bar{\mathbf{C}}$ is a wide subcategory of \mathbf{C} , whose morphisms we call *tame* such that
 - $\bar{\mathbf{C}}$ has coproducts strictly preserved by the inclusion to \mathbf{C} ;
 - the morphisms of $\bar{\mathbf{C}}$ are all linear;
 3. for every $X \in |\mathbf{C}|$, there is a Kleene iteration operator $(-)^*: \mathbf{C}(X, X) \rightarrow \mathbf{C}(X, X)$ such that the laws ***-Fix**, ***-Sum** and ***-Uni** in Figure 1, with u ranging over $\bar{\mathbf{C}}$, are satisfied.

A functor $F: (\mathbf{C}, \bar{\mathbf{C}}) \rightarrow (\mathbf{D}, \bar{\mathbf{D}})$ between KiCs is a coproduct preserving functor $F: \mathbf{C} \rightarrow \mathbf{D}$ such that $F0 = 0$, $F(q + r) = Fq + Fr$ and $Fp^* = (Fp)^*$ for all $q, r \in \mathbf{C}(X, Y)$, $p \in \mathbf{C}(X, X)$, and $Fp \in \bar{\mathbf{D}}(FX, FY)$ for all $p \in \bar{\mathbf{C}}(X, Y)$.

A KiC $(\mathbf{C}, \bar{\mathbf{C}})$ equipped with a choice of tests $\mathbf{C}^?$ in $\bar{\mathbf{C}}$ we call a KiCT (=KiC with tests). Correspondingly, functors between KiCTs are additionally required to send tests to tests.

It transpires from the definition that the role of tameness is to limit the power of the uniformity rule ***-Uni**. The principal case for $\mathbf{C} \neq \bar{\mathbf{C}}$ is Example 2. As we see later (Example 24), this yields a KiC. More generally, unless we restrict $\bar{\mathbf{C}}$ to programs that satisfy the linearity laws (4), the uniformity principle ***-Uni** would tend to be unsound. Very roughly, uniformity is some infinitary form of distributivity from the left and it fails for programs that fail the standard left distributivity. This phenomenon is expected to occur for other flavors of concurrent semantics: as long as \mathbf{C} admits morphisms that fail (4), $\bar{\mathbf{C}}$ would have to be properly smaller than \mathbf{C} . Apart from concurrency, if $\bar{\mathbf{C}}$ models a language with exceptions, those must be excluded from $\bar{\mathbf{C}}$, for otherwise uniformity would again become unsound.

If we demand all morphisms to be tame, we will obtain a notion very close to that of Kleene-Kozen category (Definition 3).

► **Definition 14** (*-Idempotence). *A KiC is *-idempotent if it satisfies (2).*

► **Proposition 15.** *A category \mathbf{C} is Kleene-Kozen iff (\mathbf{C}, \mathbf{C}) is a *-idempotent KiC.*

Proof. As shown previously [16], \mathbf{C} is a Kleene-Kozen category iff

1. \mathbf{C} is enriched over bounded join-semilattices and strict join-preserving morphisms;
2. there is an operator $(-)^*: \mathbf{C}(X, X) \rightarrow \mathbf{C}(X, X)$ such that
 - a. $p^* = 1 + p; p^*$;
 - b. $1^* = 1$;
 - c. $p^* = (p + 1)^*$;
 - d. $u; p = q; u$ implies $u; p^* = q^*; u$.

This yields sufficiency by noting that (1) states precisely that all morphisms in \mathbf{C} are linear. To show necessity, it suffices to obtain (2.c) from the assumptions that (\mathbf{C}, \mathbf{C}) is a *-idempotent KiC and that all morphisms in \mathbf{C} are linear. Indeed, we have $(p+1)^* = 1^*$; $(p; 1^*)^* = p^*$. ◀

KiCs thus deviate from Kleene algebras precisely in four respects:

1. by generalizing from monoids to categories,
2. by allowing non-linear morphisms,
3. by dropping *-idempotence, and
4. by requiring binary coproducts.

► **Example 16.** The axiom ***-Sum**, included in Definition 13, is one of the classical *Conway identities*. The other one $(p; q)^* = 1 + p; (q; p)^*$; q is derivable if $\mathbf{C} = \bar{\mathbf{C}}$, e.g. in Kleene-Kozen categories. Indeed, $q; p; q = q; p; q$ entails $q; (p; q)^* = (q; p)^*; q$ by ***-Uni**, and using ***-Fix**, $(p; q)^* = 1 + p; q; (p; q)^* = 1 + p; (q; p)^*; q$.

Clearly, this argument remains valid with only q being tame, but otherwise the requisite identity is not provable.

It may not be obvious why the requirement to support binary coproducts is part of Definition 13, given that the axioms of iteration do not involve them. The reason is that certain identities that also do not involve coproducts are only derivable in their presence.

► **Example 17.** The identity $p^* = (p; (1 + p))^*$ holds in any KiC.

A standard way to instantiate Definition 13 is to start with a category \mathbf{V} with coproducts, and a monad \mathbf{T} on it, and take $\mathbf{C} = \mathbf{V}_{\mathbf{T}}$, $\bar{\mathbf{C}} = \mathbf{V}$ or, possibly, $\bar{\mathbf{C}} = \mathbf{V}_{\mathbf{T}}$. The monad must support nondeterminism and Kleene iteration so that the axioms of KiC are satisfied. Consider a class of Kleene-Kozen categories that arise in this way.

► **Example 18.** Let Q be a unital quantale, and let $TX = Q^X$ for every set X . Then T extends to a monad on **Set** as follows: $\eta(x)(x) = 1$, $\eta(x)(y) = \perp$ if $x \neq y$, and

$$(p: X \rightarrow Q^Y)^\sharp(f: X \rightarrow Q)(y \in Y) = \bigvee_{x \in X} p(x)(y) \cdot f(x).$$

We obtain a Kleene-Kozen structure in $\mathbf{Set}_{\mathbf{T}}$ as follows:

- $0: X \rightarrow Q^Y$ sends x to $\lambda y. \perp$;
- $p + q: X \rightarrow Q^Y$ sends x to $\lambda y. p(x)(y) \vee q(x)(y)$;
- $p^*: X \rightarrow Q^X$ is the least fixpoint of the map $q \mapsto 1 + q; p$.

This construction restricts to $Q_{\omega_1}^X = \{f: X \rightarrow Q \mid |\text{supp } f| \leq \omega\}$ where $\text{supp } f$ is the set of those $x \in X$, for which $f(x) \neq 0$. Thus, e.g. the Kleisli categories of the powerset monad \mathcal{P} and the countable powerset monad \mathcal{P}_{ω_1} are Kleene-Kozen.

For a contrast, consider a similar construction that yields a KiC, which is not Kleene-Kozen.

25:10 A Unifying Categorical View of Nondeterministic Iteration and Tests

► **Example 19.** Let $Q = \{0, 1, \infty\}$ be the complete lattice under the ordering $0 < 1 < \infty$, and let us define commutative binary multiplication as follows: $0 \cdot x = 0$, $1 \cdot x = x$ and $\infty \cdot \infty = \infty$. This turns Q into a unital quantale, hence an idempotent semiring, whose binary summation $+$ is binary join. Next, define infinite summation with the formula

$$\sum_{i \in I} x_i = \begin{cases} \bigvee_{i \in I'} x_i, & \text{if } I' = \{i \in I \mid x_i > 0\} \text{ is finite} \\ \infty, & \text{otherwise} \end{cases}$$

This makes Q into a *complete semiring* [11]. Let us define the monad \mathcal{R} and the idempotent grove structure on $\mathbf{Set}_{\mathcal{R}}$ like $Q_{\omega_1}^{(-)}$ in Example 18 (with \sum instead of \bigvee). For every $f: X \rightarrow \mathcal{R}X$, let $p^* = \sum_{n \in \mathbb{N}} p^n: X \rightarrow \mathcal{R}X$ where, inductively, $p^0 = 1$ and $p^{n+1} = p; p^n$, and infinite sums are extended from Q to the Kleisli hom-sets pointwise.

It is easy to verify that $(\mathbf{Set}_{\mathcal{R}}, \mathbf{Set}_{\mathcal{R}})$ is a KiC, but $\mathbf{Set}_{\mathcal{R}}$ is not a Kleene-Kozen category, for $*$ -idempotence fails: $\eta^* = \sum_{n \in \mathbb{N}} \eta^n = \sum_{n \in \mathbb{N}} \eta = \lambda x, y. \infty \neq \eta$. We will use a more convenient notation for the elements of $\mathcal{R}X$ as infinite formal sums $\sum_{i \in I} x_i$ ($x_i \in X$), modulo associativity, commutativity, idempotence (but without countable idempotence $\sum_{i \in I} x = x!$).

Below, we provide two results for constructing new KiCs from old: Theorem 20 and Theorem 23, which are also used prominently in our characterization result in Section 7.

► **Theorem 20.** *Let $(\mathbf{C}, \bar{\mathbf{C}})$ be a KiC and let \mathbf{T} be a monad on \mathbf{C} such that*

1. $T\tau^* = (Tr)^*$, for all $r \in \mathbf{C}(X, X)$;
2. the monad \mathbf{T} restricts to a monad on $\bar{\mathbf{C}}$.

Then $\bar{\mathbf{C}}_{\mathbf{T}}$ is a subcategory of $\bar{\mathbf{C}}$ and $(\mathbf{C}_{\mathbf{T}}, \bar{\mathbf{C}}_{\mathbf{T}})$ is a KiCT where 0 and $+$ are defined as in \mathbf{C} , and for any $p: X \rightarrow TX$, the corresponding Kleene iteration is computed as $\eta; (p^\sharp)^$.*

Let us illustrate the use of Theorem 20 by a simple example.

► **Example 21 (Finite Traces).** Consider the monad $\mathcal{P}(A^* \times -)$ on \mathbf{Set} . Elements of $\mathcal{P}(A^* \times X)$ are standardly used as (finite) trace semantics of programs. A trace is then a sequence of actions from A , followed by an end result in X . Of course, it can be verified directly that the Kleisli category of $\mathcal{P}(A^* \times -)$ is Kleene-Kozen. Let us show how this follows from Theorem 20.

The Kleisli category of \mathcal{P} is isomorphic to the category of relations, and is obviously Kleene-Kozen. For \mathcal{P} , like for any commutative monad, the Kleisli category $\mathbf{Set}_{\mathcal{P}}$ is symmetric monoidal: $X \otimes Y = X \times Y$ and, given $p: X \rightarrow \mathcal{P}Y$, $q: X' \rightarrow \mathcal{P}Y'$,

$$(p \otimes q)(x, x') = \{(y, y') \mid y \in p(x), y' \in q(x')\}.$$

The set A^* is a monoid in $\mathbf{Set}_{\mathcal{P}}$ w.r.t. this monoidal structure. This yields a writer monad \mathbf{T} on $\mathbf{Set}_{\mathcal{P}}$ via $TX = A^* \otimes X$ and $(Tp)(w, x) = \{(w, y) \mid y \in p(x)\}$. Its Kleisli category $(\mathbf{Set}_{\mathcal{P}})_{\mathbf{T}}$ is isomorphic to our original Kleisli category of interest. The assumptions (1) of Theorem 20 are thus satisfied in the obvious way. The assumption (2) is vacuous, as we chose all morphisms to be tame.

Finally, we establish robustness of KiCs under the *generalized coalgebraic resumption monad transformer* [32, 18], which is defined as follows.

► **Definition 22 (Coalgebraic Resumptions).** *Let \mathbf{V} be a category with coproducts and let \mathbf{T} be a monad on \mathbf{V} . Let $H: \mathbf{V} \rightarrow \mathbf{V}$ be some endofunctor and assume that all final coalgebras $\nu\gamma. T(X \oplus H\gamma)$ exist. The assignment $X \mapsto \nu\gamma. T(X \oplus H\gamma)$ yields a monad \mathbf{T}_H , called the (generalized) coalgebraic resumption monad transformer of \mathbf{T} .*

► **Theorem 23.** Let \mathbf{T}_H be as in Definition 22 and such that $(\mathbf{V}_{\mathbf{T}}, \bar{\mathbf{C}})$ is a KiC for some choice of $\bar{\mathbf{C}}$. Then $\mathbf{V}_{\mathbf{T}}$ is a wide subcategory of $\mathbf{V}_{\mathbf{T}_H}$ and $(\mathbf{V}_{\mathbf{T}_H}, \bar{\mathbf{C}})$ is a KiC w.r.t. the following structure:

- the bottom element in every $\mathbf{V}(X, T_H Y)$ is $0 ; \text{out}^{-1}$, and the join of $p, q \in \mathbf{V}(X, T_H Y)$ is $(p ; \text{out} + q ; \text{out}) ; \text{out}^{-1}$;
- given $p \in \mathbf{V}(X, T_H X)$, $p^* \in \mathbf{V}(X, T_H X)$ is the unique solution of the equation

$$p^* ; \text{out} = \text{in}_0 ; [p ; \text{out}, 0]^* ; T(1 \oplus H p^*).$$

We defer the proof to Section 6 where we use reduction to the existing result [18], using the equivalence of Kleene and Elgot iterations, we establish in Section 6.

► **Example 24.** By taking $T = \mathcal{P}_{\omega_1}$ and $H = A \times -$ in Theorem 23, we obtain $T_H X = \nu\gamma. \mathcal{P}_{\omega_1}(X \oplus A \times \gamma)$ from Example 2. Let us illustrate the effect of Kleene iteration by example. Consider the system of equations

$$P = a.P + Q, \quad Q = b.Q + P$$

for defining the behaviour of two processes P and Q . This system induces a function $p: \{P, Q\} \rightarrow T_H\{P, Q\}$, sending P to $a.P + Q$ and Q to $b.Q + P$. The expression $[p ; \text{out}, 0]^*$ calls the iteration operator of the powerset-monad, resulting in the function that sends both P and Q to $a.P + b.Q + Q + P$. Finally, p^* sends P to P' and Q to Q' , where P' and Q' are the synchronization trees, obtained as unique solutions of the system:

$$P' = a.P' + b.Q' + Q + P, \quad Q' = a.P' + b.Q' + Q + P.$$

6 Elgot Iteration and While-Loops

In this section, we establish an equivalence between Kleene iteration, in the sense of KiC and *Elgot iteration*, as an operation with the following profile in a category \mathbf{C} with coproducts:

$$(-)^\dagger: \mathbf{C}(X, Y \oplus X) \rightarrow \mathbf{C}(X, Y). \quad (7)$$

This could be done directly, but we prove an equivalence between Elgot iteration and while-loops first, and then prove the equivalence of the latter and Kleene iteration. In this chain of equivalences, only while-loops need tests, and it will follow that a particular choice of tests is not relevant, once they are expressive. On the other hand, existence of expressive tests is guaranteed by Lemma 10. This explains why tests disappear in the resulting equivalence.

► **Definition 25** (Conway Iteration, Uniformity). *An Elgot iteration operator (7) in a category \mathbf{C} with coproducts is Conway iteration [13] if it satisfies the following principles:*

$$\mathbf{Naturality} : p^\dagger ; q = (p ; (q \oplus 1))^\dagger \quad \mathbf{Dinaturality} : (p ; [\text{in}_0, q])^\dagger = p ; [1, (q ; [\text{in}_0, p])^\dagger]$$

$$\mathbf{Codiagonal} : (p ; [1, \text{in}_1])^\dagger = p^{\dagger\dagger}$$

Moreover, given a subcategory \mathbf{D} of \mathbf{C} , $(-)^\dagger$ is uniform w.r.t. \mathbf{D} , or \mathbf{D} -uniform, if it satisfies

$$\mathbf{Uniformity} : \frac{u ; q = p ; (1 \oplus u)}{u ; q^\dagger = p^\dagger} \quad (\text{with } u \text{ from } \mathbf{D})$$

By taking $q = \text{in}_1$ in **Dinaturality**, we derive

$$\mathbf{Fixpoint} : p ; [1, p^\dagger] = p^\dagger.$$

DW-Fix:	$\underline{\text{while } d \text{ do } p} = \underline{\text{if } d \text{ then } p} ; (\underline{\text{while } d \text{ do } p}) \text{ else } 1$
DW-Or:	$\underline{\text{while } (d \parallel e) \text{ do } p} = (\underline{\text{while } d \text{ do } p}) ; \underline{\text{while } e \text{ do } (p ; \underline{\text{while } d \text{ do } p})}$
DW-And:	$\underline{\text{while } (d \ \&\& \ (e \parallel \text{tt})) \text{ do } p} = \underline{\text{while } d \text{ do } (\underline{\text{if } e \text{ then } p \text{ else } p})}$
DW-Uni:	$\frac{u ; \underline{\text{if } d \text{ then } p} ; \text{tt else } \text{ff} = \underline{\text{if } e \text{ then } q} ; u ; \text{tt else } v ; \text{ff}}{u ; \underline{\text{while } d \text{ do } p} = (\underline{\text{while } e \text{ do } q}) ; v}$

■ **Figure 2** Uniform Conway iteration in terms of decisions.

► **Theorem 26.** *Let \mathbf{C} be a category with coproducts, let \mathbf{D} be its wide subcategory with coproducts, preserved by the inclusion, and let for every $X \in |\mathbf{C}|$, $\mathbf{C}^\diamond(X)$ be a set of decisions such that (i) $\text{in}_0, \text{in}_1 \in \mathbf{C}^\diamond(X)$, (ii) $\mathbf{C}^\diamond(X)$ is closed under (5), (iii) $\text{in}_0 \oplus \text{in}_1 \in \mathbf{C}^\diamond(X)$ if $X = X_1 \oplus X_2$. Then, to give a \mathbf{D} -uniform Conway iteration on \mathbf{C} is the same as to give an operator*

$$\frac{d \in \mathbf{C}^\diamond(X) \quad p \in \mathbf{C}(X, X)}{\underline{\text{while } d \text{ do } p} \in \mathbf{C}(X, X)}$$

that satisfies the laws in Figure 2 with p, q ranging over \mathbf{C} , and with u, v ranging over \mathbf{D} .

Theorem 26 yields an equivalence between two styles of iteration: Elgot iteration and while-iteration. We next specialize it to idempotent grove categories with tests using Lemma 11.

Note that in any $\text{KiC}(\mathbf{C}, \bar{\mathbf{C}})$ with tests $\mathbf{C}^?$, in addition to the if-then-else (6), we have the while operator, defined in the standard way: given $b \in \mathbf{C}^?(X)$, $p \in \mathbf{C}(X, X)$,

$$\text{while } b \text{ do } p = (b ; p)^* ; \bar{b}. \quad (8)$$

► **Proposition 27.** *Let \mathbf{C} be an idempotent grove category, let \mathbf{D} be a wide subcategory of \mathbf{C} with coproducts, which are preserved by the inclusion to \mathbf{C} , and with expressive tests $\mathbf{C}^?$. Then \mathbf{C} supports \mathbf{D} -uniform Conway iteration iff it supports a while-operator that satisfies the laws in Figure 3, where b and c come from $\mathbf{C}^?$, p and q come from \mathbf{C} and u, v come from \mathbf{D} and the if-then-else operator is defined as in (8).*

Proof. For every $X \in |\mathbf{C}|$, let $\mathbf{C}^\diamond(X)$ be the image of $\mathbf{C}^?(X)$ under \diamond from Lemma 11. As shown in the lemma, $\mathbf{C}^\diamond(X)$ inherits the Boolean algebra structure from $\mathbf{C}^?(X)$. Using the isomorphism between $\mathbf{C}^\diamond(X)$ and $\mathbf{C}^?(X)$ and Proposition 7, the laws from Figure 2 can be reformulated equivalently, resulting in **TW-Fix**, **TW-Or**, **TW-Uni**, and additionally

$$\text{while } (b \wedge (c \vee \top)) \text{ do } p = \text{while } b \text{ do } (\text{if } c \text{ then } p \text{ else } p)$$

which however holds trivially. ◀

Thus, in grove categories with expressive tests, Elgot iteration and while-loops are equivalent. We establish a similar equivalence between Kleene iteration and while-loops, which will entail an equivalence between Elgot iteration and Kleene iteration by transitivity.

► **Theorem 28.** *Let \mathbf{C} , $\bar{\mathbf{C}}$ and $\mathbf{C}^?$ be as follows.*

1. \mathbf{C} is an idempotent grove category with coproducts.
2. $\bar{\mathbf{C}}$ is a wide subcategory of \mathbf{C} with coproducts, consisting of linear morphisms only and such that the inclusion of $\bar{\mathbf{C}}$ to \mathbf{C} preserves coproducts.
3. $\mathbf{C}^?$ are expressive tests in \mathbf{C} .

Then $(\mathbf{C}, \bar{\mathbf{C}})$ is a KiCT iff \mathbf{C} supports a while-operator satisfying the laws in Figure 3.

TW-Fix:	$\text{while } b \text{ do } p = \text{if } b \text{ then } p ; (\text{while } b \text{ do } p) \text{ else } 1$
TW-Or:	$\text{while } (b \vee c) \text{ do } p = (\text{while } b \text{ do } p) ; \text{while } c \text{ do } (p ; \text{while } b \text{ do } p)$
TW-Uni:	$\frac{u ; \bar{b} = \bar{c} ; v \quad u ; b ; p = c ; q ; u}{u ; \text{while } b \text{ do } p = (\text{while } c \text{ do } q) ; v}$

■ **Figure 3** Uniform Conway iteration in terms of tests.

We can now characterize KiCs in terms of Elgot iteration.

► **Theorem 29.** *Let \mathbf{C} be an idempotent grove category with coproducts, and let $\bar{\mathbf{C}}$ be a wide subcategory of \mathbf{C} with coproducts, consisting of linear morphisms only and such that the inclusion of $\bar{\mathbf{C}}$ to \mathbf{C} preserves coproducts.*

Then $(\mathbf{C}, \bar{\mathbf{C}})$ is a KiC iff \mathbf{C} supports $\bar{\mathbf{C}}$ -uniform Conway iteration.

Proof. Let us define \mathbf{C}' as in Definition 9. By Theorem 28, $(\mathbf{C}, \bar{\mathbf{C}})$ is a KiCT iff \mathbf{C} supports a while-operator, satisfying the laws in Figure 3. By Proposition 27, the latter is the case iff \mathbf{C} supports $\bar{\mathbf{C}}$ -uniform Conway iteration. ◀

Now we can prove Theorem 23.

Proof Theorem 23 (Sketch). We need to check that $(\mathbf{V}_{\mathbf{T}_H}, \bar{\mathbf{C}})$ is a KiC. By Theorem 29, we equivalently prove that $\mathbf{V}_{\mathbf{T}_H}$ supports $\bar{\mathbf{C}}$ -uniform Conway iteration. It is already known [18, Lemma 7.2] that if \mathbf{T} supports Conway iteration, then so does \mathbf{T}_H . By Theorem 29, we are left to check that \mathbf{T}_H satisfies **Uniformity**, which is a matter of calculation. ◀

7 Free KiCTs and Completeness

In this section, we characterize a free KiCT with strict coproducts (i.e. those, for which coherence maps $X \oplus (Y \oplus Z) \cong (X \oplus Y) \oplus Z$ are identities) on a one-sorted signature. We achieve this by combining techniques from formal languages [7], category theory and the theory of Elgot iteration with coalgebraic reasoning [34], in particular proofs by coalgebraic bisimilarity. We claim that a more general characterization of a free KiCT on a multi-sorted signature can be achieved along the same lines, modulo a significant notation overhead and the necessity to form final coalgebras in the category of multisorted sets $\mathbf{Set}^{\mathcal{S}}$ where \mathcal{S} is the set of sorts. We dispense with this option for the sake of brevity and readability. Let us fix

- a signatures of n -ary symbols Σ_n for each $n \in \mathbb{N}$, and let $\Sigma = \bigcup_n \Sigma_n$;
- a signature Γ of (unary) symbols, disjoint from Σ ;
- a finite (!) signature Θ of (unary) symbols, disjoint from $\Sigma \cup \Gamma$.

Let $\hat{\Theta}$ denote the set of finite subsets of Θ . We regard Θ as a signature for tests, Γ as a signature for tame morphisms and Σ as a signature for general morphisms; $\hat{\Theta}$ is meant to capture finite conjunctions of the form $\mathbf{b}_1 \wedge \dots \wedge \mathbf{b}_n \wedge \bar{\mathbf{b}}_{n+1} \wedge \dots \wedge \bar{\mathbf{b}}_m$ as semantic correspondents of subsets $\{\mathbf{b}_1, \dots, \mathbf{b}_n\} \in \hat{\Theta}$, assuming an enumeration $\Theta = \{\mathbf{b}_1, \dots, \mathbf{b}_m\}$. This is inspired by Kleene algebra with tests [24]. Furthermore, we accommodate *guarded strings* from *op. cit.*: let Γ^Θ be the set of strings $\langle b_1, u_1, \dots, b_n, u_n, b_{n+1} \rangle$ with $u_i \in \Gamma$, $b_i \in \hat{\Theta}$.

7.1 Interpretations

An *interpretation* $\llbracket - \rrbracket$ of (Σ, Γ, Θ) over a KiCT $(\mathbf{C}, \bar{\mathbf{C}}, \mathbf{C}^?)$ is specified as follows:

$$\begin{aligned} \llbracket 1 \rrbracket &\in |\mathbf{C}| & \llbracket f \rrbracket &\in \mathbf{C}(\llbracket 1 \rrbracket, \llbracket n \rrbracket) & (f \in \Sigma_n) \\ \llbracket u \rrbracket &\in \bar{\mathbf{C}}(\llbracket 1 \rrbracket, \llbracket 1 \rrbracket) & (u \in \Gamma) & \llbracket b \rrbracket &\in \mathbf{C}^2(\llbracket 1 \rrbracket, \llbracket 1 \rrbracket) & (b \in \Theta) \end{aligned}$$

where $\llbracket n \rrbracket$ abbreviates the n -fold sum $\llbracket 1 \rrbracket \oplus \dots \oplus \llbracket 1 \rrbracket$. The latter immediately extends to $\hat{\Theta}$: $\llbracket \{ \} \rrbracket = 1$, $\llbracket \{b_1, \dots, b_n\} \rrbracket = b_1 ; \dots ; b_n ; \bar{b}_{n+1} ; \dots ; \bar{b}_m$, assuming that $\Theta = \{b_1, \dots, b_m\}$. Note that we interpret n -ary symbols over $\mathbf{C}(\llbracket 1 \rrbracket, \llbracket n \rrbracket) = \mathbf{C}^{\text{op}}(\llbracket 1 \rrbracket^n, \llbracket 1 \rrbracket)$. This equation seems to suggest that it could be more natural to use categories with products as models, rather than categories with coproducts. Our present choice helps us to treat generic KiCTs on the same footing with the free KiCT, which is defined in terms of coproducts and not products.

► **Definition 30** (Free KiCT). *A free KiCT w.r.t. (Σ, Γ, Θ) is a KiCT $(\mathfrak{F}_{\Sigma, \Gamma, \Theta}, \overline{\mathfrak{F}_{\Sigma, \Gamma, \Theta}}, \mathfrak{F}_{\Sigma, \Gamma, \Theta}^?)$ together with an interpretation of (Σ, Γ, Θ) in $\mathfrak{F}_{\Sigma, \Gamma, \Theta}$, such that for any other interpretation of (Σ, Γ, Θ) over a KiCT $(\mathbf{C}, \bar{\mathbf{C}}, \mathbf{C}^?)$, there is unique compatible functor from $\mathfrak{F}_{\Sigma, \Gamma, \Theta}$ to \mathbf{C} . More formally, for any interpretation $\llbracket - \rrbracket$, there is unique KiCT-functor $\llbracket - \rrbracket_{\dagger} : (\mathfrak{F}_{\Sigma, \Gamma, \Theta}, \overline{\mathfrak{F}_{\Sigma, \Gamma, \Theta}}, \mathfrak{F}_{\Sigma, \Gamma, \Theta}^?) \rightarrow (\mathbf{C}, \bar{\mathbf{C}}, \mathbf{C}^?)$ such that the diagram*

$$\begin{array}{ccc} \mathfrak{F}_{\Sigma, \Gamma, \Theta} & \xrightarrow{\llbracket - \rrbracket_{\dagger}} & \mathbf{C} \\ \llbracket - \rrbracket^{\mathfrak{s}} \uparrow & \nearrow \llbracket - \rrbracket & \\ (\Sigma, \Gamma, \Theta) & & \end{array} \quad (9)$$

commutes.

In what follows, we characterize $\mathfrak{F}_{\Sigma, \Gamma, \Theta}$ as a certain category of rational trees, i.e. trees with finitely many distinct subtrees. An alternative, equivalent formulation would be to view $\mathfrak{F}_{\Sigma, \Gamma, \Theta}$ as a free model of the (Lawvere) theory of KiCTs.

Like in the case of original Kozen's completeness result [23], a characterization of the free model immediately entails completeness of the corresponding axiomatization over it. Indeed, by generalities, a free KiCT is isomorphic to the free algebra of terms, quotiented by the provable equality relation. Hence, if an equality holds over the free model, it is provable.

7.2 A KiCT of Coalgebraic Resumptions

For any set X , define $TX = \mathcal{R}(\Gamma^{\Theta} \times X)$ and $T_{\nu}X = \nu\gamma.T(X \oplus \Sigma\gamma)$, in the category of sets \mathbf{Set} where \mathcal{R} is the monad from Example 19.

A stepping stone for constructing $\mathfrak{F}_{\Sigma, \Gamma, \Theta}$ is the observation that $(\mathbf{Set}_{\mathcal{R}_{\nu}}, \mathbf{Set}_{\mathcal{T}})$ forms a KiC. Indeed, $(\mathbf{Set}_{\mathcal{R}}, \mathbf{Set}_{\mathcal{R}})$ is a KiC and the monad \mathcal{R} is commutative, hence symmetric monoidal. In $\mathbf{Set}_{\mathcal{R}}$, Γ^{Θ} is a monoid under the following operations:

$$\langle w_1, \dots, w_{n+1} \rangle \cdot \langle u_1, \dots, u_{m+1} \rangle = \begin{cases} \langle w_1, \dots, w_n, u_2, \dots, u_{m+1} \rangle & \text{if } w_n = u_1 \\ 0 & \text{otherwise} \end{cases}$$

This produces the monad \mathbf{T} , whose Kleisli category is a KiC by Theorem 20, analogously to Example 21. Now, $(\mathbf{Set}_{\mathcal{T}_{\nu}}, \mathbf{Set}_{\mathcal{T}})$ is a KiC by Theorem 23. We will use the following representation for generic elements of $T_{\nu}X$:

$$t = \sum_{i \in I} b_i \cdot u_i \cdot t_i + \sum_{i \in J} b_i \cdot f_i(t_{i,1}, \dots, t_{i,n_i}) + \sum_{i \in K} b_i \cdot x_i \quad (10)$$

where I, J, K are mutually disjoint countable sets, b_i range over $\widehat{\Theta}$, u_i range over Γ , f_i range over Σ , $t_i, t_{i,j}$ range over $T_\nu X$ and x_i range over X .

► **Definition 31** (Derivatives). *For every $t \in T_\nu X$, as in (10), define the following derivative operations:*

- $\partial_{b,u}(t) = \sum_{i \in I, b=b_i, u=u_i} t_i$, for $b \in \widehat{\Theta}$, $u \in \Gamma$;
- $\partial_{b,f}^k(t) = \sum_{i \in J, b=b_i, f=f_i} t_{i,k}$, for $b \in \widehat{\Theta}$, $k \in \{1, \dots, n_i\}$, $f \in \Sigma_{n_i}$ with $n_i > 0$.

Additionally, let $o(t) = \sum_{i \in K} b_i \cdot x_i$. We extend these operations to arbitrary morphisms $Y \rightarrow T_\nu X$ pointwise. The set of derivatives of $t \in T_\nu X$ is the smallest set $\mathfrak{D}(t)$ that contains t and is closed under all $\partial_{b,u}$ and $\partial_{b,f}^k$.

The following property is a direct consequence of these definitions:

► **Lemma 32.** *Let $t \in T_\nu X$ be as in (10), and let $s: X \rightarrow T_\nu Y$. Then*

$$\begin{aligned} \partial_{b,u}(t; s^\sharp) &= \partial_{b,u}(t); s^\sharp + o(t); (\partial_{b,u}(s))^\sharp & o(t; s^\sharp) &= o(t); (o(s))^\sharp \\ \partial_{b,f}^k(t; s^\sharp) &= \partial_{b,f}^k(t); s^\sharp + o(t); (\partial_{b,f}^k(s))^\sharp \end{aligned}$$

► **Lemma 33.** *Given a set X , let $\mathcal{B} \subseteq T_\nu X \times T_\nu X$ be such a relation that whenever $t \mathcal{B} s$,*

1. $\partial_{b,u}(t) \mathcal{B} \partial_{b,u}(s)$ for all $b \in \widehat{\Theta}$, $u \in \Gamma$,
2. $\partial_{b,f}^k(t) \mathcal{B} \partial_{b,f}^k(s)$ for all $b \in \widehat{\Theta}$, $f \in \Sigma$,
3. $o(t) = o(s)$.

Then, $t = s$ whenever $t \mathcal{B} s$.

Proof Sketch. It suffices to show that \mathcal{B} is a coalgebraic bisimulation. The claim is then a consequence of strong extensionality of the final coalgebra $T_\nu X$. Let us spell out what it means for \mathcal{B} to be a coalgebraic bisimulation. Given t and t' , such that $t \mathcal{B} t'$, and assuming representations

$$\begin{aligned} t &= \sum_{i \in I} g_i \cdot f_i(t_{i,1}, \dots, t_{i,n_i}) + \sum_{i \in J} g_i \cdot x_i, \\ t' &= \sum_{i \in I'} g_i \cdot f_i(t_{i,1}, \dots, t_{i,n_i}) + \sum_{i \in J'} g_i \cdot x_i \end{aligned}$$

where the g_i range over Γ^Θ , the sums $\sum_{i \in J} g_i \cdot x_i$ and $\sum_{i \in J'} g_i \cdot x_i$ must be equal, and there must exist a set K and surjections $e: K \rightarrow I$, $e': K \rightarrow I'$, such that for every $k \in K$, $g_{e(k)} = g_{e'(k)}$, $f_{e(k)} = f_{e'(k)}$ and $t_{e(k),1} \mathcal{B} t_{e'(k),1}, \dots, t_{e(k),m} \mathcal{B} t_{e'(k),m}$ where m is the arity of $f_{e(k)}$. This is indeed true for \mathcal{B} . The reason for it is that t and t' can be represented as

$$\begin{aligned} t &= \sum_{n \in \mathbb{N}} \sum_{i \in I, |g_i|=n} g_i \cdot f_i(t_{i,1}, \dots, t_{i,n_i}) + \sum_{n \in \mathbb{N}} \sum_{i \in J, |g_i|=n} g_i \cdot x_i, \\ t' &= \sum_{n \in \mathbb{N}} \sum_{i \in I', |g_i|=n} g_i \cdot f_i(t'_{i,1}, \dots, t'_{i,n_i}) + \sum_{n \in \mathbb{N}} \sum_{i \in J', |g_i|=n} g_i \cdot x_i \end{aligned}$$

and we can derive the requisite properties for inner sums by induction on n from the assumptions. ◀

7.3 Rational Trees

In what follows, we identify every $n \in \mathbb{N}$ with the set $\{0, \dots, n-1\}$, and select binary coproducts in **Set** so that $n \oplus m = \{0, \dots, n-1\} \oplus \{0, \dots, m-1\} = \{0, \dots, n+m-1\} = n+m$. The inclusion of n to $m \geq n$ is then a coproduct injection, which we refer to as in_n^m .

► **Definition 34** (Prefinite, Flat, (Non-)Guarded, Rational, Definable).

1. The set of prefinite elements of $T_\nu X$ is defined by induction: $t \in T_\nu X$ of the form (10) is prefinite if the involved sums contain finitely many distinct elements and all the $t_i, t_{i,j} \in T_\nu X$ are prefinite.
2. A prefinite $t \in T_\nu X$ of the form (10) is flat if $t_i, t_{i,j} \in X$.
3. An element $t \in T_\nu X$ of the form (10) is guarded if $K = \emptyset$.
4. An element $t \in T_\nu X$ of the form (10) is non-guarded if $I \cup J = \emptyset$.
5. An element $t \in T_\nu X$ is rational if $\mathcal{D}(t)$ is finite and t depends on a finite subset of $\Sigma \cup \Gamma$. A map $t: Y \rightarrow T_\nu X$ is prefinite/flat/guarded/non-guarded if correspondingly for every $x \in X$, every $t(x)$ is prefinite/flat/guarded/non-guarded. Finally:
6. A map $t: k \rightarrow T_\nu n$ (with $k, n \in \mathbb{N}$) is definable if for some $m \geq k$ there is flat guarded $s: m \rightarrow T_\nu m$ and non-guarded $r: m \rightarrow T_\nu n$, such that $t = \text{in}_k^m; s^*; r^\sharp$.

Using Lemma 32, one can show

► **Lemma 35.** *Sum, composition and Kleene iteration of rational maps are again rational.*

The following property is a form of Kleene theorem, originally stating the equivalence of regular and recognizable languages [35]. In our setting it is proven with the help of Lemma 33.

► **Proposition 36.** *Given $n, k \in \mathbb{N}$, a map $n \rightarrow T_\nu k$ is rational iff it is definable.*

Let $\mathfrak{F} = \mathfrak{F}_{\Sigma, \Gamma, \Theta}$ be the (non-full) subcategory of \mathbf{Set}_{T_ν} , identified as follows:

- the objects of \mathfrak{F} are positive natural numbers,
- the morphisms in $\mathfrak{F}(n, k)$ are rational maps $f: n \rightarrow T_\nu k$ (equivalently: (co)tuples $[t_0, \dots, t_{n-1}]$ of rational elements of $T_\nu k$).

Let the wide subcategory of tame morphisms $\overline{\mathfrak{F}}$ consist of such tuples $[t_0, \dots, t_{n-1}]$ that $t_i \in Tk$ for all i , and let $\mathfrak{F}^?$ consist of those maps in $\overline{\mathfrak{F}}$ that do not involve symbols from Γ . This defines a KiCT essentially due to the closure properties from Lemma 35.

Given an interpretation $\llbracket - \rrbracket: (\Sigma, \Gamma, \Theta) \rightarrow \mathbf{C}$, let us extend it to flat elements first via

$$\llbracket 0 \rrbracket = 0, \quad \llbracket t + s \rrbracket = \llbracket t \rrbracket + \llbracket s \rrbracket, \quad \llbracket \eta^* \rrbracket = \llbracket 1 \rrbracket^*, \quad \llbracket t; s^\sharp \rrbracket = \llbracket t \rrbracket; \llbracket s \rrbracket, \quad \llbracket k \rrbracket = \text{in}_k \quad (k \in \mathbb{N})$$

where η^* stands for a tuple of infinite sum $[\sum_{i \in \mathbb{N}} \{ \cdot \}.0, \dots, \sum_{i \in \mathbb{N}} \{ \cdot \}.(n-1)]$, and is the interpretation of η^* in \mathfrak{F} . The clause for η^* is necessary to cater for infinite sums that can occur in prefinite elements. Such sums can only contain finitely many distinct elements, and thus can be expressed via finite sums and composition with η^* . Next, define $\llbracket - \rrbracket_\dagger: \mathfrak{F} \rightarrow \mathbf{C}$

- on objects via $\llbracket n \rrbracket_\dagger = \llbracket 1 \rrbracket \oplus \dots \oplus \llbracket 1 \rrbracket$ ($\llbracket 1 \rrbracket$ repeated n times),
- on morphisms, via $\llbracket t \rrbracket_\dagger = \text{in}_n^m; \llbracket s \rrbracket^*; \llbracket r \rrbracket$, where $t = \text{in}_n^m; s^*; r^\sharp$, for a guarded flat $s: m \rightarrow T_\nu m$, and a non-guarded $r: m \rightarrow T_\nu k$, computed with Proposition 36.

► **Theorem 37.** *$\mathfrak{F}_{\Sigma, \Gamma, \Theta}$ is a free KiCT over Σ, Γ, Θ .*

The following property is instrumental for proving this result:

► **Lemma 38.** *Let $(\mathbf{C}, \overline{\mathbf{C}})$ and $(\mathbf{D}, \overline{\mathbf{D}})$ be two KiCs, and let F be the following map, acting on objects and on morphisms: $FX \in |\mathbf{D}|$ for every $X \in |\mathbf{C}|$, $Fp \in \mathbf{D}(FX, FY)$ for every $p \in \mathbf{C}(X, Y)$. Suppose that F preserves coproducts, $Fp \in \overline{\mathbf{D}}(FX, FY)$ for all $p \in \overline{\mathbf{C}}(X, Y)$. F is a KiC-functor if the following further preservation properties hold*

$$Fp^* = (Fp)^*, \quad F(p; \text{in}_0) = Fp; \text{in}_0, \quad F(p; \text{in}_1) = Fp; \text{in}_1, \quad F(p; [0, 1]) = Fp; [0, 1].$$

Let us briefly outline a potential application of Theorem 37 to may-diverge Kleene algebras, which we informally described in the introduction. Let us now define them formally:

► **Definition 39** (May-Diverge Kleene Algebra). *A may-diverge Kleene algebra is an idempotent semiring $(S, 0, 1, +, ;)$ equipped with an iteration operator $(-)^*: S \rightarrow S$ satisfying the laws:*

$$p^* = 1 + p ; p^* \quad (p + q)^* = p^* ; (q ; p^*)^* \quad \frac{r ; p = q ; r}{r ; p^* = q^* ; r}$$

Thus, may-diverge Kleene algebras are very close to KiCs of the form (\mathbf{C}, \mathbf{C}) with $|\mathbf{C}| = 1$, except that in our present treatment all KiCs come with binary coproducts as an additional structure. We conjecture though that any may-diverge Kleene algebra, viewed as a category, can be embedded to a KiC (\mathbf{C}, \mathbf{C}) with $|\mathbf{C}| = \{1, 2, \dots\}$. Theorem 37 will then entail a characterization of the free may-diverge Kleene algebra on Γ as the full subcategory induced by the single object 1 of the Kleisli category of the monad $TX = \mathcal{R}(\Gamma^* \times X)$. In other words, the free may-diverge Kleene algebra is carried (up-to-isomorphism) by rational elements of $\mathcal{R}(\Gamma^*)$, similarly to that how the free Kleene algebra is carried by rational elements of $\mathcal{P}(\Gamma^*)$.

8 Conclusions and Further Work

We developed a general and robust categorical notion of Kleene iteration – KiC(T) (=Kleene-iteration category (with tests)) – inspired by Kleene algebra (with tests) and its numerous cousins. We attested this notion with various yardsticks: stability under the generalized coalgebraic resumption monad transformer (hence under the exception transformer, as its degenerate case), equivalence to the classical notion of Conway iteration and to a suitably axiomatized theory of while-loops, but most remarkably, we established an explicit description of the ensuing free model, as a category of certain nondeterministic rational trees, playing the same role for our theory as the algebra of regular events for Kleene algebra. However, in our case, the free model is much more intricate and difficult to construct, as the iteration operator of it is neither a least fixpoint nor a unique fixpoint. A salient feature of our notion, mirrored in the structure of the free model, is that it can mediate between linear time and branching time semantics via corresponding specified classes of morphisms.

Given the abstract nature of our results, we expect them to be reusable for varying and enriching the core notion of Kleene iteration with other features. For example, our underlying notion of nondeterminism is that of idempotent grove category. General grove categories are a natural base for probabilistic or graded semantics, and we expect that most of our results, including completeness can be adapted to this case. Yet more generally, a relevant ingredient of our construction is monad \mathcal{R} , currently capturing the effect of nondeterminism, but which can potentially be varied to obtain other flavors of linear behavior.

An important open problem that remains for future work is that of defining KiCTs without coproducts, potentially providing a bridge to relevant algebraic structures as single-object categories. Now that the free KiCT with coproduct is identified, the free KiCT without coproducts is expected to be complete over the same model. Identifying such a notion is hard, because it would simultaneously encompass independent axiomatizations of iterative behavior, e.g. branching time and linear time. As of now, such axiomatizations are built on hard-to-reconcile approaches to iteration as either a least or a unique fixpoint.

References

- 1 Luca Aceto, Arnaud Carayol, Zoltán Ésik, and Anna Ingólfssdóttir. Algebraic synchronization trees and processes. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Proc. of 39th Int. Coll on Automata, Languages, and Programming, ICALP 2012, Part 2*, volume 7392 of *Lect. Notes in Comput. Sci.*, pages 30–41. Springer, 2012. doi: 10.1007/978-3-642-31585-5_7.

- 2 Michael A. Arbib and Ernest G. Manes. Partially additive categories and flow-diagram semantics. *J. Algebra*, 62(1):203–227, 1980. doi:10.1016/0021-8693(80)90212-4.
- 3 Steve Awodey. *Category Theory*. Oxford University Press, 2nd edition, 2010.
- 4 David B. Benson and Jerzy Tiuryn. Fixed points in free process algebras, part I. *Theor. Comput. Sci.*, 63(3):275–294, 1989. doi:10.1016/0304-3975(89)90010-8.
- 5 S.L. Bloom, Z. Esik, and D. Taubner. Iteration theories of synchronization trees. *Information and Computation*, 102(1):1–55, 1993. doi:10.1006/inco.1993.1001.
- 6 Stephen L. Bloom and Zoltán Ésik. *Iteration Theories: The Equational Logic of Iterative Processes*. Springer, 1993. doi:10.1007/978-3-642-78034-9.
- 7 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.
- 8 Kenta Cho. Total and partial computation in categorical quantum foundations. In Chris Heunen, Peter Selinger, and Jamie Vicary, editors, *Proc. of 12th Int. Workshop on Quantum Physics and Logic, QPL 2015*, volume 195 of *Electron. Proc. in Theor. Comput. Sci.*, pages 116–135. Open Publishing Assoc., 2015. doi:10.4204/eptcs.195.9.
- 9 Robin Cockett. Itegories & PCAs, 2007. Slides from Fields Institute Meeting on Traces (Ottawa, 2007). URL: <https://pages.cpsc.ucalgary.ca/~robin/talks/itegory.pdf>.
- 10 Robin Cockett and Stephen Lack. Restriction categories III: colimits, partial limits and extensivity. *Math. Struct. Comput. Sci.*, 17(4):775–817, 2007. doi:10.1017/s0960129507006056.
- 11 Manfred Droste and Werner Kuich. Semirings and formal power series. In *Handbook of Weighted Automata*, pages 3–28. Springer, 2009. doi:10.1007/978-3-642-01492-5_1.
- 12 Calvin Elgot. Monadic computation and iterative algebraic theories. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium 1973*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 175–230. Elsevier, 1975. doi:10.1016/s0049-237x(08)71949-9.
- 13 Zoltán Ésik. Equational properties of fixed-point operations in cartesian categories: An overview. *Math. Struct. Comput. Sci.*, 29(6):909–925, 2019. doi:10.1017/s0960129518000361.
- 14 Wan J. Fokkink and Hans Zantema. Basic process algebra with iteration: Completeness of its equational axioms. *Comput. J.*, 37(4):259–268, 1994. doi:10.1093/comjnl/37.4.259.
- 15 Leandro Gomes, Alexandre Madeira, and Luís S. Barbosa. On Kleene algebras for weighted computation. In Simone Cavalheiro and José Fiadeiro, editors, *Proc. of 20th Brazilian Symp. on Formal Methods, SBMF 2017*, volume 10623 of *Lect. Notes in Comput. Sci.*, pages 271–286. Springer, 2017. doi:10.1007/978-3-319-70848-5_17.
- 16 Sergey Goncharov. Shades of iteration: From Elgot to Kleene. In Alexandre Madeira and Manuel A. Martins, editors, *Revised Selected Papers from 26th IFIP WG 1.3 Int. Workshop on Recent Trends in Algebraic Development Techniques, WADT 2022*, volume 13710 of *Lect. Notes in Comput. Sci.*, pages 100–120. Springer, 2023. doi:10.1007/978-3-031-43345-0_5.
- 17 Sergey Goncharov, Lutz Schröder, and Till Mossakowski. Kleene monads: Handling iteration in a framework of generic effects. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Proc. of 3rd Int. Conf. Algebra and Coalgebra in Computer Science, CALCO 2009*, volume 5728 of *Lect. Notes in Comput. Sci.*, pages 18–33. Springer, 2009. doi:10.1007/978-3-642-03741-2_3.
- 18 Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Julian Jakob. Unguarded recursion on coinductive resumptions. *Log. Methods in Comput. Sci.*, 14(3):10:1–10:47, 2018. doi:10.23638/lmcs-14(3:10)2018.
- 19 Clemens Grabmayer. Milner’s proof system for regular expressions modulo bisimilarity is complete: Crystallization: Near-collapsing process graph interpretations of regular expressions. In *Proc. of 37th Ann. ACM/IEEE Symp. on Logic in Computer Science, LICS ’22*, pages 34:1–34:13, New York, 2022. doi:10.1145/3531130.3532430.
- 20 Niels Grathwohl, Dexter Kozen, and Konstantinos Mamouras. KAT + B! In *Proc. of 23rd EACL Ann. Conf. on Computer Science Logic and 29th Ann. ACM/IEEE Symp. on Logic in Computer Science, CSL-LICS 2014*, pages 44:1–44:10. ACM, 2014. doi:10.1145/2603088.2603095.

- 21 Bart Jacobs. New directions in categorical logic, for classical, probabilistic and quantum logic. *Log. Methods Comput. Sci.*, 11(3):24:1–24:76, 2015. doi:10.2168/lmcs-11(3:24)2015.
- 22 S. C. Kleene. Representation of events in nerve nets and finite automata. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.
- 23 Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.
- 24 Dexter Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, 1997. doi:10.1145/256167.256195.
- 25 Dexter Kozen and Konstantinos Mamouras. Kleene algebra with products and iteration theories. In Simona Ronchi Della Rocca, editor, *Proc. of 22nd EACSL Ann. Conf. on Computer Science Logic, CSL 2013*, volume 23 of *Leibniz Int. Proc. in Inform.*, pages 415–431. Dagstuhl Publishing, 2013. doi:10.4230/lipics.csl.2013.415.
- 26 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1971.
- 27 Annabelle McIver, Tahiry M. Rabehaja, and Georg Struth. On probabilistic Kleene algebras, automata and simulations. In Harrie de Swart, editor, *Proc. of 12th Int. Conf. on Relational and Algebraic Methods in Computer Science, RAMICS 2011*, volume 6663 of *Lect. Notes in Comput. Sci.*, pages 264–279. Springer, 2011. doi:10.1007/978-3-642-21070-9_20.
- 28 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Comput. Sci.* Springer, 1980. doi:10.1007/3-540-10235-3.
- 29 Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984. doi:10.1016/0022-0000(84)90023-0.
- 30 Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.
- 31 Bernhard Möller. Kleene getting lazy. *Sci. Comput. Program.*, 65(2):195–214, 2007. doi:10.1016/j.scico.2006.01.010.
- 32 Maciej Piróg and Jeremy Gibbons. Monads for behaviour. *Electron. Notes Theor. Comput. Sci.*, 298:309–324, 2013. doi:10.1016/j.entcs.2013.09.019.
- 33 Gordon D. Plotkin. A powerdomain construction. *SIAM J. Comput.*, 5(3):452–487, 1976. doi:10.1137/0205035.
- 34 J.J.M.M. Rutten. Behavioural differential equations: A coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1):1–53, 2003. doi:10.1016/S0304-3975(02)00895-2.
- 35 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
- 36 Alex Simpson and Gordon Plotkin. Complete axioms for categorical fixed-point operators. In *Proc. of 15th Ann. IEEE Symp. on Logic in Computer Science, LICS '00*, pages 30–41. IEEE, 2000. doi:10.1109/lics.2000.855753.
- 37 Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: Verification of uninterpreted programs in nearly linear time. *Proc. ACM Program. Lang.*, 4(POPL), 2020. doi:10.1145/3371129.
- 38 Daniele Turi and Jan Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Math. Struct. Comput. Sci.*, 8(5):481–540, 1998. doi:10.1017/s0960129598002588.
- 39 Tarmo Uustalu. Generalizing substitution. *Theor. Inform. Appl.*, 37(4):315–336, 2003. doi:10.1051/ita:2003022.

A Selected Proof Details

Proof of Proposition 8. The necessity is obvious. Let us show sufficiency. For every $b \in \mathbf{C}^?(X)$, let us fix some choice of $\bar{b} \in \mathbf{C}^?(X)$, for which the pair (b, \bar{b}) satisfies contradiction and excluded middle, and show that $\mathbf{C}^?(X)$ forms a Boolean algebra, i.e. $\mathbf{C}^?(X)$ is a complemented distributive lattice. Complementation amounts to the assumed identities, and we are left to

25:20 A Unifying Categorical View of Nondeterministic Iteration and Tests

show the laws of distributive lattices. Since complements are uniquely defined in Boolean algebras, it will follow that \bar{b} is uniquely determined by b . Of course, this is not used in the subsequent proof.

It follows by definition that $(\mathbf{C}^?(X), 0, +)$ and $(\mathbf{C}^?(X), 1, ;)$ are monoids. Showing that they are idempotent and commutative (hence, are semilattices) amounts to showing that $b; b = b$ and $b; c = c; b$ for all $b, c \in \mathbf{C}^?(X)$. The first identity is shown as follows, using linearity and the assumed identities:

$$b = b; 1 = b; (b + \bar{b}) = b; b + b; \bar{b} = b; b + 0 = b; b.$$

For the second one, note that $1 = b + \bar{b} = b + b + \bar{b} = b + 1$, and then

$$\begin{aligned} b; c = b; c; 1 = b; c; (b + 1) &= b; c; b + b; c \\ &= b; c; b + b; c; b; c = b; c; b; (1 + c) = b; c; b, \end{aligned}$$

where we used the instance of idempotence $b; c = b; c; b; c$ that we just established. Analogously, $c; b = b; c; b$, and hence $b; c = c; b$.

Finally, distributivity amounts to $(a + b); c = a; c + b; c$ and $a + b; c = (a + b); (a + c)$ for all $a, b, c \in \mathbf{C}^?(X)$. The first identity is an axiom of idempotent grove categories. The second one is obtained as follows:

$$\begin{aligned} (a + b); (a + c) &= a; a + a; c + b; a + b; c = a + a; c + a; b + b; c \\ &= a; (1 + c + b) + b; c = a; 1 + b; c = a + b; c \end{aligned} \quad \blacktriangleleft$$

Proof of Lemma 10. Negation is defined as follows:

$$\begin{aligned} \bar{0} &= 1, & \overline{[\text{in}_0, 0]} &= [0, \text{in}_1], & \overline{\bar{b} + c} &= \bar{b}; \bar{c}, \\ \bar{1} &= 0, & \overline{[0, \text{in}_1]} &= [\text{in}_0, 0], & \overline{\bar{b}; c} &= \bar{b} + \bar{c}. \end{aligned}$$

For every X , let $\mathbf{C}^?(X)$ be the smallest subset of $\mathbf{C}(X, X)$ that contains $0, 1, [\text{in}_0, 0]$ and $[0, \text{in}_1]$, and closed under $+$ and $;$. By Proposition 8, we need to show that every $b \in \mathbf{C}^?(X)$ is linear and satisfies $b; \bar{b} = 0, b + \bar{b} = 1$, which we do by induction. Let us strengthen the induction invariant by also adding $\bar{b}; b = 0$. Note that the above equations do not uniquely define complement, e.g. $\overline{[\text{in}_0, 0]} = [0, \text{in}_1]$ refers to a particular decomposition of X as $X_1 \oplus X_2$, while another decomposition could theoretically produce a different result. Thus, more precisely, we use the fact that every element of $\mathbf{C}^?(X)$ has a representation in the free algebra of terms over $0, 1, [\text{in}_0, 0], [0, \text{in}_1], +$ and $;$. The claim is then obtained by induction over this representation. \blacktriangleleft

Proof of Lemma 35. The dependency condition is obvious in all three cases. We will prove finiteness of sets of derivatives only.

Sum. Let $t, s \in T_\nu n$ be rational. Then $\mathfrak{D}(t + s) = \{t + s\} \cup \mathfrak{D}(t) \cup \mathfrak{D}(s)$, which is finite, since $\mathfrak{D}(t)$ and $\mathfrak{D}(s)$ are so.

Composition. It suffices to stick to the following instance: given $n, k \in \mathbb{N}$, a rational element $t \in T_\nu n$ and a rational map $s: n \rightarrow T_\nu k$, show that $t; s^\sharp \in T_\nu k$ is rational. Consider the set P of sums of the form

$$t'; s^\sharp + \sum_{s' \in \mathfrak{D}(s)} r_{s'}; (s')^\sharp$$

where t' ranges over $\mathfrak{D}(t)$ and $r_{s'}$ range over non-guarded elements of $T_\nu n$. Then P is finite. Moreover, P is closed under derivatives: using Lemma 32,

$$\begin{aligned}
& \partial_{b,u}(t'; s^\sharp + \sum_{s' \in \mathfrak{D}(s)} r_{s'}; (s')^\sharp) \\
&= \partial_{b,u}(t'); s^\sharp + o(t'); (\partial_{b,u}(s))^\sharp + \sum_{s' \in \mathfrak{D}(s)} \partial_{b,u}(r_{s'}); (s')^\sharp \\
&\quad + \sum_{s' \in \mathfrak{D}(s)} o(r_{s'}); (\partial_{b,u}(s'))^\sharp \\
&= \partial_{b,u}(t'); s^\sharp + o(t'); (\partial_{b,u}(s))^\sharp + \sum_{s' \in \mathfrak{D}(s)} o(r_{s'}); (\partial_{b,u}(s'))^\sharp \\
&= \partial_{b,u}(t'); s^\sharp + (o(t') + o(r_s)); (\partial_{b,u}(s))^\sharp + \sum_{s' \in \mathfrak{D}(s) \setminus \{s\}} o(r_{s'}); (\partial_{b,u}(s'))^\sharp,
\end{aligned}$$

and analogously for $\partial_{b,f}^k$. Note that $t; s^\sharp \in P$. Therefore $\mathfrak{D}(t; s^\sharp) \subseteq P$. Since P is finite, so is $\mathfrak{D}(t; s^\sharp)$.

Iteration. Let $t = [t_0, \dots, t_{n-1}] : n \rightarrow T_\nu n$, and $\mathfrak{D}(t_i)$ be finite for $i = 0, \dots, n-1$. Analogously to the previous clause, consider the set P of sums of the form

$$\sum_{t' \in \mathfrak{D}(t)} r_{t'}; (t')^\sharp; (t^*)^\sharp + r$$

where t' ranges over $\mathfrak{D}(t)$ and $r_{s'}, r$ range over those non-guarded elements of $T_\nu n$. In the same manner as in the previous clause: P is finite, contains t^* and is closed under derivatives, hence $\mathfrak{D}(t^*)$ is finite. \blacktriangleleft

Proof of Theorem 37. The key observation is that $\llbracket \text{in}_n^m; s^*; r^\sharp \rrbracket_\uparrow$ does not depend on the choice of s and r . This is argued as follows. Using the construction in Proposition 36, for a given $t = \text{in}_n^m; s^*; r^\sharp$, we obtain a canonical representation $t = \text{in}_n^l; \widehat{s}^*; \widehat{r}^\sharp$, with $\widehat{s} : l \rightarrow T_\nu l$, $\widehat{r} : l \rightarrow T_\nu k$, and this representation only depends on t , hence, it suffices to show that

$$\llbracket \text{in}_n^m; s^*; r^\sharp \rrbracket_\uparrow = \llbracket \text{in}_n^l; \widehat{s}^*; \widehat{r}^\sharp \rrbracket_\uparrow. \quad (11)$$

Because of the restrictions on s and r , there is an epimorphism $u : m \rightarrow l$, such that $s; T_\nu u = u; \widehat{s}$ and $u; \widehat{r} = r$. W.l.o.g. assume that $\text{in}_{l,m}$ is a left inverse of u . Now, (11) is obtained as follows:

$$\begin{aligned}
\llbracket \text{in}_n^m; s^*; r^\sharp \rrbracket_\uparrow &= \text{in}_n^m; \llbracket s \rrbracket^*; \llbracket r \rrbracket \\
&= \text{in}_n^m; \llbracket s \rrbracket^*; \llbracket u \rrbracket; \llbracket \widehat{r} \rrbracket \\
&= \text{in}_n^m; u; \llbracket \widehat{s} \rrbracket^*; \llbracket \widehat{r} \rrbracket && // \text{* -Uni} \\
&= \text{in}_n^l; \text{in}_{l,m}; u; \llbracket \widehat{s} \rrbracket^*; \llbracket \widehat{r} \rrbracket \\
&= \text{in}_n^l; \llbracket \widehat{s} \rrbracket^*; \llbracket \widehat{r} \rrbracket \\
&= \llbracket \text{in}_n^l; \widehat{s}^*; \widehat{r}^\sharp \rrbracket_\uparrow
\end{aligned}$$

The defined lifting $\llbracket - \rrbracket_\uparrow : \mathfrak{F} \rightarrow \mathbf{C}$ is easily seen to make (9) commute. Also, note that there is no more than one structure-preserving candidate for $\llbracket - \rrbracket_\uparrow$, to make (9) commute: indeed, since every morphism in \mathfrak{F} is representable as $t = \text{in}_n^m; s^*; r^\sharp$, $\llbracket t \rrbracket_\uparrow$ must only be defined as $\text{in}_n^m; \llbracket s \rrbracket^*; \llbracket r \rrbracket$.

We are left to check that $\llbracket - \rrbracket_\uparrow$ is a KiCT-functor, which is facilitated by Lemma 38. The only non-trivial clause is preservation of Kleene star. As an auxiliary step, we show that

$$\llbracket \eta + \text{in}_n^m; s^*; r^\sharp \rrbracket_\uparrow = \llbracket \eta \rrbracket_\uparrow + \llbracket \text{in}_n^m; s^*; r^\sharp \rrbracket_\uparrow \quad (12)$$

25:22 A Unifying Categorical View of Nondeterministic Iteration and Tests

for any guarded flat $s : m \rightarrow T_\nu m$, and a non-guarded $r : m \rightarrow T_\nu n$. In order to calculate the left-hand side of (12), we need to find a suitable representation for $1 + \text{in}_n^m ; s^* ; r^\sharp$. Concretely, we show that

$$\eta + \text{in}_n^m ; s^* ; r^\sharp = \text{in}_n^{m+m} ; [\text{in}_1 ; \eta, s ; T_\nu \text{in}_1]^* ; [[\eta_n, 0], r]^\sharp$$

Indeed, using ***-Fix** and ***-Uni**,

$$\begin{aligned} & \text{in}_n^{m+m} ; [\text{in}_1 ; \eta, s ; T_\nu \text{in}_1]^* ; [[\eta_n, 0], r]^\sharp \\ &= \text{in}_n^{m+m} ; (\eta + [\text{in}_1 ; \eta, s ; T_\nu \text{in}_1] ; ([\text{in}_1 ; \eta, s ; T_\nu \text{in}_1]^*)^\sharp) ; [[\eta_n, 0], r]^\sharp \\ &= \eta + \text{in}_n^m ; \text{in}_1 ; \eta ; ([\text{in}_1 ; \eta, s ; T_\nu \text{in}_1]^*)^\sharp ; [[\eta_n, 0], r]^\sharp \\ &= \eta + \text{in}_n^m ; s^* ; T_\nu \text{in}_1 ; [[\eta_n, 0], r]^\sharp \\ &= \eta + \text{in}_n^m ; s^* ; r^\sharp. \end{aligned}$$

Now, (12) turns into

$$\text{in}_n^{m+m} ; [\text{in}_1, \llbracket s \rrbracket ; \text{in}_1]^* ; [[1_n, 0], \llbracket r \rrbracket] = 1 + \text{in}_n^m ; \llbracket s \rrbracket^* ; \llbracket r \rrbracket.$$

This equation is shown as above, since ***-Fix** and ***-Uni** are sound for **C**.

An analogous method is used to show that $\llbracket - \rrbracket_\uparrow$ preserves Kleene star. Let $s : m \rightarrow T_\nu m$ be guarded flat, and let $r : m \rightarrow T_\nu n$ be non-guarded, and prove that:

$$\llbracket (\text{in}_n^m ; s^* ; r^\sharp)^* \rrbracket_\uparrow = \llbracket \text{in}_n^m ; s^* ; r^\sharp \rrbracket_\uparrow^*. \quad (13)$$

The following equation is provable using the axioms of KiC

$$(\text{in}_n^m ; s^* ; r^\sharp)^* = \eta + \text{in}_n^m ; ((r ; T_\nu \text{in}_n^m)^* ; s^\sharp)^* ; ((r ; T_\nu \text{in}_n^m)^* ; r^\sharp)^\sharp,$$

hence, the equation

$$(\text{in}_n^m ; \llbracket s \rrbracket^* ; \llbracket r \rrbracket^\sharp)^* = 1 + \text{in}_n^m ; ((\llbracket r \rrbracket ; \text{in}_n^m)^* ; \llbracket s \rrbracket^*)^* ; (\llbracket r \rrbracket ; \text{in}_n^m)^* ; \llbracket r \rrbracket$$

is provable as well. Now, the proof of (13) is as follows:

$$\begin{aligned} \llbracket (\text{in}_n^m ; s^* ; r^\sharp)^* \rrbracket_\uparrow &= \llbracket \eta + \text{in}_n^m ; ((r ; T_\nu \text{in}_n^m)^* ; s^\sharp)^* ; ((r ; T_\nu \text{in}_n^m)^* ; r^\sharp)^\sharp \rrbracket_\uparrow \\ &= 1 + \text{in}_n^m ; ((\llbracket r \rrbracket ; \text{in}_n^m)^* ; \llbracket s \rrbracket^\sharp)^* ; (\llbracket r \rrbracket ; \text{in}_n^m)^* ; \llbracket r \rrbracket \quad // (12) \\ &= (\text{in}_n^m ; \llbracket s \rrbracket^* ; \llbracket r \rrbracket^\sharp)^* \\ &= \llbracket \text{in}_n^m ; s^* ; r^\sharp \rrbracket_\uparrow^*. \quad \blacktriangleleft \end{aligned}$$



Phase-Bounded Broadcast Networks over Topologies of Communication

Lucie Guillou  

IRIF, CNRS, Université Paris Cité, France

Arnaud Sangnier  

DIBRIS, Università di Genova, Italy

Nathalie Sznajder  

LIP6, CNRS, Sorbonne Université, Paris, France

Abstract

We study networks of processes that all execute the same finite state protocol and that communicate through broadcasts. The processes are organized in a graph (a *topology*) and only the neighbors of a process in this graph can receive its broadcasts. The coverability problem asks, given a protocol and a state of the protocol, whether there is a topology for the processes such that one of them (at least) reaches the given state. This problem is undecidable [6]. We study here an under-approximation of the problem where processes alternate a bounded number of times k between phases of broadcasting and phases of receiving messages. We show that, if the problem remains undecidable when k is greater than 6, it becomes decidable for $k = 2$, and EXPSpace-complete for $k = 1$. Furthermore, we show that if we restrict ourselves to line topologies, the problem is in P for $k = 1$ and $k = 2$.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Parameterized verification, Coverability, Broadcast Networks

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.26

Related Version *Long Version*: <https://arxiv.org/abs/2406.15202> [15]

Funding *Lucie Guillou*: ANR project PaVeDyS (ANR-23-CE48-0005)

1 Introduction

Verifying networks with an unbounded number of entities. Ensuring safety properties for concurrent and distributed systems is a challenging task, since all possible interleavings must be taken into account; hence, even if each entity has a finite state behavior, the verification procedure has to deal with the state explosion problem. Another level of difficulty arises when dealing with distributed protocols designed for an unbounded number of entities. In that case, the safety verification problem consists in ensuring the safety of the system, for any number of participants. Here, the difficulty comes from the infinite number of possible instantiations of the network. In their seminal paper [13], German and Sistla propose a formal model to represent and analyze such networks: in this work, all the processes in the network execute the same protocol, given by a finite state automaton, and they communicate thanks to pairwise synchronized rendez-vous. The authors study the parameterized coverability problem, which asks whether there exists an initial number of processes that allow an execution leading to a configuration in which (at least) one process is in an error state (here the parameter is the number of processes). They show that it is decidable in polynomial time. Later on, different variations of this model have been considered, by modifying the communication means: token-passing mechanism [1, 5], communication through shared register [8, 11], non-blocking rendez-vous mechanism [14], or adding a broadcast mechanism to send a message to all the



© Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 26; pp. 26:1–26:16



Leibniz International Proceedings in Informatics

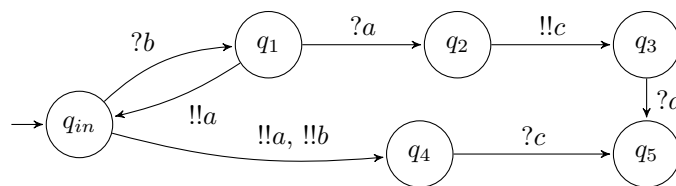
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

entities [9]. The model of population protocol proposed in [2] and for which verification methods have been developed recently in [10,12] belongs also to this family of systems. In this latter model, the properties studied are different, and more complex than safety conditions.

Broadcast networks working over graphs. In [6], Delzanno et. al propose a new model of parameterized network in which each process communicates *with its neighbors* by broadcasting messages. The neighbors of an entity are given thanks to a graph: the *communication topology*. This model was inspired by ad hoc networks, where nodes communicate with each other thanks to radio communication. The difficulty in proving safety properties for this new model lies in the fact that one has to show that the network is safe for all possible numbers of processes and all possible communication topologies. So the verification procedure not only looks for the number of entities, but also for a graph representing the relationship of the neighbours to show unsafe execution. As mentioned earlier, it is not the first work to propose a parameterized network with broadcast communication; indeed the parameterized coverability problem in networks with broadcast is decidable [9] and non-primitive recursive [24] when the communication topology is complete (each entity is a neighbor of all the others). However, when there is no restriction on the allowed communication topologies the problem becomes undecidable [6] but decidability can be regained by providing a bound on the length of all simple paths in allowed topologies [6]. This restriction has then been extended in [7] to allow also cliques in the model. However, with this restriction, the complexity of parameterized coverability is non-primitive recursive [7].

Bounding the number of phases. When dealing with infinite-state systems with an undecidable safety verification problem, one option consists in looking at under-approximations of the global behavior, restricting the attention to a subset of executions. If proving whether the considered subset of executions is safe is a decidable problem, this technique leads to a sound but incomplete method for safety verification. Good under-approximation candidates are the ones that can be extended automatically to increase the allowed behavior. For instance, it is known that safety verification of finite systems equipped with integer variables that can be incremented, decremented, or tested to zero is undecidable [19], but if one considers only executions in which, for each counter, the number of times the execution alternates between an increasing mode and a decreasing mode is bounded by a given value, then safety verification becomes decidable [16]. Similarly, verifying concurrent programs manipulating stacks is undecidable [22] but decidability can be regained by bounding the number of allowed context switches (a context being a consecutive sequence of transitions performed by the same thread) [20]. Context-bounded analysis has also been applied to concurrent programs with stacks and dynamic creation of threads [3]. Another type of underapproximation analysis has been conducted by [17] (and by [4] in another context), by considering bounded round-robin schedules of processes. Inspired by this work, we propose here to look at executions of broadcast networks over communication topologies where, for each process, the number of alternations between phases where it broadcasts messages and phases where it receives messages is bounded. We call such protocols *k*-phase-bounded protocols where *k* is the allowed number of alternations.

Our contributions. We study the parameterized coverability problem for broadcast networks working over communication topologies. We first show in Section 2 that it is enough to consider only tree topologies. This allows us to ease our presentation in the sequel and is also an interesting result by itself. In Section 3, we prove that the coverability problem



■ **Figure 1** Example of a broadcast protocol denoted P .

is still undecidable when considering k -phase-bounded broadcast protocols with k greater than 6. The undecidability proof relies on a technical reduction from the halting problem for two counter Minsky machines. We then show in Sections 4 and 5 that if the number of alternations is smaller or equal to 2, then decidability can be regained. More precisely, we show that for 1-phase-bounded protocols, we can restrict our attention to tree topologies of height 1, which provides an EXPSPACE-algorithm for the coverability problem. To solve this problem in the case of 2-phase-bounded protocols, we prove that we can bound the height of the considered tree and rely on the result of [6] which states that the coverability problem for broadcast networks is decidable when considering topologies where the length of all simple paths is bounded. We furthermore show that if we consider line topologies then the coverability problem restricted to 1- and 2-phase-bounded protocols can be solved in polynomial time.

Due to lack of space, omitted proofs and reasonings can be found in [15].

2 Preliminaries

Let A be a countable set, we denote A^* as the set of finite sequences of elements taken in A . Let $w \in A^*$, the length of w is defined as the number of elements in the sequence w and is denoted $|w|$. For a sequence $w = a_1 \cdot a_2 \cdots a_k \in A^+$, we denote by $w[-1]$ the sequence $a_1 \cdot a_2 \cdots a_{k-1}$. Let $\ell, n \in \mathbb{N}$ with $\ell \leq n$, we denote by $[\ell, n]$ the set of integers $\{\ell, \ell+1, \dots, n\}$.

2.1 Networks of processes

We study networks of processes where each process executes the same protocol given as a finite-state automaton. Given a finite set of messages Σ , a transition of the protocol can be labelled by three types of actions: (1) the broadcast of a message $m \in \Sigma$ with label $!!m$, (2) the reception of a message $m \in \Sigma$ with label $?m$ or (3) an internal action with a special label $\tau \notin \Sigma$. Processes are organised according to a topology which gives for each one of them its set of neighbors. When a process broadcasts a message $m \in \Sigma$, the only processes that can receive m are its neighbors, and the ones having an output action $?m$ have to receive it. Furthermore, the topology remains fixed during an execution.

Let Σ be a finite alphabet. In order to refer to the different types of actions, we write $!!\Sigma$ for the set $\{!!m \mid m \in \Sigma\}$ and $?\Sigma$ for $\{?m \mid m \in \Sigma\}$.

► **Definition 2.1.** A Broadcast Protocol is a tuple $P = (Q, \Sigma, q_{in}, \Delta)$ such that Q is a finite set of states, Σ is a finite alphabet of messages, q_{in} is an initial state and $\Delta \subseteq Q \times (!!\Sigma \times ?\Sigma \cup \{\tau\}) \times Q$ is a finite set of transitions.

We depict an example of a broadcast protocol in Figure 1. Processes are organised according to a topology, defined formally as follows.

► **Definition 2.2.** A topology is an undirected graph, i.e. a tuple $\Gamma = (V, E)$ such that V is a finite set of vertices, and $E \subseteq V \times V$ is a finite set of edges such that $(u, v) \in E$ implies $(v, u) \in E$ for all $(u, v) \in V^2$, and for all $u \in V$, $(u, u) \notin E$ (there is no self-loop).

We will use $V(\Gamma)$ and $E(\Gamma)$ to denote the set of vertices and edges of Γ respectively, namely V and E . For $v \in V$, we will denote $N_\Gamma(v)$ the set $\{u \mid (v, u) \in E\}$. When the context is clear, we will write $N(v)$. For $u, v \in V(\Gamma)$, we denote $\langle v, u \rangle$ for the two pairs $(v, u), (u, v)$. We name **Graphs** the set of topologies. In this work, we will also be interested in some families of topologies: line and tree topologies. A topology $\Gamma = (V, E)$ is a *tree topology* if V is a set of words of \mathbb{N}^* which is prefix closed with $\epsilon \in V$, and if $E = \{\langle w[-1], w \rangle \mid w \in V \cap \mathbb{N}^+\}$. This way, the *root* of the tree is the unique vertex $\epsilon \in V$ and a node $w \in V \cap \mathbb{N}^+$ has a unique parent $w[-1]$. The *height* of the tree is $\max\{n \in \mathbb{N} \mid |w| = n\}$. We denote by **Trees** the set of tree topologies. A topology $\Gamma = (V, E)$ is a *line topology* if V is such that $V = \{v_1, \dots, v_n\}$ for some $n \in \mathbb{N}$ and $E = \{\langle v_i, v_{i+1} \rangle \mid 1 \leq i < n\}$. We denote by **Lines** the set of line topologies.

Semantics. A *configuration* C of a broadcast protocol $P = (Q, \Sigma, q_{in}, \Delta)$ is a tuple (Γ, L) where Γ is a topology, and $L : V(\Gamma) \rightarrow Q$ is a labelling function associating to each vertex v of the topology its current state of the protocol. In the sequel, we will sometimes call processes or nodes the vertices of Γ . A configuration C is *initial* if $L(v) = q_{in}$ for all $v \in V(\Gamma)$. We let \mathcal{C}_P be the set of all configurations of P , and \mathcal{I}_P the set of all initial configurations. When P is clear from the context, we may drop the subscript and simply use \mathcal{C} and \mathcal{I} . Given a protocol $P = (Q, \Sigma, q_{in}, \Delta)$, and a state $q \in Q$, we let $R(q) = \{m \in \Sigma \mid \exists q' \in Q, (q, ?m, q') \in \Delta\}$ be the set of messages that can be received when in the state q .

Consider $\delta = (q, \alpha, q') \in \Delta$ a transition of P , and $C = (\Gamma, L)$ and $C' = (\Gamma', L')$ two configurations of P , and let $v \in V(\Gamma)$ be a vertex. The transition relation $\xrightarrow{v, \delta} \in \mathcal{C} \times \mathcal{C}$ is defined as follows: we have $C \xrightarrow{v, \delta} C'$ if and only if $\Gamma = \Gamma'$, and one of the following conditions holds:

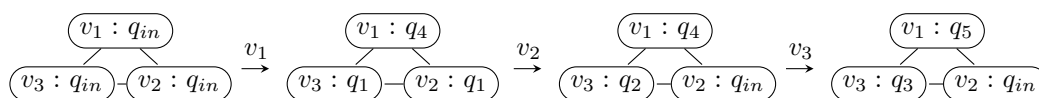
- $\alpha = \tau$ and $L(v) = q, L'(v) = q'$ and $L'(u) = L(u)$ for all $u \in V(\Gamma) \setminus \{v\}$: vertex v performs an internal action;
- $\alpha = !m$ and $L(v) = q, L'(v) = q'$ (vertex v performs a broadcast), and for each process $u \in N(v)$ neighbor of v , either $(L(u), ?m, L'(u)) \in \Delta$ (vertex u receives message m from v), or $m \notin R(L(u))$ and $L(u) = L'(u)$ (vertex u is not in a state in which it can receive m and stays in the same state). Furthermore, $L'(w) = L(w)$ for all other vertices $w \in V(\Gamma) \setminus (\{v\} \cup N(v))$ (vertex w does not change state).

We write $C \rightarrow C'$ whenever there exists $v \in V(\Gamma)$ and $\delta \in \Delta$ such that $C \xrightarrow{v, \delta} C'$. We denote by \rightarrow^* [resp. \rightarrow^+] for the reflexive and transitive closure [resp. transitive] of \rightarrow . An *execution* of P is a sequence of configurations $C_0, \dots, C_n \in \mathcal{C}_P$ such that for all $0 \leq i < n$, $C_i \rightarrow C_{i+1}$.

► **Example 2.3.** We depict in Figure 2 an execution of protocol P (from Figure 1): it starts with an initial configuration with three processes v_1, v_2, v_3 , organised as a clique (each vertex is a neighbour of the two others), each on the initial state q_{in} . More formally, $\Gamma = (V, E)$ with $V = \{v_1, v_2, v_3\}$ and $E = \{\langle v_1, v_2 \rangle, \langle v_2, v_3 \rangle, \langle v_1, v_3 \rangle\}$. From the initial configuration, the following chain of events happens: $C_0 \xrightarrow{v_1, (q_{in}, !!b, q_4)} C_1 \xrightarrow{v_2, (q_1, !!a, q_{in})} C_2 \xrightarrow{v_3, (q_2, !!c, q_3)} C_3$.

2.2 Verification problem

In this work, we focus on the *coverability problem* which consists in ensuring a safety property: we want to check that, no matter the number of processes in the network, nor the topology in which the processes are organised, a specific error state can never be reached.



■ **Figure 2** Example of an execution of protocol P (Figure 1).

The coverability problem over a family of topologies $\mathcal{S} \in \{\text{Graphs}, \text{Trees}, \text{Lines}\}$ is stated as follows:

COVER[\mathcal{S}]	
Input:	A broadcast protocol P and a state $q_f \in Q$;
Question:	Is there $\Gamma \in \mathcal{S}$, $C = (\Gamma, L) \in \mathcal{I}_P$ and $C' = (\Gamma, L') \in \mathcal{C}_P$ and $v \in \mathbf{V}(\Gamma)$ such that $C \rightarrow^* C'$ and $L'(v) = q_f$?

For a family \mathcal{S} , if indeed there exist $C = (\Gamma, L)$ and $C' = (\Gamma, L')$ such that $C \rightarrow^* C'$ and $L'(v) = q_f$ for some $v \in \mathbf{V}(\Gamma)$, we say that q_f is *coverable* (in P) with Γ . We also say that the execution $C \rightarrow^* C'$ *covers* q_f . For short, we write COVER instead of COVER[Graphs]. Observe that COVER is a generalisation of COVER[Trees] which is itself a generalisation of COVER[Lines]. In [6], the authors proved that the three problems are undecidable, and they later showed in [7] that the undecidability of COVER still holds when restricting the problem to families of topologies with bounded diameter.

However, in [6], the authors show that COVER becomes decidable when searching for an execution covering q_f with a K -bounded path topology for some $K \in \mathbb{N}$, i.e. for a topology in which all simple paths between any pair of vertices $v_1, v_2 \in V$ have a length bounded by K . In [7], it is also shown that COVER is Ackermann-hard when searching for an execution covering q_f with a topology where all maximal cliques are connected by paths of bounded length. We establish the first result.

► **Theorem 2.4.** *COVER[Graphs] and COVER[Trees] are equivalent.*

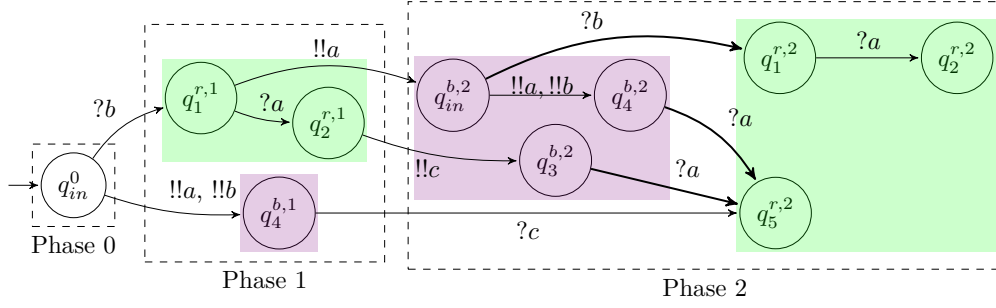
Indeed, if it is obvious that when a state is coverable with a tree topology, it is coverable with a topology from Graphs, we can show that whenever a state is coverable, it is coverable with a tree topology. If a set q_f of a protocol P is coverable with a topology $\Gamma \in \text{Graphs}$, let $\rho = C_0 \rightarrow \dots \rightarrow C_n = (\Gamma, L_n)$ be an execution covering q_f , and a vertex $v_f \in \mathbf{V}(\Gamma)$ such that $L_n(v_f) = q_f$. We can build an execution covering q_f with a tree topology Γ' where the root reaches q_f . Actually, Γ' is the unfolding of Γ in a tree of height n .

3 Phase-Bounded Protocols

As COVER[Graphs], COVER[Trees] and COVER[Lines] are undecidable in the general case, we investigate a restriction on broadcast protocols: phase-bounded protocols.

For $k \in \mathbb{N}$, a k -phase-bounded protocol is a protocol that ensures that each process alternates at most k times between phases of broadcasts and phases of receptions. Before giving our formal definition of a phase-bounded protocol, we motivate this restriction.

Phase-bounded protocols can be seen as a semantic restriction of general protocols in which each process can only switch a bounded number of times between phases where it receives messages and phases where it broadcasts messages. When, usually, restricting the behavior of processes immediately yields an underapproximation of the reachable states, we highlight in [15] the fact that preventing messages from being received can in fact lead to new reachable states. Actually, the reception of a message is something that is not under



■ **Figure 3** P_2 : the 2-unfolding of protocol P (Figure 1).

the control of a process. If another process broadcasts a message, a faithful behavior of the system is that all the processes that can receive it indeed do so, no matter in which phase they are in their own execution. Hence, in a restriction that attempts to limit the number of switches between broadcasting and receiving phases, one should not prevent a reception to happen. This motivates our definition of phase-bounded protocols, in which a process in its last broadcasting phase, can still receive messages. A k -unfolding of a protocol P is then a protocol in which we duplicate the vertices by annotating them with the type and the number of phase (b or r for broadcast or reception and an integer between 0 and k for the number).

► **Example 3.1.** Figure 3 pictures the 2-unfolding of protocol P (Figure 1). Observe that from state $q_4^{b,2}$, which is a broadcast state, it is still possible to receive message a and go to state $q_5^{r,2}$. However, it is not possible to send a message from $q_5^{r,2}$ (nor from any reception state of phase 2).

We show in [15] that this definition of unfolding can be used as an underapproximation for COVER. In the remaining of the paper, we study the verification problems introduced in Section 2.2 when considering phase-bounded behaviors. We turn this restriction into a syntactic one over the protocol, defined as follows.

► **Definition 3.2.** Let $k \in \mathbb{N}$. A broadcast protocol $P = (Q, \Sigma, q_{in}, \Delta)$ is k -phase-bounded if Q can be partitioned into $2k + 1$ sets $\mathcal{Q} = \{Q_0, Q_1^b, Q_1^r, \dots, Q_k^b, Q_k^r\}$, such that $q_{in} \in Q_0$ and for all $(q, \alpha, q') \in \Delta$ one of the following conditions holds:

1. there exist $0 \leq i \leq k$ and $\beta \in \{r, b\}$ such that $q, q' \in Q_i^\beta$ and $\alpha = \tau$ (for ease of notation, we take $Q_0 = Q_0^b = Q_0^r$);
2. there exists $1 \leq i \leq k$ such that $q, q' \in Q_i^b$ and $\alpha \in !!\Sigma$;
3. there exists $1 \leq i \leq k$ such that $q, q' \in Q_i^r$ and $\alpha \in ?\Sigma$;
4. there exists $0 \leq i < k$ such that $q \in Q_i^b$, $q' \in Q_{i+1}^r$ and $\alpha \in ?\Sigma$;
5. there exists $0 \leq i < k$ such that $q \in Q_i^r$, $q' \in Q_{i+1}^b$ and $\alpha \in !!\Sigma$;
6. $q \in Q_k^b$, $q' \in Q_k^r$ and $\alpha \in ?\Sigma$

A protocol P is phase-bounded if there exists $k \in \mathbb{N}$ such that P is k -phase-bounded.

► **Example 3.3.** Observe that the protocol P displayed in Figure 1 is not phase-bounded: by definition, it holds that $Q_0 = \{q_{in}\}$, and $q_1 \in Q_1^r$ (because of the transition $(q_{in}, ?b, q_1)$). As a consequence $q_{in} \in Q_2^b$, because of the transition $(q_1, !!a, q_{in})$. This contradicts the fact that $Q_2^b \cap Q_0 = \emptyset$. Intuitively, P does not ensure that every vertex alternates at most a bounded number of times between receptions and broadcasts, in particular, for any integer $k \in \mathbb{N}$, it might be that there exists an execution where a process alternates $k + 1$ times

between reception of a message b from state q_{in} , and broadcast of a message a from state q_1 . Removing the transition $(q_1, !!a, q_{in})$ from P would give a 2-phase-bounded protocol P' : $Q_0 = \{q_{in}\}$, $Q_1^r = \{q_1, q_2\}$, $Q_1^b = \{q_4\}$, $Q_2^b = \{q_3\}$ and $Q_2^r = \{q_5\}$.

The following table summarizes our results (PB stands for phase-bounded).

	1-PB Protocols	2-PB Protocols	PB Protocols
COVER[Lines]	$\in P$ (Section 6.2)		Undecidable ($k \geq 4$) (Sec 4)
COVER[Graphs]	EXPSpace-complete	Decidable	Undecidable ($k \geq 6$)
COVER[Trees]	(Section 5)	(Section 6.1)	(Section 4)

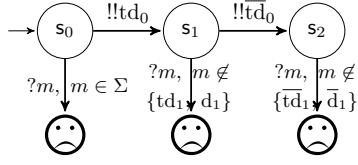
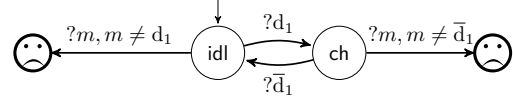
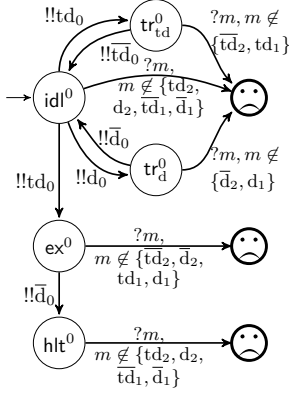
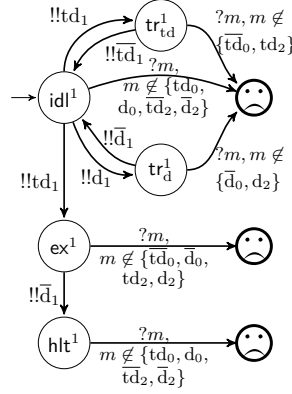
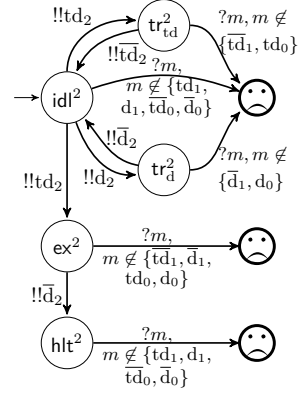
4 Undecidability Results

We prove that COVER restricted to k -phase-bounded protocols (with $k \geq 6$) is undecidable by a reduction from the halting problem of a Minsky machine [19]: a Minsky machine is a finite-state machine (whose states are called locations) with two counters, x_1 and x_2 (two variables that take their values in \mathbb{N}). Each transition of the machine is associated with an instruction: increment one of the counters, decrement one of the counters or test if one of the counters is equal to 0. The halting problem asks whether there is an execution that ends in the halting location. In a first step, the protocol will enforce the selection of a line of nodes from the topology. All other nodes will be inactive. In a second step, the first node of the line (that we call the head) visits the different states of the machine during an execution, while all other nodes (except the last one) simulate counters' values: they are either in a state representing value 0, or a state representing x_1 (respectively x_2). The number of processes on states representing x_1 gives the actual value of x_1 in the execution. The last node (called the tail) checks that everything happens as expected. When the head has reached the halting location of the machine, it broadcasts a message which is received and forwarded by each node of the line until the tail receives it and reaches the final state to cover.

When the head of the line simulates a transition of the machine, it broadcasts a message (the instruction for one of the counters), which is transmitted by each node of the line until the tail receives it. A classical way of forwarding the message through receptions and broadcasts would not give a phase-bounded protocol. Hence, during the transmission, the tail only receives messages and all other nodes only broadcast and do not receive any message. The main idea is that we do not use the reception of messages to move into the next state of the execution but to detect errors (and in that case, go to a bad sink state from which the process can not do anything). The processes will have to guess the correct message to send, and the correct instant to send it, otherwise some of them will go to the sink state upon the reception of this "wrong" message. Hence, when everyone makes the correct guesses, the only reception that occurs in the transmission is done by the tail process, whereas when someone makes an incorrect guess, a process goes to a bad state with a reception. In the reduction, if the halting state of the Minsky Machine is not reachable, there will be no way to make a correct guess that allows to cover the final state. In the next subsection, we explain how this is achieved. To do so, we explain the mechanism by abstracting away the actual instruction, and just show how to transmit a message.

4.1 Propagating a message using only broadcasts in a line

In a line, a node has at most two neighbors, but cannot necessarily distinguish between the two (its left and its right one). To do so, nodes broadcast messages with subscript 0, 1 or 2, and we ensure that: if a node broadcasts with subscript 1, its right [resp. left] neighbor broadcasts with subscript 0 [resp. subscript 2]. Similarly, if a node broadcasts with subscript

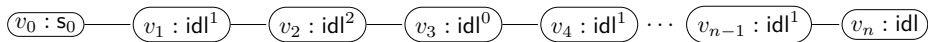

 ■ Figure 4 Protocol P_h executed by v_0 .

 ■ Figure 5 Protocol P_t executed by v_n .

 ■ Figure 6 P_0 .

 ■ Figure 7 P_1 .

 ■ Figure 8 P_2 .

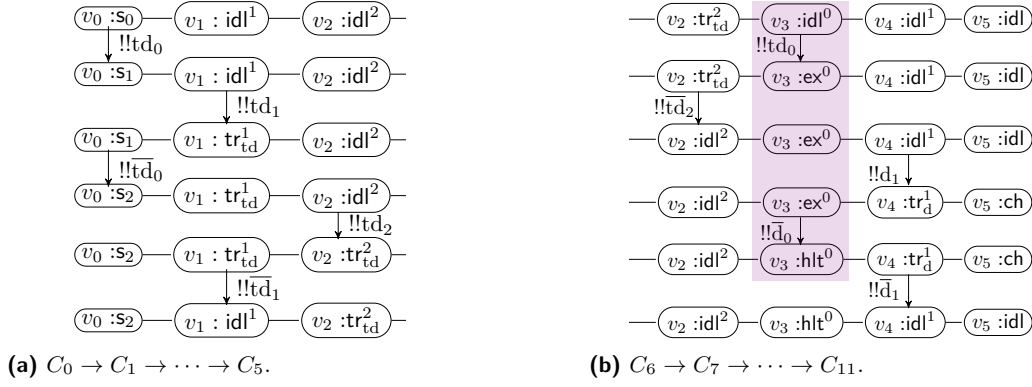
0 [resp. 2], its right neighbor broadcasts with subscript 2 [resp. 1] and its left one with subscript 1 [resp. 0].

Consider the five protocols displayed in Figures 4–8. The states marked as initial are the ones from which a process enters the protocol. Protocol P_h is executed by the head of the line, P_t by the tail of the line and other nodes execute either P_0 , P_1 or P_2 . Observe that messages go by pairs: td_i, \bar{td}_i and d_i, \bar{d}_i for all $i \in \{0, 1, 2\}$.

The head broadcasts a request to be done with the pair of messages td_0, \bar{td}_0 . Each process in one of the P_i starts in idl^i and has a choice: either it transmits a message without executing it, or it “executes” it and tells it to the others. When it transmits a message not yet executed, it broadcasts the messages td_i and \bar{td}_i and visits states tr_{td}^i and idl^i . When it executes the request, it broadcasts the messages td_i and \bar{d}_i and visits states ex^i and hlt^i . Finally, when it transmits a request already done, it broadcasts the messages d_i and \bar{d}_i and visits states tr_d^i and idl^i . Once a process has executed the request (i.e. broadcast a pair td_j, \bar{d}_j for some $j \in \{0, 1, 2\}$), only pairs d_j, \bar{d}_j , with $j \in \{0, 1, 2\}$, are transmitted in the rest of the line.

Correct transmission of a request. Take for instance the configuration C_0 depicted in Figure 9 for $n = 5$ (i.e. there are six vertices). We say that a configuration is *stable* if the head is in s_0 or s_2 , the tail is in idl and other nodes are in idl^i or hlt^i for $i \in \{0, 1, 2\}$. Note that C_0 is stable. We depict a transmission in Figures 10a and 10b, starting from C_0 . We denote the successive depicted configurations C_0, C_1, \dots, C_{11} . Note that C_{11} is stable. Between C_0 and


 ■ Figure 9 A configuration from which the transmission can happen: a node in state idl^i can only broadcast messages with subscript i .



■ **Figure 10** Example of correct transmission.

C_{11} , the following happens: Between C_0 and C_3 , v_0 broadcasts the request with messages td_0 and \bar{td}_0 . Between C_1 and C_8 , v_1 and v_2 successively repeat the request to be done with messages td_1 and \bar{td}_1 for v_1 and td_2 \bar{td}_2 for v_2 . Between C_6 and C_{10} , v_3 executes the request by broadcasting messages td_0 and \bar{d}_0 . Between C_7 and C_{11} , v_4 transmits the done request with messages d_1 and \bar{d}_1 . Hence, the request is executed by exactly one vertex (namely v_3), as highlighted in Figure 10b. Observe that the processes sort of spontaneously emit broadcast to avoid to receive a message. A correct guess of when to broadcast yields the interleaving of broadcasts that we have presented in this example.

How to prevent wrong behaviors? Observe that, when a node is in state idl^1 , if one of its neighbor broadcasts a message which is not td_0, d_0 or \bar{td}_2, \bar{d}_2 , then the node in idl^1 reaches \ominus . We say that a process *fails* whenever it reaches \ominus . We have the following lemma:

► **Lemma 4.1.** *Let $C \in \mathcal{C}$ be a stable configuration such that $C_0 \rightarrow^+ C$. Then in C , it holds that v_0 is in s_2 , and there is exactly one vertex $v \in \{v_1, v_2, v_3, v_4\}$ on a state hlt^j for some $j \in \{0, 1, 2\}$.*

Indeed, let C be a *stable* configuration such that $C_0 \rightarrow^+ C$. It holds that:

1. *From C_0 , the first broadcast is from v_0 and it broadcasts td_0 .*
Indeed, if another vertex than v_0 broadcasts a message m with subscript i from C_0 , its left neighbor would fail with transition $(idl^j, ?m, \ominus)$ as $j = (i - 1) \bmod 3$ and $m \in \{td_i, d_i\}$. Let us consider an example depicted in Figure 11b: Assume v_1 is in state idl^1 and v_2 broadcasts td_2 or d_2 (it issues a request whereas v_1 is not broadcasting any request), then v_1 receives the message with transition that goes from idl^1 to \ominus , as depicted in Figure 7. Hence, we can not reach a stable configuration from there.
2. *Each vertex (except the tail) broadcasts one pair of messages between C_0 and C .*
Assume for instance that v_1 does not broadcast anything. From Item 1, v_0 broadcasts td_0 , and so at some point it will also broadcasts \bar{td}_0 otherwise it would not be in s_0 or s_2 in C . Hence v_1 fails as depicted in Figure 11a. Actually, each vertex (except the tail) broadcasts exactly one pair: if it broadcasts more, its left neighbor would fail as well.
3. *When a node broadcasts a pair (td_j, \bar{td}_j) , its right neighbor broadcasts either a pair (td_i, \bar{td}_i) or (d_i, \bar{d}_i) , for $j, i \in \{0, 1, 2\}$.*
Assume its right neighbor broadcasts d_i , it must be that $i = (j + 1) \bmod 3$. Such an example is depicted in Figure 11b: v_1 fails with $(tr_{td}^1, ?d_2, \ominus)$. Similarly, we have:
4. *When a node broadcasts a pair (td_j, \bar{d}_j) or a pair (d_j, \bar{d}_j) , its right neighbor broadcasts a pair (d_i, \bar{d}_i) , for $j, i \in \{0, 1, 2\}$.*



(a) v_1 does not transmit the request.

(b) v_2 broadcasts the wrong pair of messages.

■ **Figure 11** Example of wrong behaviors during the transmission.

4.2 Putting everything together

We adapt the construction of Section 4.1 to propagate operations on counters of the machine issued by the head of the line. Counters processes will evolve in three different protocols as in Section 4.1. They can be either in a zero state, from which all the types of instructions can be transmitted, or in a state 1_x for x one of the two counters, from which all the types of operations can be transmitted, except 0-tests of x . Increments and decrements of a counter x are done in a similar fashion as in Section 4.1 (exactly one node changes its state). 0-tests are somewhat easier: no node changes state nor executes anything, and the tail accepts the same pair as the one broadcast by the head. However, if a node is in a 1_x when x is the counter compared to 0, it fails when its left neighbor broadcasts the request.

We ensure that we can select a line with a similar structure as the one depicted in Figure 9 thanks to a first part of the protocol where each node: (i) receives an announcement message from its predecessor with a subscript j (except the head which broadcasts first), (ii) broadcasts an announcement message with the subscript $(j + 1) \bmod 3$ (head broadcasts with subscript 0) and (iii) waits for the announcement of its successor with subscript $(j + 2) \bmod 3$ (except for the tail). If it receives any new announcement at any point of its execution, it fails. When considering only line topologies, as each node has at most two neighbors, this part can be achieved with fewer alternations. We get the two following theorems.

► **Theorem 4.2.** *COVER and COVER[Trees] are undecidable for k -phase-bounded protocols with $k \geq 6$.*

► **Theorem 4.3.** *COVER[Lines] is undecidable for k -phase-bounded protocols with $k \geq 4$.*

5 Cover in 1-Phase-Bounded Protocols

We show that COVER[Graphs] restricted to 1-phase-bounded protocols is EXPSPACE-complete.

We begin by proving that for such protocols COVER[Graphs] and COVER[Stars] are equivalent (where Stars correspond to the tree topologies of height one). To get this property, we first rely on Theorem 2.4 (stating that COVER and COVER[Trees] are equivalent) and without loss of generality we can assume that if a control state can be covered with a tree topology, it can be covered by the root of the tree. We then observe that when dealing with 1-phase-bounded protocols, the behaviour of the processes of a tree which are located at a height strictly greater than 1 have no incidence on the root node. Indeed if a process at depth 2 performs a broadcast received by a node at depth 1, then this latter node will not be able to influence the state of the root because in 1-phase-bounded protocols, once a process has performed a reception, it cannot broadcast anymore. In the sequel we fix a 1-phase-bounded protocol $P = (Q, \Sigma, q_{in}, \Delta)$ and a state $q_f \in Q$. We then have:

► **Lemma 5.1.** *There exist $\Gamma \in \mathbf{Graphs}$, $C = (\Gamma, L) \in \mathcal{I}_P$ and $D = (\Gamma, L') \in \mathcal{C}_P$ and $v \in \mathcal{V}(\Gamma)$ such that $C \rightarrow^* D$ and $L'(v) = q_f$ iff there exists $\Gamma' \in \mathbf{Stars}$, $C' = (\Gamma', L'') \in \mathcal{I}$ and $D' = (\Gamma', L''') \in \mathcal{C}_P$ such that $C' \rightarrow_P^* D'$ and $L'''(\epsilon) = q_f$.*

To solve COVER[Stars] in EXPSPACE, we proceed as follows (1) we first propose an abstract representation for the configurations reachable by executions where the root node does not perform any reception, and that only keeps track of states in Q_0 and Q_1^b (2) we show that we can decide in polynomial space whether a configuration corresponding to a given abstract representation can be reached from an initial configuration (3) relying on reduction to the control state reachability problem in VASS (Vector Addition System with States), we show how to decide whether there exists a configuration corresponding to a given abstract representation from which q_f can be covered in an execution where the root node does not perform any broadcast. This reasoning relies on the fact that a process executing a 1-phase-bounded protocol first performs only broadcast (or internal actions) and then performs only receptions (or internal actions).

We use Q^b to represent the set $Q_0 \cup Q_1^b$ and we say that a configuration $C = (\Gamma, L)$ in \mathcal{C}_P is a *star-configuration* whenever $\Gamma \in \mathbf{Stars}$. For a star-configuration $C = (\Gamma, L)$ in \mathcal{C}_P such that $L(\epsilon) \in Q^b$, the *broadcast-print* of C , denoted by $\mathbf{bprint}(C)$, is the pair $(L(\epsilon), \{L(v) \in Q^b \mid v \in \mathcal{V}(\Gamma) \setminus \{\epsilon\}\})$ in $Q^b \times 2^{Q^b}$. We call such a configuration C a *b-configuration*. Note that any initial star-configuration $C_{in} = (\Gamma_{in}, L_{in}) \in \mathcal{I}$ is a b-configuration verifying $\mathbf{bprint}(C_{in}) \in \{(q_{in}, \emptyset), (q_{in}, \{q_{in}\})\}$ (the first case corresponding to $\mathcal{V}(\Gamma) = \{\epsilon\}$). We now define a transition relation \Rightarrow between broadcast-prints. Given (q, Λ) and (q', Λ') in $Q^b \times 2^{Q^b}$, we write $(q, \Lambda) \Rightarrow (q', \Lambda')$ if there exists two b-configurations C and C' such that $\mathbf{bprint}(C) = (q, \Lambda)$ and $\mathbf{bprint}(C') = (q', \Lambda')$ and $C \rightarrow C'$. We denote by \Rightarrow^* the reflexive and transitive closure of \Rightarrow .

One interesting point of this abstract representation is that we can compute in polynomial time the \Rightarrow -successor of a given broadcast-print. The intuition is simple: either the root performs a broadcast of $m \in \Sigma$, and in that case we have to remove from the set Λ all the states from which a reception of m can be done (as the associated processes in C' will not be in a state in Q^b anymore) or one process in a state of Λ performs a broadcast and in that case it should not be received by the root node (otherwise the reached configuration will not be a b-configuration anymore).

► **Lemma 5.2.** *Given $(q, \Lambda) \in Q^b \times 2^{Q^b}$, we can compute in polynomial time the set $\{(q', \Lambda') \mid (q, \Lambda) \Rightarrow (q', \Lambda')\}$.*

In order to show that our abstract representation can be used to solve COVER[Stars], we need to rely on some further formal definitions. Given two star-configurations $C = (\Gamma, L)$ and $C' = (\Gamma', L')$, we write $C \preceq C'$ iff the two following conditions hold (i) $L(\epsilon) = L'(\epsilon)$, and, (ii) $|\{v \in \mathcal{V}(\Gamma) \setminus \{\epsilon\} \mid L(v) = q\}| \leq |\{v \in \mathcal{V}(\Gamma') \setminus \{\epsilon\} \mid L'(v) = q\}|$ for all $q \in Q^b$. We then have the following lemma where the two first points show that when dealing with star-configurations, the network generated by 1-phase-bounded protocol enjoys some monotonicity properties. Indeed, if the root node performs a broadcast received by other nodes, then if we put more nodes in the same state, they will also receive the message. On the other hand if it is another node that performs a broadcast, only the root node is able to receive it. The last point of the lemma shows that we can have as many processes as we want in reachable states in Q^b (as soon as the root node does not perform any reception) by duplicating nodes and mimicking behaviors.

► **Lemma 5.3.** *The following properties hold:*

- (i) *If C_1, C'_1 and C_2 are star-configurations such that $C_1 \rightarrow C'_1$ and $C_1 \preceq C_2$ then there exists a star-configuration C'_2 such that $C'_1 \preceq C'_2$ and $C_2 \rightarrow^* C'_2$.*
- (ii) *If C_1, C'_1 and C_2 are b-configurations such that $C_1 \rightarrow C'_1$ and $\mathbf{bprint}(C_1) = \mathbf{bprint}(C_2)$ and $C_1 \preceq C_2$ then there exists a b-configuration C'_2 such that $C'_1 \preceq C'_2$ and $\mathbf{bprint}(C'_1) = \mathbf{bprint}(C'_2)$ and $C_2 \rightarrow^* C'_2$.*
- (iii) *If C is a b-configuration such that $C_{in} \rightarrow^* C$ for some initial configuration C_{in} then for all $N \in \mathbb{N}$, there exists an initial configuration C'_{in} and a b-configuration $C' = (\Gamma', L')$ such that $C'_{in} \rightarrow^* C'$ and $\mathbf{bprint}(C) = \mathbf{bprint}(C') = (q, \Lambda)$ and $|\{v \in V(\Gamma') \setminus \{\epsilon\} \mid L'(v) = q'\}| \geq N$ for all $q' \in \Lambda$.*

We can now prove that we can reason in a sound and complete way with broadcast prints to characterise the b-configurations reachable from initial star-configurations. To prove this next lemma, we rely on the two last points of the previous lemma and reason by induction on the length of the \Rightarrow -path leading from (q_{in}, Λ_{in}) to (q, Λ) .

► **Lemma 5.4.** *Given $(q, \Lambda) \in Q^b \times 2^{Q^b}$, we have $(q_{in}, \Lambda_{in}) \Rightarrow^* (q, \Lambda)$ with $\Lambda_{in} \in \{\emptyset, \{q_{in}\}\}$ iff there exist two b-configurations $C_{in} \in \mathcal{I}$ and $C \in \mathcal{C}$ such that $C_{in} \rightarrow^* C$ and $\mathbf{bprint}(C) = (q, \Lambda)$.*

Finally, we show that we can verify in exponential space whether there exists a configuration with a given broadcast-print (q, Λ) from which we can reach a configuration covering q_f thanks to an execution where the root node does not perform any broadcast. This result is obtained by a reduction to the control state reachability problem in (unary) VASS which is known to be EXPSpace-complete [18, 21]. VASS are finite state machines equipped with variables (called counters) taking their values in \mathbb{N} , and where each transition of the machine can either change the value of a counter, by incrementing or decrementing it, or do nothing. In our reduction, we encode the state of the root in the control state of the VASS and we associate a counter to each state of Q^b to represent the number of processes in this state. In a first phase, the VASS generates a configuration with (q, Λ) as broadcast-print and in a second phase it simulates the network. For instance, if a process performs a broadcast received by the root node, then we decrement the counter associated to the source state of the broadcast, we increment the one associated to the target state and we change the control state of the VASS representing the state of the root node accordingly. We need a last definition to characterise executions where the root node does not perform any broadcast: given two star-configurations $C = (\Gamma, L)$ and $C' = (\Gamma, L')$, we write $C \rightarrow_r C'$ whenever there exist $v \in V(\Gamma)$ and $\delta \in \Delta$ such that $C \xrightarrow{v, \delta} C'$ and either $v \neq \epsilon$ or $\delta = (q, \tau, q')$ for some $q, q' \in Q$. We denote by \rightarrow_r^* the reflexive and transitive closure of \rightarrow_r .

► **Lemma 5.5.** *Given $(q, \Lambda) \in Q^b \times 2^{Q^b}$, we can decide in EXPSpace whether there exist a b-configuration $C = (\Gamma_f, L)$ and a star-configuration $C_f = (\Gamma_f, L_f)$ such that $\mathbf{bprint}(C) = (q, \Lambda)$ and $L_f(\epsilon) = q_f$ and $C \rightarrow_r^* C_f$.*

Combining the results of the previous lemmas leads to an EXPSpace-algorithm to solve COVER[Stars]. We first guess a broadcast-print (q, Λ) and check in polynomial space whether it is \Rightarrow -reachable from an initial broadcast-print in $\{(q_{in}, \emptyset), (q_{in}, \{q_{in}\})\}$ thanks to Lemma 5.2 (relying on a non-deterministic polynomial space algorithm for reachability). Then we use Lemma 5.5 to check the existence of a b-configuration C with $\mathbf{bprint}(C) = (q, \Lambda)$ from which we can cover q_f . By Savitch's theorem [23], we conclude that the problem is in EXPSpace. The completeness of this method is direct. For the soundness, we reason as

follows: using Lemma 5.4, there exists a configuration C reachable from an initial star-configuration such that $\mathbf{bprint}(C) = (q, \Lambda)$, and by Lemma 5.5, there is a configuration C' such that $\mathbf{bprint}(C') = (q, \Lambda)$ from which we cover q_f . Thanks to Lemma 5.3.(iii), there is a configuration C'' reachable from an initial configuration such that $C \preceq C''$ and $C' \preceq C''$ and $\mathbf{bprint}(C'') = (q, \Lambda)$. Thanks to Lemma 5.3.(i) applied to each transition, we can build an execution from C'' that covers q_f . The lower bound is obtained by a reduction from the control state reachability in VASS.

► **Theorem 5.6.** *COVER[Graphs] and COVER[Trees] are EXPSPACE-complete for 1-phase-bounded protocols.*

6 Decidability Results for 2-Phase-Bounded Protocols

6.1 Cover and Cover[Trees] are Decidable on 2-PB Protocols

A *simple path between u and u'* in a topology $\Gamma = (V, E)$ is a sequence of distinct vertices v_0, \dots, v_k such that $u = v_0$, $u' = v_k$, and for all $0 \leq i < k$, $(v_i, v_{i+1}) \in E$. Its length is denoted $d(v_0, \dots, v_k)$ and is equal to k . Given an integer K , we say that a topology Γ is K -bounded path (and we write $\Gamma \in K\text{-BP}$) if there is no simple path v_0, \dots, v_k such that $d(v_0, \dots, v_k) > K$. The result of this subsection relies on the following theorem.

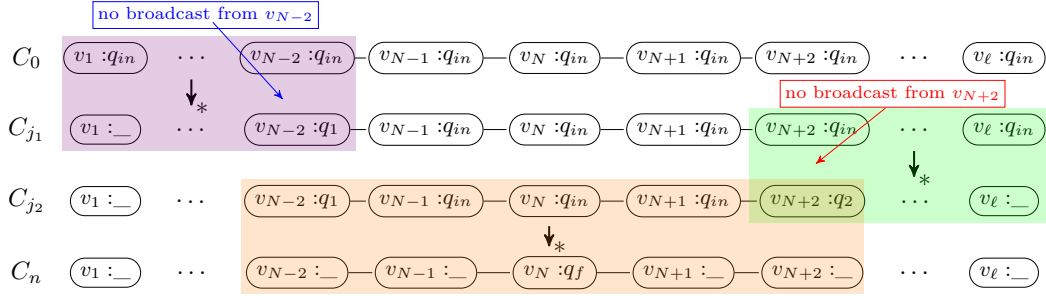
► **Theorem 6.1** ([6], Theorem 5). *For $K \geq 1$, COVER[K-BP] is decidable.*

Hence, we show that if a state q_f of a protocol P is coverable with a tree topology, then q_f is actually coverable with a tree topology that is also $2(|Q| + 1)\text{-BP}$. To establish this result, consider a coverable state q_f of a protocol P with a tree topology Γ , such that Γ is minimal in the number of nodes needed to cover q_f . We can suppose wlog that q_f is covered by the root of the tree. We argue that all nodes (except maybe the root) in the execution covering q_f broadcast something, as otherwise they are useless and could then be removed. We also argue that, since P is 2-phase-bounded, a node that would first broadcast after the first broadcast of its father would also be useless for the covering of q_f : this broadcast will only be received by its father in its *last phase of reception*, hence it will have no influence on the behavior of the root. These two properties are the key elements needed to establish the following lemma.

► **Lemma 6.2.** *Let $P = (Q, \Sigma, q_{in}, \Delta)$ be a 2-phase-bounded protocol and $q_f \in Q$. If q_f can be covered with a tree topology, then it can be covered with a topology $\Gamma \in \text{Trees}$ such that, for all $u \in V(\Gamma)$, $|u| \leq |Q| + 1$.*

Indeed, a counting argument implies that if this is not the case, there exist two nodes u_1 and u_2 on the same branch, different from the root, with u_1 a prefix of u_2 , that both execute their first broadcast from the same state q . In this case, we could replace the subtree rooted in u_1 by the subtree rooted in u_2 , and still obtain an execution covering q_f . Once u_1 has reached q (possibly by receiving broadcasts from the children of u_2), it will behave as in the initial execution. Behaviors of the children of u_1 might differ in this second part, but it can only influence u_1 in its reception phase, which will be the last phase, and hence will not influence the behavior of the root. Thanks to Theorems 2.4 and 6.1, we can then conclude.

► **Theorem 6.3.** *COVER and COVER[Trees] are decidable for 2-phase-bounded protocols.*



■ **Figure 12** Illustration of execution ρ obtained from Lemma 6.4.

6.2 Polynomial Time Algorithm for Cover[Lines] on 2-PB Protocols

In the rest of this section, we fix a 2-phase-bounded protocol $P = (Q, \Sigma, q_{in}, \Delta)$ and a state $q_f \in Q$ to cover. For an execution $\rho = C_0 \rightarrow C_1 \rightarrow \dots \rightarrow C_n$ with $C_n = (\Gamma, L_n)$, for all $v \in V(\Gamma)$, we denote by $b_{\text{first}}(v, \rho)$ the *smallest* index $0 \leq i < n$ such that $C_i \xrightarrow{v, t} C_{i+1}$ with $t = (q, \tau, q') \in \Delta$. If v never broadcasts anything, $b_{\text{first}}(v, \rho) = -1$. We also denote by $t_{\text{last}}(v, \rho)$ the *largest* index $0 \leq i < n$, such that $C_i \xrightarrow{v, t} C_{i+1}$ for some transition $t \in \Delta$. If v never issues any transition, we let $t_{\text{last}}(v, \rho) = -1$.

The polynomial time algorithm relies on the fact that to cover a state, one can consider only executions that have a specific shape, described in the following lemma.

► **Lemma 6.4.** *If q_f is coverable with a line topology Γ such that $V(\Gamma) = \{v_1, \dots, v_\ell\}$ then there exists an execution $\rho = C_0 \rightarrow C_1 \rightarrow \dots \rightarrow C_n$ such that $C_n = (\Gamma, L_n)$, and $3 \leq N \leq \ell - 2$ with $L_n(v_N) = q_f$, and*

1. *there exist $0 \leq j_1 < j_2 < n$ such that for all $0 \leq j < n$, if we let $C_j \xrightarrow{v^j, t^j} C_{j+1}$:*
 - (a) *if $0 \leq j < j_1$, then $v^j \in \{v_1, \dots, v_{N-2}\}$ and if $v^j = v_{N-2}$, then $t^j = (q, \tau, q')$ for some $q, q' \in Q$; and*
 - (b) *if $j_1 \leq j < j_2$, then $v^j \in \{v_{N+2}, \dots, v_\ell\}$ and if $v^j = v_{N+2}$, then $t^j = (q, \tau, q')$ for some $q, q' \in Q$; and*
 - (c) *if $j_2 \leq j < n$, then $v^j \in \{v_{N-2}, \dots, v_{N+2}\}$.*
2. (a) *for all $1 \leq i \leq N - 2$, $t_{\text{last}}(v_i, \rho) \leq b_{\text{first}}(v_{i+1}, \rho)$, and*
 (b) *for all $N + 2 \leq i \leq \ell$, $t_{\text{last}}(v_i, \rho) \leq b_{\text{first}}(v_{i-1}, \rho)$.*

Figure 12 illustrates the specific form of the execution described in Item 1 of Lemma 6.4: the first nodes to take actions are the ones in the purple part (on the left), then, only nodes in the green part (on the right) issue transitions), and finally the nodes in the orange central part take actions in order to reach q_f . The fact that P is 2-phase bounded allows us to establish Item 2 of Lemma 6.4: when v_{i+1} starts broadcasting, no further broadcasts from v_i will influence v_{i+1} 's broadcasts (it can only receive them in its last reception phase).

Figure 12 highlights why we get a polynomial time algorithm: when we reach the orange part of the execution, the nodes v_{N-1} , v_N and v_{N+1} are still in the initial state of the protocol. Moreover, in the orange part (which is the one that witnesses the covering of q_f), only five nodes take actions. Once one has computed in which set of states the nodes v_{N-2} and v_{N+2} can be at the beginning of the orange part, it only remains to compute the set of reachable configurations from a finite set of configurations. Let H be the set of possible states in which

v_{N-2} and v_{N+2} can be at the beginning of the last part of the execution, and for $q_1, q_2 \in H$, let $C_{q_1, q_2} = (\Gamma_5, L_{q_1, q_2})$ where Γ_5 is the line topology with five vertices $\{v_1, v_2, v_3, v_4, v_5\}$ and $L_{q_1, q_2}(v_1) = q_1$, $L_{q_1, q_2}(v_5) = q_2$ and for all other vertex v , $L_{q_1, q_2}(v) = q_{in}$.

Our algorithm is then: (1) Compute H ; (2) For all $q_1, q_2 \in H$, explore reachable configurations from C_{q_1, q_2} ; (3) Answer yes if we reach a configuration covering q_f , answer no otherwise. It remains to explain how to compute H . This computation relies on Item 2 of Lemma 6.4: locally, each node v_i at the left of v_{N-1} (resp. at the right of v_{N+1}) stops issuing transitions once its right neighbor v_{i+1} (resp. its left neighbor v_{i-1}) starts broadcasting.

Hence we compute iteratively set of coverable pairs of states $S \subseteq Q \times Q$ by relying on a family $(S_i)_{i \in \mathbb{N}}$ of subsets of $Q \times Q$ formally defined as follows:

$$\begin{aligned} S_0 &= \{(q_{in}, q_{in})\} \\ S_{i+1} &= S_i \cup \{(q_1, q_2) \mid \text{there exist } (p_1, p_2) \in S_i, j \in \{1, 2\} \text{ s.t. } (p_j, \tau, q_j) \in \Delta \text{ and } p_{3-j} = q_{3-j}\} \\ &\cup \{(q_1, q_2) \mid \text{there exists } (p_1, p_2) \in S_i, \text{ s.t. } (p_2, !!m, q_2) \in \Delta, (p_1, ?m, q_1) \in \Delta, m \in \Sigma\} \\ &\cup \{(q_1, q_2) \mid \text{there exists } p_2 \in Q \text{ s.t. } (q_1, p_2) \in S_i, \text{ and } (p_2, !!m, q_2) \in \Delta \text{ and } m \notin R(q_1)\} \\ &\cup \{(q_{in}, q) \mid \text{there exists } (q, q') \in S_i \text{ for some } q' \in Q\}. \end{aligned}$$

We then define $S = \bigcup_{n \in \mathbb{N}} S_n$, and $H = \{q \in Q \mid \text{there exists } q' \text{ and } (q, q') \in S\}$. Observe that $(S_i)_{i \in \mathbb{N}}$ is an increasing sequence bounded by $|Q|^2$. The computation reaches then a fixpoint and S can be computed in polynomial time. We define $H = \{q \mid \exists q' \in Q, (q, q') \in S\}$. Note that $H \subseteq Q_0 \cup Q_1^r$, as expected by Item 2 of Lemma 6.4. We also state that our construction is complete and correct, leading to the following theorem.

► **Theorem 6.5.** *COVER[Lines] is in P for k-phase-bounded protocols with $k \in \{1, 2\}$.*

Proof. We explain why the algorithm takes a polynomial time: step 1 (computing H) is done in polynomial time as explained above. For step 2, there are at most $|H| \times |H| \leq |Q|^2$ pairs, and for each pair, we explore a graph of at most $|Q|^5$ nodes in which each vertex represents a configuration $C = (\Gamma_5, L)$. Accessibility in a graph can be done non-deterministically in logarithmic space, and so in polynomial time. Observe that all the lemmas of this section hold true when considering 1-phase-bounded protocols, hence the theorem. ◀

References

- 1 B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parametrized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281. Springer-Verlag, 2014.
- 2 D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC'04*, pages 290–299. ACM, 2004.
- 3 M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Methods Comput. Sci.*, 7(4), 2011.
- 4 B. Bollig, M. Lehaut, and N. Sznajder. Round-bounded control of parameterized systems. In *ATVA'18*, volume 11138 of *Lecture Notes in Computer Science*, pages 370–386. Springer, 2018.
- 5 E. M. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR'04*, volume 3170 of *LNCS*, pages 276–291. Springer-Verlag, 2004.
- 6 G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR'10*, volume 6269 of *LNCS*, pages 313–327. Springer, 2010.
- 7 G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.
- 8 A. Durand-Gasselin, J. Esparza, P. Ganty, and R. Majumdar. Model checking parameterized asynchronous shared-memory systems. *Formal Methods Syst. Des.*, 50(2-3):140–167, 2017.

- 9 J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359. IEEE Computer Society, 1999.
- 10 J. Esparza, P. Ganty, J. Leroux, and R. Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017.
- 11 J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. *J. ACM*, 63(1):10:1–10:48, 2016.
- 12 J. Esparza, S. Jaax, M. A. Raskin, and C. Weil-Kennedy. The complexity of verifying population protocols. *Distributed Comput.*, 34(2):133–177, 2021.
- 13 S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.
- 14 L. Guillou, A. Sangnier, and N. Sznajder. Safety analysis of parameterised networks with non-blocking rendez-vous. In *CONCUR'23*, volume 279 of *LIPICs*, pages 7:1–7:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.
- 15 L. Guillou, A. Sangnier, and N. Sznajder. Phase-bounded broadcast networks over topologies of communication, 2024. [arXiv:2406.15202](https://arxiv.org/abs/2406.15202).
- 16 O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.
- 17 S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV'10*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.
- 18 R.J. Lipton. *The reachability problem requires exponential space*. Research report (Yale University. Department of Computer Science). Department of Computer Science, Yale University, 1976.
- 19 M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.
- 20 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
- 21 C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978.
- 22 G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.
- 23 W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.
- 24 S. Schmitz and P. Schnoebelen. The power of well-structured systems. In *CONCUR'13*, volume 8052 of *LNCS*, pages 5–24. Springer, 2013.

Inapproximability in Weighted Timed Games

Quentin Guilmant   

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Joël Ouaknine   

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We consider two-player, turn-based weighted timed games played on timed automata equipped with (positive and negative) integer weights, in which one player seeks to reach a goal location whilst minimising the cumulative weight of the underlying path. Although the *value problem* for such games (is the value of the game below a given threshold?) is known to be undecidable, the question of whether one can *approximate* this value has remained a longstanding open problem. In this paper, we resolve this question by showing that approximating arbitrarily closely the value of a given weighted timed game is computationally unsolvable.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Weighted timed games, approximation, undecidability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.27

Funding *Joël Ouaknine*: Also affiliated with Keble College, Oxford as *emmy.network* Fellow, and supported by DFG grant 389792660 as part of TRR 248.

1 Introduction

Weighted timed games are zero-sum games played by two players on a timed automaton equipped with weights, where one player seeks to reach a goal location whilst minimising the cumulative weight. Such games generalise *timed games*, which were introduced in the 1990s as a means to model open systems (whose behaviours are influenced by external environments), and to study controller-synthesis problems for real-time systems [19, 2, 17]. The introduction of numerical weights within the formalism of timed games, initiated independently in the early 2000s by Alur *et al.* [1] and Bouyer *et al.* [4], serves a dual purpose: first, it enables one to ascribe a quantitative quality measure to various controllers able to achieve a given objective, by computing the *value* of the corresponding game-theoretic strategy; and second, it allows one to model various resources (energy, bandwidth, memory, etc.) and associated costs incurred following a particular strategy. Here, one may choose to restrict weights to have either exclusively non-negative values, or both positive and negative values. The latter is useful when modelling resources that can both decrease and grow during an execution of the system, such as energy. Much of the early work in this area focussed on weighted timed games with non-negative weights, but over the last decade weighted timed games featuring arbitrary integer (or rational) weights have been fairly extensively studied.

Another important consideration in the modelling of real-time systems via weighted timed games is whether to adopt a *turn-based* or *concurrent* formalism. The former partitions discrete locations into those belonging to Player Min (representing the controller, which seeks to minimise the overall cumulative cost) and those belonging to Player Max (representing the environment). Concurrent games, on the other hand, enable both Min and Max to act at any given point and time (subject to the constraints imposed by the game). Both paradigms are well established. Concurrent games are strictly more expressive than their turn-based counterparts, but in general unfortunately suffer from not being determined (this



© Quentin Guilmant and Joël Ouaknine;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 27; pp. 27:1–27:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is in fact true even for unweighted timed games [15]). We focus on turn-based weighted timed games in the present paper; since concurrent games are at least as expressive, our main inapproximability result immediately carries over to the concurrent setting as well.

The central algorithmic problem concerning weighted timed games is the calculation of their *value*, i.e., the optimal cost assuming best play for each of the players. As noted earlier, we are exclusively considering *reachability* objectives in the present work: in other words, Player Min seeks to reach a specified goal location whilst minimising the cumulative weight of the underlying path, whereas Player Max seeks to prevent Min from reaching said goal location and, failing that, to extract as high a cost as possible in the process. Unfortunately, it has been known for some two decades that whether there exists a strategy for Player Min whose value is below a given threshold is an undecidable problem [7, 3]. A related (but subtly different) question, whether the optimal value of a weighted timed game falls below a given threshold (the so-called *value problem*), is also known to be undecidable [5]. These results hold even when restricting to turn-based games with exclusively non-negative weights. Nevertheless, various restrictions have been investigated in the literature, leading to decidability; see, e.g., [6, 22, 16, 10, 8, 11, 9, 20, 13].

The negative results cited above have spurred researchers to examine the *approximation problem* for weighted timed games: under what conditions, if any, can the value of a given weighted timed game be approximated arbitrarily closely? As noted in [13, Sec. 12], this is a “longstanding open problem”, and furthermore “the value of a weighted timed game could be *non approximable*, though we are not aware of any such game”. Bouyer *et al.* provided the first positive result in 2015, showing that the value of *almost strongly non-Zeno weighted timed games* (a class of turn-based games with weights in \mathbb{N} in which the weight of any cycle is either null or uniformly lower bounded) is approximable [5]. This result is all the more remarkable since the value problem for this class of games is undecidable. Busatto-Gaston *et al.* extended this line of work a couple of years later to the class of *divergent* and *almost-divergent weighted timed games* (in which the restriction to non-negative weights is lifted but additional mild conditions are imposed) [11, 12]. For a thorough overview of both the history and the state of the art concerning weighted timed games, we refer the reader to the recent and comprehensive article [13].

We are now in a position to state our main contribution:

► **Theorem 1.** *Given a two-player, turn-based, weighted timed game with (positive and negative) integer weights, the problem of approximating its value arbitrarily closely is computationally unsolvable.*

An important open problem is whether this result can be extended to timed games in which only non-negative integer weights are allowed. We return to this question in Sec. 4.

2 Weighted Timed Games

Let \mathcal{X} be a finite set of **clocks**. **Clock constraints** over \mathcal{X} are expressions of the form $x \sim n$ or $x - y \sim n$, where $x, y \in \mathcal{X}$ are clocks, $\sim \in \{<, \leq, =, \geq, >\}$ is a comparison symbol, and $n \in \mathbb{N}$ is a natural number. We write \mathcal{C} to denote the set of all clock constraints over \mathcal{X} . A **valuation** on \mathcal{X} is a function $\nu : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$. For $d \in \mathbb{R}_{\geq 0}$ we denote by $\nu + d$ the valuation such that, for all clocks $x \in \mathcal{X}$, $(\nu + d)(x) = \nu(x) + d$. Let $X \subseteq \mathcal{X}$ be a subset of all clocks. We write $\nu[X := 0]$ for the valuation such that, for all clocks $x \in X$, $\nu[X := 0](x) = 0$, and $\nu[X := 0](y) = \nu(y)$ for all other clocks $y \notin X$. For $C \subseteq \mathcal{C}$ a set of clock constraints over \mathcal{X} , we say that the valuation ν **satisfies** C , denoted $\nu \models C$, if and only if all the comparisons in C hold when replacing each clock x by its corresponding value $\nu(x)$.

► **Definition 2.** A (*turn-based*) *weighted timed game* is given by a tuple $\mathcal{G} = (L_{\text{Min}}, L_{\text{Max}}, G, \mathcal{X}, T, w)$, where:

- L_{Min} and L_{Max} are the (*disjoint*) sets of **locations** belonging to Players Min and Max respectively; we let $L = L_{\text{Min}} \cup L_{\text{Max}}$ denote the set of all locations. (In drawings, locations belonging to Min are depicted by blue circles, and those belonging to Max are depicted by red squares.)
- $G \subseteq L_{\text{Min}}$ are the **goal locations**.
- \mathcal{X} is a set of clocks.
- $T \subseteq (L \setminus G) \times 2^{\mathcal{C}} \times 2^{\mathcal{X}} \times L$ is a set of (**discrete**) **transitions**. A transition $\ell \xrightarrow{C, X} \ell'$ enables moving from location ℓ to location ℓ' , provided all clock constraints in C are satisfied, and afterwards resetting all clocks in X to zero.
- $w : (L \setminus G) \cup T \rightarrow \mathbb{Z}$ is a **weight function**.

In the above, we assume that all data (set of locations, set of clocks, set of transitions, set of clock constraints) are finite.

► **Remark 3.** The weight function w associates integer weights to each discrete transition and each non-goal location. It is worth pointing out that in our proof of inapproximability (Theorem 1), only *transitions* may carry negative weights; all *locations* have weights in \mathbb{N} .

Let $\mathcal{G} = (L_{\text{Min}}, L_{\text{Max}}, G, \mathcal{X}, T, w)$ be a weighted timed game. A **configuration** over \mathcal{G} is a pair (ℓ, ν) , where $\ell \in L$ and ν is a valuation on \mathcal{X} . Let $d \in \mathbb{R}_{\geq 0}$ be a **delay** and $t = \ell \xrightarrow{C, X} \ell' \in T$ be a discrete transition. One then has a **delayed transition** (or simply a **transition** if the context is clear) $(\ell, \nu) \xrightarrow{d, t} (\ell', \nu')$ provided that $\nu + d \models C$ and $\nu' = (\nu + d)[X := 0]$. Intuitively, control remains in location ℓ for d time units, after which it transitions to location ℓ' , resetting all the clocks in X to zero in the process. The **weight** of such a delayed transition is $d \cdot w(\ell) + w(t)$, taking account both of the time spent in ℓ as well as the weight of the discrete transition t .

As noted in [13], without loss of generality one can assume that no configuration (other than those associated with goal locations) is deadlocked; in other words, for any location $\ell \in L \setminus G$ and valuation $\nu \in \mathbb{R}_{\geq 0}^{\mathcal{X}}$, there exists $d \in \mathbb{R}_{\geq 0}$ and $t \in T$ such that $(\ell, \nu) \xrightarrow{d, t} (\ell', \nu')$.¹

Let $k \in \mathbb{N}$. A **run** ρ of length k over \mathcal{G} from a given configuration (ℓ_0, ν_0) is a sequence of matching delayed transitions, as follows:

$$\rho = (\ell_0, \nu_0) \xrightarrow{d_0, t_0} (\ell_1, \nu_1) \xrightarrow{d_1, t_1} \dots \xrightarrow{d_{k-1}, t_{k-1}} (\ell_k, \nu_k).$$

The **weight** of ρ is the cumulative weight of the underlying delayed transitions:

$$\text{weight}(\rho) = \sum_{i=0}^{k-1} (d_i \cdot w(\ell_i) + w(t_i)).$$

An infinite run ρ is defined in the obvious way; however, since no goal location is ever reached, its weight is defined to be infinite: $\text{weight}(\rho) = +\infty$.

A run is **maximal** if it is either infinite or cannot be extended further. Thanks to our deadlock-freedom assumption, finite maximal runs must end in a goal location. We refer to maximal runs as **plays**.

¹ In our setting, this can be achieved by adding unguarded transitions to a sink location for all locations controlled by Min and unguarded transitions to a goal location for the ones controlled by Max (noting that in all our constructions, Max-controlled locations always have weight 0). Nevertheless, in the pictorial representations of timed-game fragments that appear in this paper, in the interest of clarity we omit such extraneous transitions and locations; we merely assume instead that neither player allows him- or herself to end up in a deadlocked situation, unless a goal location has been reached.

We now define the notion of **strategy**. Recall that locations of \mathcal{G} are partitioned into sets L_{Min} and L_{Max} , belonging respectively to Players Min and Max. Let Player $P \in \{\text{Min}, \text{Max}\}$, and write $\mathcal{FR}_{\mathcal{G}}^P$ to denote the collection of all non-maximal finite runs of \mathcal{G} ending in a location belonging to Player P . A **strategy** for Player P is a mapping $\sigma_P : \mathcal{FR}_{\mathcal{G}}^P \rightarrow \mathbb{R}_{\geq 0} \times T$ such that for all finite runs $\rho \in \mathcal{FR}_{\mathcal{G}}^P$ ending in configuration (ℓ, ν) with $\ell \in L_P$, the delayed transition $(\ell, \nu) \xrightarrow{d,t} (\ell', \nu')$ is valid, where $\sigma_P(\rho) = (d, t)$ and (ℓ', ν') is some configuration (uniquely determined by $\sigma_P(\rho)$ and ν).

Let us fix a starting configuration (ℓ_0, ν_0) , and let σ_{Min} and σ_{Max} be strategies for Players Min and Max respectively (one speaks of a *strategy profile*). We write $\text{play}_{\mathcal{G}}((\ell_0, \nu_0), \sigma_{\text{Min}}, \sigma_{\text{Max}})$ to denote the unique maximal run starting from configuration (ℓ_0, ν_0) and unfolding according to the strategy profile $(\sigma_{\text{Min}}, \sigma_{\text{Max}})$: in other words, for every strict finite prefix ρ of $\text{play}_{\mathcal{G}}((\ell_0, \nu_0), \sigma_{\text{Min}}, \sigma_{\text{Max}})$ in $\mathcal{FR}_{\mathcal{G}}^P$, the delayed transition immediately following ρ in $\text{play}_{\mathcal{G}}((\ell_0, \nu_0), \sigma_{\text{Min}}, \sigma_{\text{Max}})$ is labelled with $\sigma_P(\rho)$.

Recall that the objective of Player Min is to reach a goal location through a play whose weight is as small possible. Player Max has an opposite objective, trying to avoid goal locations, and, if not possible, to maximise the cumulative weight of any attendant play. This gives rise to the following two symmetrical definitions:

$$\begin{aligned} \overline{\text{Val}}_{\mathcal{G}}(\ell_0, \nu_0) &= \inf_{\sigma_{\text{Min}}} \left\{ \sup_{\sigma_{\text{Max}}} \left\{ \text{weight}(\text{play}_{\mathcal{G}}((\ell_0, \nu_0), \sigma_{\text{Min}}, \sigma_{\text{Max}})) \right\} \right\} \text{ and} \\ \underline{\text{Val}}_{\mathcal{G}}(\ell_0, \nu_0) &= \sup_{\sigma_{\text{Max}}} \left\{ \inf_{\sigma_{\text{Min}}} \left\{ \text{weight}(\text{play}_{\mathcal{G}}((\ell_0, \nu_0), \sigma_{\text{Min}}, \sigma_{\text{Max}})) \right\} \right\}. \end{aligned}$$

$\overline{\text{Val}}_{\mathcal{G}}(\ell_0, \nu_0)$ represents the smallest possible weight that Player Min can possibly achieve, starting from configuration (ℓ_0, ν_0) , against best play from Player Max, and conversely for $\underline{\text{Val}}_{\mathcal{G}}(\ell_0, \nu_0)$: the latter represents the largest possible weight that Player Max can enforce, against best play from Player Min.² As noted in [13], turned-based weighted timed games are *determined*, and therefore $\overline{\text{Val}}_{\mathcal{G}}(\ell_0, \nu_0) = \underline{\text{Val}}_{\mathcal{G}}(\ell_0, \nu_0)$ for any starting configuration (ℓ_0, ν_0) ; we denote this common value by $\text{Val}_{\mathcal{G}}(\ell_0, \nu_0)$.

► **Remark 4.** Note that $\text{Val}_{\mathcal{G}}(\ell_0, \nu_0)$ can take on real numbers, or either of the values $-\infty$ and $+\infty$. Our proof of inapproximability, however, only makes use of games having finite values.

3 Inapproximability

3.1 Probabilistic Finite Automata

We establish value inapproximability for weighted timed games by reducing from an unsolvable approximation problem for probabilistic automata. We start with some definitions.

► **Definition 5 (PFA).** A (*two-letter*) **probabilistic automaton** is given by a tuple $\mathcal{A} = (Q, q_1, F, A_a, A_b)$, where:

- $Q = \{q_1, \dots, q_\ell\}$ is a finite set of **states**.
- $q_1 \in Q$ is the **initial state**.
- $F \subseteq Q$ are the **accepting states**.
- $A_a, A_b \in ([0, 1] \cap \mathbb{Q})^{\ell \times \ell}$ are left stochastic **transition matrices** corresponding to letters a and b respectively.³

² Technically speaking, these values may not be literally achievable; however given any $\varepsilon > 0$, both players are guaranteed to have strategies that can take them to within ε of the optimal value.

³ Left stochasticity means that each column of the matrix sums to 1.

Given such a probabilistic automaton \mathcal{A} , any word $w \in \{a, b\}^*$ induces a probability distribution on Q , as follows. For the empty word λ , we let the distribution $\mathbb{D}(\lambda) = (1, 0, \dots, 0)^T$, i.e., initially all the probability mass lies in the initial state q_1 . Suppose now that the distribution on Q upon reading word w is $\mathbb{D}(w)$, i.e., the probability $\mathbb{P}_w(q_i)$ of being in state q_i after reading w is precisely the i th component of $\mathbb{D}(w)$. We then let $\mathbb{D}(wa) = A_a \mathbb{D}(w)$ and $\mathbb{D}(wb) = A_b \mathbb{D}(w)$.

Finally, for any word $w \in \{a, b\}^*$, we write $\mathcal{A}(w) = \mathbb{P}_w(F) = \sum_{q \in F} \mathbb{P}_w(q)$ to denote the probability that the automaton \mathcal{A} accepts word w .

The key result we need (the main ingredient of which is due to Condon and Lipton [14]) is the following [18, Thm. 3.3]:

► **Theorem 6.** *There exists an algorithm which takes a Turing machine TM as input and outputs a two-letter probabilistic automaton \mathcal{A} satisfying the following:*

- *if TM does not accept the empty string, then \mathcal{A} accepts no word with probability exceeding $1/10$, and*
- *if TM does accept the empty string, then \mathcal{A} accepts some word with probability at least $1/2$.*

Theorem 6 states, in effect, that the maximum probability with which a given probabilistic automaton accepts some word cannot in general be approximated.⁴ In the remainder of this section, we show how to exploit this fact to establish that the value of a given weighted time game is, in turn, also not approximable in general.

3.2 Reduction Overview

Let probabilistic automaton $\mathcal{A} = (Q, q_1, F, A_a, A_b)$, with $Q = \{q_1, \dots, q_\ell\}$, be fixed for the rest of this paper. Without loss of generality, we may assume that all non-zero probabilistic transitions have weight $1/M$, for some constant $M \in \mathbb{N}$.⁵ In other words, $A_a, A_b \in \{0, 1/M\}^{\ell \times \ell}$.

Players **Min** and **Max** will play a weighted timed game \mathcal{G} representing the evolution of \mathcal{A} as it reads a word $w \in \{a, b\}^*$. **Min** will choose the letters of w , and will simulate running this word through \mathcal{A} , seeking to minimise the cumulative weight of the underlying path in \mathcal{G} . As long as **Min** faithfully simulates the behaviour of \mathcal{A} , the cumulative weight will remain constant. Any error or “cheating” by **Min**, however, will be “punishable” by **Max** in the form of an increase in the cumulative weight, the size of which will be proportional to the magnitude of the error. Naturally, as **Max** seeks to maximise the cumulative weight of the path, the dominant strategy for him will always be to seek to extract as large a cost as possible.

To this end, the game \mathcal{G} will be equipped with two sets of clocks:

- $Z = \{z_1, \dots, z_\ell\} \cup \{z_F\}$; intuitively, for $i \in \{1, \dots, \ell\}$, z_i is intended to store the current value of the probability of being in state q_i , and z_F is intended to store the probability of being in one of the accepting states in F .
- $X_{\text{Min}} = \{\mu_1, \dots, \mu_\ell\} \cup \{\mu_F\}$; intuitively, for $i \in \{1, \dots, \ell\}$, μ_i will store **Min**’s guess of the value to which to update clock z_i next, and likewise for μ_F and z_F .

In addition, \mathcal{G} has use of an auxiliary clock t to ensure proper synchronisation, etc.

⁴ Technically speaking, we should speak of a *supremum*.

⁵ This can straightforwardly be achieved via an increase in the number of states of \mathcal{A} , as follows. Take M to be the least common multiple of all denominators of all transition weights. For every state q of \mathcal{A} , create M fresh states, denoted q'_1, \dots, q'_M . And for each a -labelled transition $q \rightarrow s$ in \mathcal{A} with weight k/M , for all $1 \leq i \leq M$ and for all $1 \leq j \leq k$, create a fresh a -labelled transition $q'_i \rightarrow s'_j$ having weight $1/M$, and similarly for the letter b . The desired new matrices A_a and A_b are now obtained from these fresh states and transitions.

The game unfolds through a cycle of modules, as follows (see also Fig. 5):

1. Min chooses a letter (a or b) to append to the word that has been played so far.
2. Min compiles her guesses as to how the resulting probabilities of being in each state (q_1 to q_ℓ) should then be updated, and stores the corresponding values in clocks μ_1, \dots, μ_ℓ (and in μ_F for the collection of states in F). In so doing, the game infrastructure ensures that clocks z_1, \dots, z_ℓ and z_F remain untouched (their values at the beginning and at the end of the relevant module are the same).
3. Max extracts a cost (an increase to the cumulative weight) for every gap between the values guessed by Min and the actual freshly computed values of the clocks in Z , as per the transition matrices of \mathcal{A} .
4. The aforementioned gaps are then erased, by updating each of the clocks in Z to assume the value of its counterpart in X_{Min} .
5. Finally, before looping back, Min is given the opportunity to reach the goal location of \mathcal{G} ; this transition is however only available if $z_F \geq 1/2$.

► **Remark 7.** Note in the above that \mathcal{G} contains a transition in which a clock is compared to $1/2$ (rather than an integer). If desired, this is easily circumvented by considering an equivalent weighted timed game in which all constants have been multiplied by 2. We opted for the present half-integer formulation as this enables clock values directly to represent probabilities, rather than twice the corresponding probabilities.

Assuming that \mathcal{A} does accept some word w with probability at least $1/2$, Min need not make any error; she only has to guess correctly each letter of w in turn, along with the corresponding exact distribution updates, and eventually z_F will rise to $1/2$ or above, allowing her to reach the goal location at zero cumulative cost.

On the other hand, if no word is accepted by \mathcal{A} with probability exceeding $1/10$, then Min will be forced to make errors in order to enable μ_F , and thus in turn z_F , to reach $1/2$. Such errors will be punished by Max, extracting a total cumulative cost of at least $0.4M$. This is formalised in the following proposition, whose proof is deferred to Sec. 3.4.

► **Proposition 8.** *Let \mathcal{A} and \mathcal{G} be as above. Then:*

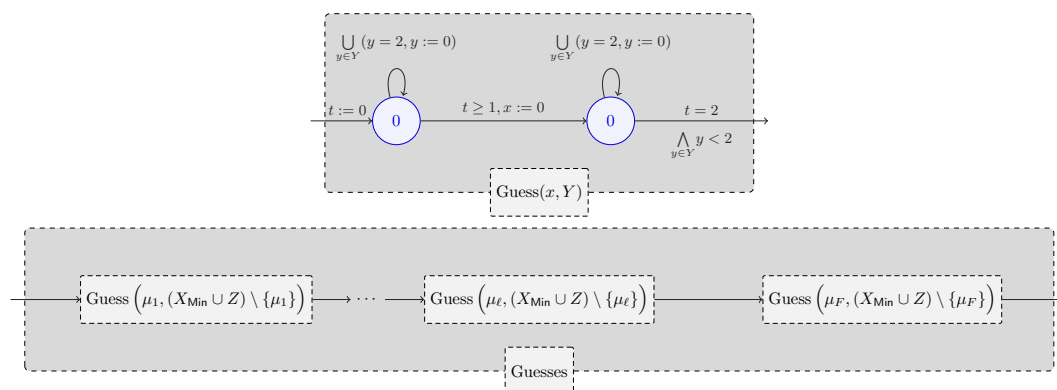
- *If there is a word $w \in \{a, b\}^*$ such that $\mathcal{A}(w) \geq 1/2$, then the value of \mathcal{G} is exactly 0.*
- *If, for all words $w \in \{a, b\}^*$, $\mathcal{A}(w) \leq 1/10$, then the value of \mathcal{G} is at least $0.4M$.*

Since approximating the value of \mathcal{G} to within $0.1M$ would enable, thanks to Theorem 6 and Proposition 8, to decide whether the Turing machine TM corresponding to \mathcal{A} halts or not, one concludes that weighted timed game values cannot in general be approximated, since otherwise one could solve the Halting Problem.

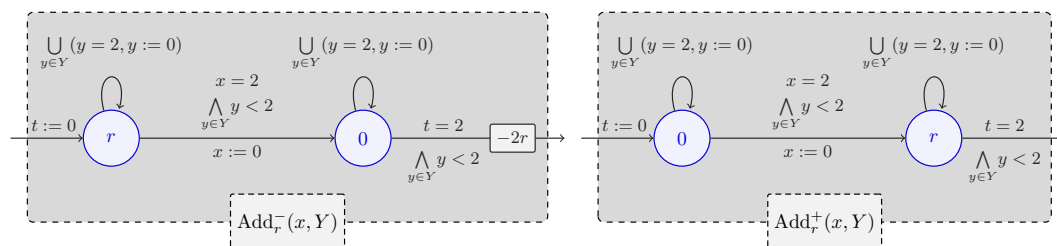
3.3 Modules and Widgets

We now describe a number of modules enabling us to implement the high-level protocol described in the previous section. In what follows, recall our assumption from Sec. 2, made without loss of generality, to the effect that neither player allows him- or herself to be deadlocked; in particular, if a clock constraint on a given transition requires the transition to be taken at a certain specific time, or within a certain time interval, for otherwise the run would deadlock (either immediately or shortly afterwards), then we assume the transition in question is indeed taken at the correct time.

We begin with the modules $\text{Guess}(x, Y)$ and Guesses , depicted in Fig. 1, which enable Min to set the clocks in X_{Min} to arbitrary values of her choosing in $[0, 1]$. Here x stands for an arbitrary clock, and Y for an arbitrary set of clocks not containing x .



■ **Figure 1** The guessing modules enable Min to set clocks μ_1, \dots, μ_ℓ and μ_F to arbitrary values of her choosing in $[0, 1]$, whilst leaving the values of clocks in Z unchanged. Recall that blue circles depict locations belonging to Player Min. The value 0 inside these circles, in module $\text{Guess}(x, Y)$, represents the weight of these locations (i.e., the rate at which the cumulative weight changes when control is in one of these locations). The notation $\cup_{y \in Y} (y = 2; y := 0)$ represents a collection of transitions, one for each $y \in Y$. Note that any such transition, when enabled (i.e., upon the corresponding clock $y \in Y$ reaching value 2), *must* instantly be taken, otherwise the guard on the transition exiting module $\text{Guess}(x, Y)$ could never be satisfied, and deadlock would ensue.



■ **Figure 2** Modules $\text{Add}_r^-(x, Y)$ and $\text{Add}_r^+(x, Y)$ enable to alter the cumulative weight by $-r\tilde{x}$ and $r\tilde{x}$ respectively, where \tilde{x} denotes the value of clock x upon entering the module and $r \in \mathbb{N}$ is a positive weight. Upon exiting the module, x as well as all clocks in Y have recovered their initial values. Note the negative weight of $-2r$ on the transition exiting module $\text{Add}_r^-(x, Y)$; this is the only place in our weighted timed game \mathcal{G} in which a negative weight is used.

The correctness of the following two statements is clear upon inspection; we therefore omit the proofs.

► **Lemma 9.** *Provided $x \notin Y$ and the initial values of all clocks in Y upon entering $\text{Guess}(x, Y)$ lie in $[0, 2)$, then upon exiting $\text{Guess}(x, Y)$ all clocks in Y have their respective initial values, x has value in $[0, 1]$, and the cumulative weight is unchanged.*

► **Corollary 10.** *Provided all clocks in X_{Min} and Z have values in $[0, 2)$ upon entering module Guesses , then upon exit the values of clocks in Z are unchanged, all clocks in X_{Min} have values in $[0, 1]$, and the cumulative weight is unchanged.*

We now introduce modules $\text{Add}_r^-(x, Y)$ and $\text{Add}_r^+(x, Y)$, depicted in Fig. 2. Here $r \in \mathbb{N}$ stands for a positive weight, x is a clock, and Y is a set of clocks not containing x . The role of these two modules is to alter the cumulative weight, as follows:

► **Lemma 11.** *Assume $x \notin Y$ and all clocks in $\{x\} \cup Y$ have values in $[0, 2)$ upon entering either $\text{Add}_r^-(x, Y)$ or $\text{Add}_r^+(x, Y)$. Let \tilde{x} denote the initial value of clock x . Then upon exiting $\text{Add}_r^-(x, Y)$, the cumulative weight has changed by $-r\tilde{x}$ (a decrease), whereas upon exiting $\text{Add}_r^+(x, Y)$, the cumulative weight has changed by $r\tilde{x}$ (an increase). Moreover, all clocks in $\{x\} \cup Y$ have recovered their initial values upon exiting either module.*

Once again, the statements are clear upon inspection.

We now turn to the payment modules, depicted in Fig. 3, which enable Player Max to extract a cost for guessing errors committed by Min. We first need to introduce some auxiliary definitions.

Given a state $q_i \in Q$, let $\text{in}_a(q_i)$ denote the set of all states $q_j \in Q$ such that there is an a -labelled non-null transition from q_j to q_i in \mathcal{A} (and recall, as noted in Sec. 3.2, that all such transitions have weight $1/M$). Formally, $\text{in}_a(q_i) = \{q_j \in Q \mid (A_a)_{i,j} = 1/M\}$. Overloading notation, write $\text{in}_a(F) = \cup_{q \in F} \text{in}_a(q)$. We define $\text{in}_b(q_i)$ and $\text{in}_b(F)$ in similar fashion.

We also define $\text{out}_a(q_i)$ symmetrically, representing the set of states q_j such that there is an a -labelled non-null transition from q_i to q_j , and similarly for $\text{out}_b(q_i)$.

For $q_i \in Q$, let $\text{clock}(q_i) = z_i$, and extend the clock function to sets of states in the obvious way: $\text{clock}(S) = \cup_{q \in S} \text{clock}(q)$.

We also consider the inverse function clock^{-1} , which associates to clock $z_i \in Z \setminus \{z_F\}$ the state $q_i \in Q$.

► **Lemma 12.** *Let $Y_1 = \{y_1, \dots, y_k\}$, Y_2 be two disjoint sets of clocks, and $x \notin Y_1 \cup Y_2$ be another clock. Let \tilde{x} and $\tilde{y}_1, \dots, \tilde{y}_k$ denote the initial values of clocks x and y_1, \dots, y_k respectively. Provided that all clocks in $\{x\} \cup Y_1 \cup Y_2$ have values in $[0, 2)$ upon entering $\text{Control}_M(x, Y_1, \Lambda, Y_2)$, the cumulative weight of this submodule is $\left| M\tilde{x} - \sum_{i=1}^k \tilde{y}_i \right|$. Moreover, upon exiting, all clocks (aside from t) have recovered their initial values.*

► **Corollary 13.** *For $\mu \in X_{\text{Min}}$ and $z \in Z$, let $\tilde{\mu}$ and \tilde{z} respectively denote the initial values of clocks μ and z upon entering module Pay_a . Assuming all clocks in $X_{\text{Min}} \cup Z$ have initial values in $[0, 2)$ upon entering Pay_a , then all clocks have recovered their initial values upon exiting Pay_a , and the cumulative weight has increased by*

$$\sum_{j=1}^{\ell} \left| M\tilde{\mu}_j - \sum_{i|q_i \in \text{in}_a(q_j)} \tilde{z}_i \right| + \left| M\tilde{\mu}_F - \sum_{i|q_i \in F} \sum_{j|q_j \in \text{in}_a(q_i)} \tilde{z}_j \right|.$$

By symmetry, an entirely similar assertion holds for module Pay_b .

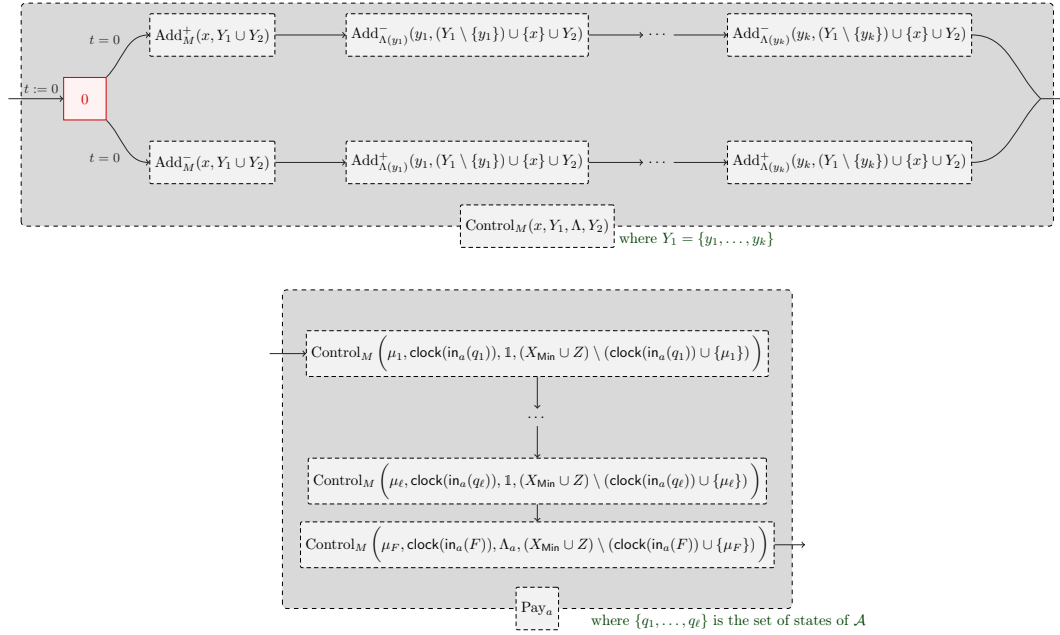
Both statements follow by inspection, making use of the previous assertions laid out in this section.

Our last module updates the values of clocks in Z to agree with their counterparts in X_{Min} , thereby erasing any gaps created by errors in Min's guesses, and setting the stage for a fresh cycle to play out; see Fig. 4.

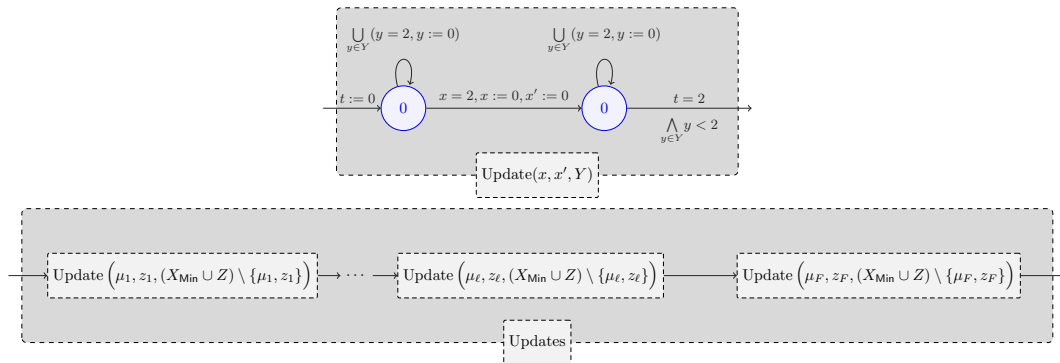
The following is immediate:

► **Lemma 14.** *Provided that all initial values of clocks in $\{x\} \cup Y$ lie in $[0, 2)$ upon entering module $\text{Update}(x, x', Y)$, then upon exiting all variables in $\{x\} \cup Y$ have recovered their initial values, x' agrees with x , and the cumulative weight remains unchanged.*

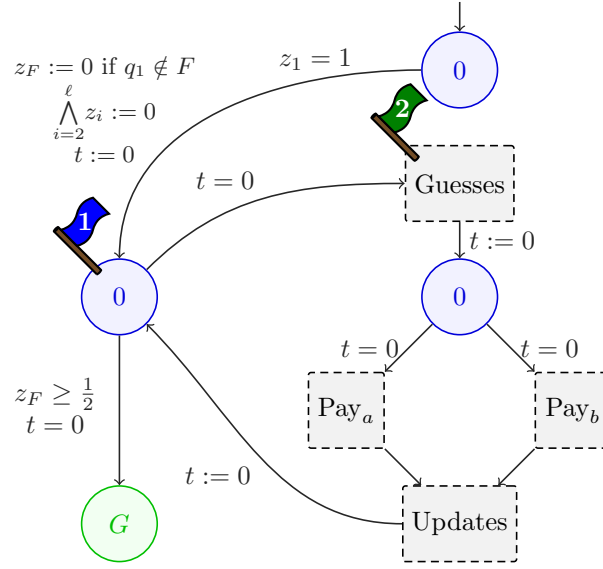
Likewise, assuming all initial values of clocks in $X_{\text{Min}} \cup Z$ lie in $[0, 2)$, module Updates preserves the values of all clocks in X_{Min} , does not alter the cumulative weight, and ensures that every clock in Z has the same value as its counterpart in X_{Min} upon exit.



■ **Figure 3** The payment modules, enabling Max to charge Min for guessing errors. x is a clock, and Y_1 and Y_2 are disjoint sets of clocks, neither of which contains x . Only Pay_a is depicted here; Pay_b is defined in entirely symmetrical fashion. The submodule $\text{Control}_M(x, Y_1, \Lambda, Y_2)$ is entered via a location represented as a red square, and hence belonging to Max, who can then choose between the upper and lower paths, whichever increases the cumulative weight (the two paths carry weights of equal magnitude but opposite signs). Note however that Max must act instantly upon entering submodule $\text{Control}_M(x, Y_1, \Lambda, Y_2)$, as the clock guard $t = 0$ would otherwise lead to deadlock. The function $\Lambda_a : Z \rightarrow \mathbb{N}$ is defined as follows: $\Lambda_a(z) = \#(\text{out}_a(\text{clock}^{-1}(z)) \cap F)$; in other words, Λ_a retrieves the state of \mathcal{A} corresponding to clock z (call it q), and counts how many non-null a -labelled transitions into F originate from q in \mathcal{A} . This is required in order to properly calculate the total probability of being in F upon reading letter a . The function $\mathbb{1}$ simply returns the value 1 on all inputs.



■ **Figure 4** The Updates module resets the value of each clock in Z to its counterpart in X_{Min} .



■ **Figure 5** The weighted timed game \mathcal{G} . The start location sits at the top, and all clocks have value 0 in the initial configuration. All three blue circles have null weight (or rate), and likewise all transitions appearing in the drawing carry null weight. The clock constraint $t = 0$ on edges forces an immediate transition to the next location. The goal state (in green, bottom left) is designated by the letter G . In order to reach it, clock z_f must have value at least $1/2$ in the preceding location.

3.4 The Reduction

Recall that we are given a probabilistic automaton $\mathcal{A} = (Q, q_1, F, A_a, A_b)$ with set of states $Q = \{q_1, \dots, q_\ell\}$, over alphabet $\{a, b\}$, with the property that every non-zero state transition carries probability exactly $1/M$ for some $M \in \mathbb{N}$. We are moreover promised that *either* \mathcal{A} accepts some word with probability at least $1/2$, *or* \mathcal{A} accepts no word with probability exceeding $1/10$.

Our corresponding weighted timed game \mathcal{G} is depicted in Fig. 5. As noted earlier, the convenient use of the half-integral constant $1/2$ in one of the clock constraints is easily circumvented if desired.

Recall the following proposition:

► **Proposition 8.** *Let \mathcal{A} and \mathcal{G} be as above. Then:*

- *If there is a word $w \in \{a, b\}^*$ such that $\mathcal{A}(w) \geq 1/2$, then the value of \mathcal{G} is exactly 0.*
- *If, for all words $w \in \{a, b\}^*$, $\mathcal{A}(w) \leq 1/10$, then the value of \mathcal{G} is at least $0.4M$.*

Proof. Consider a run of \mathcal{G} in which word $w = w_1 \dots w_n \in \{a, b\}^*$ has been played. Let $k \in \{1, \dots, n\}$, and for $i \in \{1, \dots, \ell\}$, let $\widetilde{z_{i,k}}$ and $\widetilde{z_{F,k}}$ be the respective values of clocks z_i and z_F upon exiting location for the k th time, and let $\widetilde{\mu_{i,k}}$ and $\widetilde{\mu_{F,k}}$ be the respective values of clocks μ_i and μ_F upon exiting module for the k th time.


By the lemmas and corollaries from the previous section, we have, for all i and k ,

$$\widetilde{\mu_{i,k}} = \widetilde{z_{i,k+1}}, \text{ and likewise } \widetilde{\mu_{F,k}} = \widetilde{z_{F,k+1}}. \quad (*)$$

Let us also introduce the following expressions:

$$E_{i,k} = \widetilde{z_{i,k}} - \mathbb{P}_{w_1 \dots w_{k-1}}(q_i) \quad \text{and} \quad E_{F,k} = \widetilde{z_{F,k}} - \mathbb{P}_{w_1 \dots w_{k-1}}(F),$$

$$\varepsilon_{i,k} = \widetilde{\mu_{i,k}} - \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \frac{\widetilde{z_{j,k}}}{M} \quad \text{and} \quad \varepsilon_{F,k} = \widetilde{\mu_{F,k}} - \sum_{i|q_i \in F} \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \frac{\widetilde{z_{j,k}}}{M}.$$

Intuitively, $E_{i,k}$ is the absolute cumulative error on the probability of being in state q_i after $k-1$ iterations, and $\varepsilon_{i,k}$ is the marginal error on this probability upon reading letter w_k . Finally, we let cost_k be the maximum cumulative weight that Player Max can achieve upon exiting state  for the k th time. For $k \in \{1, \dots, n\}$ and $i \in \{1, \dots, \ell\}$, we have:

$$\begin{aligned} E_{i,k+1} &= \widetilde{z_{i,k+1}} - \mathbb{P}_{w_1 \dots w_k}(q_i) \\ &= \widetilde{\mu_{i,k}} - \mathbb{P}_{w_1 \dots w_k}(q_i) && \text{(by (*))} \\ &= \varepsilon_{i,k} + \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \frac{\widetilde{z_{j,k}}}{M} - \mathbb{P}_{w_1 \dots w_k}(q_i) \\ &= \varepsilon_{i,k} + \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \frac{\widetilde{z_{j,k}}}{M} - \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \mathbb{P}_{w_1 \dots w_{k-1}}(q_j)(A_{w_k})_{i,j} \\ &= \varepsilon_{i,k} + \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \frac{\widetilde{z_{j,k}}}{M} - \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \frac{\mathbb{P}_{w_1 \dots w_{k-1}}(q_j)}{M} \\ E_{i,k+1} &= \varepsilon_{i,k} + \frac{1}{M} \sum_{j|q_j \in \text{in}_{w_k}(q_i)} E_{j,k}. && (\dagger) \end{aligned}$$

Similarly:


$$E_{F,k+1} = \varepsilon_{F,k} + \frac{1}{M} \sum_{q_i \in F} \sum_{j|q_j \in \text{in}_{w_k}(q_i)} E_{j,k}. \quad (\star)$$

Let us now consider the following properties for $k \in \{1, \dots, n+1\}$:

$$\mathcal{P}(k) : \quad \sum_{i=1}^{\ell} |E_{i,k}| \leq \sum_{i=1}^{\ell} \sum_{m=1}^{k-1} |\varepsilon_{i,m}|$$

$$\mathcal{Q}(k) : \quad \text{cost}_k = M \sum_{i=1}^{\ell} \sum_{m=1}^{k-1} |\varepsilon_{i,m}| + M \sum_{m=1}^{k-1} |\varepsilon_{F,m}|.$$

We prove both properties by induction.

- The base case is $k=1$. Since only the start location and location  have been visited, we have $\widetilde{z_{1,1}} = 1$, $\widetilde{z_{i,1}} = 0$ for $2 \leq i \leq \ell$, and $\widetilde{z_{F,1}} = 1$ if $q_1 \in F$, and $\widetilde{z_{F,1}} = 0$ otherwise. On the other hand, $\mathbb{P}_{\lambda}(q_1) = 1$, $\mathbb{P}_{\lambda}(q_i) = 0$ for $2 \leq i \leq \ell$, and $\mathbb{P}_{\lambda}(F) = 1$ if $q_1 \in F$, and $\mathbb{P}_{\lambda}(F) = 0$ otherwise. Therefore $E_{i,1} = 0$ for $1 \leq i \leq \ell$ and $E_{F,1} = 0$. Likewise, $\text{cost}_1 = 0$, whence $\mathcal{P}(1)$ and $\mathcal{Q}(1)$ hold.

27:12 Inapproximability in Weighted Timed Games

- Let $k \in \{1, \dots, n\}$, and assume that both $\mathcal{P}(k)$ and $\mathcal{Q}(k)$ hold. Thanks to Corollary 13, we have:

$$\begin{aligned} \text{cost}_{k+1} &= \text{cost}_k + \sum_{j=1}^{\ell} \left| M\widetilde{\mu}_{j,k} - \sum_{i|q_i \in \text{in}_{w_k}(q_j)} \widetilde{z}_{i,k} \right| + \left| M\widetilde{\mu}_{F,k} - \sum_{i|q_i \in F} \sum_{j|q_j \in \text{in}_{w_k}(q_i)} \widetilde{z}_{j,k} \right| \\ &= \text{cost}_k + M \left(\sum_{i=1}^{\ell} |\varepsilon_{i,k}| + |\varepsilon_{F,k}| \right) \\ &= M \sum_{i=1}^{\ell} \sum_{m=1}^k |\varepsilon_{i,m}| + M \sum_{m=1}^k |\varepsilon_{F,m}|, \text{ as required.} \end{aligned}$$

Also,

$$\begin{aligned} \sum_{i=1}^{\ell} |E_{i,k+1}| &\leq \sum_{i=1}^{\ell} |\varepsilon_{i,k}| + \frac{1}{M} \sum_{i=1}^{\ell} \sum_{j|q_j \in \text{in}_{w_k}(q_i)} |E_{j,k}|, \text{ (by } (\dagger)) \\ &\leq \sum_{i=1}^{\ell} |\varepsilon_{i,k}| + \frac{1}{M} \sum_{i,j|(A_{w_k})_{i,j}=1/M} |E_{j,k}| \\ &\leq \sum_{i=1}^{\ell} |\varepsilon_{i,k}| + \frac{1}{M} \sum_{i=1}^{\ell} \sum_{j|q_j \in \text{out}_{w_k}(q_i)} |E_{i,k}|. \end{aligned}$$

By our assumption on \mathcal{A} , each state has exactly M non-null outgoing transitions for each letter. Therefore

$$\sum_{i=1}^{\ell} |E_{i,k+1}| \leq \sum_{i=1}^{\ell} |\varepsilon_{i,k}| + \frac{1}{M} \sum_{i=1}^{\ell} M |E_{i,k}|.$$

Applying $\mathcal{P}(k)$,

$$\sum_{i=1}^{\ell} |E_{i,k+1}| \leq \sum_{i=1}^{\ell} |\varepsilon_{i,k}| + \sum_{i=1}^{\ell} \sum_{m=1}^{k-1} |\varepsilon_{i,m}| = \sum_{i=1}^{\ell} \sum_{m=1}^k |\varepsilon_{i,m}|,$$

and therefore $\mathcal{P}(k+1)$ holds, concluding the induction step.

Now if w is such that $\mathcal{A}(w) = \mathbb{P}_{w_1 \dots w_k}(F) \geq 1/2$, Player Min need only correctly set each clock μ_i to its expected value in every iteration, so that, for all $i \in \{1, \dots, \ell\}$ and all $k \in \{1, \dots, n+1\}$, we have $\varepsilon_{i,k} = 0$ and $\varepsilon_{F,k} = 0$. By $\mathcal{Q}(n+1)$, the value of \mathcal{G} is at most 0. Since it is easily seen that the value of \mathcal{G} cannot be negative, it must indeed be precisely 0.

If, on the other hand, $\mathcal{A}(w) \leq 1/10$, then in order for Min to reach the goal state after playing w , it is necessary to have $E_{F,n+1} \geq 1/2 - 1/10 = 0.4$. Using (\star) ,

$$\begin{aligned} 0.4 &\leq \varepsilon_{F,n+1} + \frac{1}{M} \sum_{q_i \in F} \sum_{j|q_j \in \text{in}_{w_n}(q_i)} E_{j,n+1} \\ &\leq |\varepsilon_{F,n+1}| + \frac{1}{M} \sum_{q_i \in F} \sum_{j|q_j \in \text{in}_{w_n}(q_i)} |E_{j,n+1}| \\ &\leq |\varepsilon_{F,n+1}| + \frac{1}{M} \sum_{i,j|(A_{w_n})_{i,j}=1/M \wedge q_i \in F} |E_{j,n+1}|. \end{aligned}$$

Recall that each state of \mathcal{A} has exactly M non-null outgoing transitions for each letter, and thus at most M w_n -labeled outgoing transitions to a final state. We then get

$$0.4 \leq |\varepsilon_{F,n+1}| + \sum_{j=1}^{\ell} |E_{j,n+1}|.$$

Using $\mathcal{P}(n+1)$,

$$0.4 \leq |\varepsilon_{F,n+1}| + \sum_{i=1}^{\ell} \sum_{m=1}^n |\varepsilon_{i,m}| \leq \sum_{i=1}^{\ell} \sum_{m=1}^n |\varepsilon_{i,m}| + \sum_{m=1}^n |\varepsilon_{F,m}|,$$

whence, using $\mathcal{Q}(n+1)$,

$$0.4M \leq \text{cost}_{n+1}.$$

This is true for any word played. Therefore if no word is accepted by \mathcal{A} with probability exceeding 0.1, the value of \mathcal{G} must be at least $0.4M$, as claimed. \blacktriangleleft

Our main result now immediately follows:

► **Theorem 1.** *Given a two-player, turn-based, weighted timed game with (positive and negative) integer weights, the problem of approximating its value arbitrarily closely is computationally unsolvable.*

4 Conclusion

We have shown that the problem of approximating the value of weighted timed games with positive and negative weights is computationally unsolvable. An obvious question is whether this result can be extended to games only making use of non-negative weights. This appears to be rather difficult. Negative weights play a critical role in our construction in enabling us, thanks to modules $\text{Add}_r^-(x, Y)$ and $\text{Add}_r^+(x, Y)$ (depicted in Fig. 2), to keep a cumulative tally of the costs incurred through the repeated commission of small errors (or “cheating”) by Player Min. Without the use of negative weights, it does not seem possible to implement a “punishing” mechanism for Player Max that can be re-used arbitrarily many times, and accordingly we conjecture that in this case, the value of such games can be approximated arbitrarily closely.

A related observation is that both the above modules require the passage of two full time units to execute properly. It follows that over bounded time, one would need to implement a different approach. We conjecture that the value problem for weighted timed games is undecidable *even over bounded time*, but however that the corresponding time-bounded approximation problem is solvable.⁶

⁶ Here “bounded time” refers to the requirement that there be some global constant T such that all plays are required to have total duration at most T . Various undecidable real-time problems are known to become decidable in a time-bounded setting [21].

References

- 1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.
- 2 Rajeev Alur and Thomas A. Henzinger. Modularity for timed and hybrid systems. In *CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 1997.
- 3 Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Inf. Process. Lett.*, 98(5):188–194, 2006.
- 4 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim Guldstrand Larsen. Optimal strategies in priced timed game automata. In *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.
- 5 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In *CONCUR*, volume 42 of *LIPICs*, pages 311–324. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.
- 6 Patricia Bouyer, Kim Guldstrand Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in one clock priced timed games. In *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.
- 7 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *FORMATS*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.
- 8 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefauchaux, and Benjamin Monmege. Simple priced timed games are not that simple. In *FSTTCS*, volume 45 of *LIPICs*, pages 278–292. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.
- 9 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefauchaux, and Benjamin Monmege. One-clock priced timed games with negative weights. *Log. Methods Comput. Sci.*, 18(3), 2022.
- 10 Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. Adding negative prices to priced timed games. In *CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 560–575. Springer, 2014.
- 11 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Optimal reachability in divergent weighted timed games. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 162–178, 2017.
- 12 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Symbolic approximation of weighted timed games. In *FSTTCS*, volume 122 of *LIPICs*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 13 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Optimal controller synthesis for timed systems. *Log. Methods Comput. Sci.*, 19(1), 2023.
- 14 Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *FOCS*, pages 462–467. IEEE Computer Society, 1989.
- 15 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003.
- 16 Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In *CONCUR*, volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013.
- 17 Thomas A. Henzinger, Benjamin Horowitz, and Rupak Majumdar. Rectangular hybrid games. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 1999.
- 18 Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1-2):5–34, 2003.
- 19 Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995.

- 20 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Decidability of one-clock weighted timed games with arbitrary weights. In *CONCUR*, volume 243 of *LIPICs*, pages 15:1–15:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 21 Joël Ouaknine, Alexander Rabinovich, and James Worrell. Time-bounded verification. In *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 496–510. Springer, 2009.
- 22 Michal Rutkowski. Two-player reachability-price games on single-clock timed automata. In *QAPL*, volume 57 of *EPTCS*, pages 31–46, 2011.

Faster and Smaller Solutions of Obliging Games

Daniel Hausmann  

University of Gothenburg, Gothenburg, Sweden
Chalmers University of Technology, Gothenburg, Sweden
University of Liverpool, Liverpool, United Kingdom

Nir Piterman  

University of Gothenburg, Gothenburg, Sweden
Chalmers University of Technology, Gothenburg, Sweden

Abstract

Obliging games have been introduced in the context of the game perspective on reactive synthesis in order to enforce a degree of cooperation between the to-be-synthesized system and the environment. Previous approaches to the analysis of obliging games have been small-step in the sense that they have been based on a reduction to standard (non-obliging) games in which single moves correspond to single moves in the original (obliging) game. Here, we propose a novel, large-step view on obliging games, reducing them to standard games in which single moves encode long-term behaviors in the original game. This not only allows us to give a meaningful definition of the environment winning in obliging games, but also leads to significantly improved bounds on both strategy sizes and the solution runtime for obliging games.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Two-player games, reactive synthesis, Emerson-Lei games, parity games

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.28

Related Version *Extended Version*: <https://arxiv.org/abs/2407.11856>

Funding Both authors supported by the ERC Consolidator grant D-SynMA (No. 772459).

Daniel Hausmann: Supported by the EPSRC through grant EP/Z003121/1.

1 Intro

Infinite duration games [11] and their analysis are central to various logical problems in computer science; problems with existing game reductions subsume model checking [5, 16] and satisfiability checking [9, 10] for temporal logics (such as CTL or the μ -calculus), or reactive synthesis for LTL specifications [8]. Game arenas that are used in such reductions typically incorporate two antagonistic players (called player \exists and player \forall in the current work) that have dual objectives. Then the reductions construct game arenas and objectives in such a way that the instance of the original problem has a positive answer if and only if player \exists has a strategy that ensures that the player's objective is satisfied (that is, player \exists *wins* the game). *Solving* games then amounts to determining their winner. Games with Borel objectives are known to be *determined* [21], that is, for each node in them, exactly one of the players i has a winning strategy of type $V^*V_i \rightarrow V$, where V is the set of game nodes, and V_i the set of nodes controlled by player i .

Reactive synthesis [22] considers a setting in which a system works within an antagonistic environment, and the system-environment interaction is modelled by means of input variables from a set I (controlled by the environment), and output variables from a set O (controlled by the system). The synthesis problem then takes a logical specification φ over the variables $I \cup O$ as input and asks for the automated construction of a system in which all interactions between the system and the environment satisfy the specification; if such a system exists, then φ is said to be *realizable*. The reactive synthesis problem therefore goes beyond checking realizability by also asking for a witnessing system in the case that the input specification is realizable. While the problem has been shown to be 2EXPTIME-complete for specifications that are



© Daniel Hausmann and Nir Piterman;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 28; pp. 28:1–28:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

formulated in LTL, a landscape of algorithms and implementations has been developed that shows good performance in (some) practical use cases (e.g. [18, 23]). The feasibility of these algorithms is largely owed to an underlying reduction to infinite-duration two-player games, most commonly with parity objectives. Answering the realizability problem then corresponds to deciding the winner of the reduced game, while the construction of a witnessing system for a realizable specification corresponds to the extraction of a winning strategy from that game. This motivates the interest not only in game solving algorithms, but also in the analysis and extraction of winning strategies. In particular, the amount of memory required by winning strategies in the types of games at hand determines the minimum size of witnessing systems.

Building on the described game perspective on reactive synthesis, *obliging games* have been proposed to deal with the situation that a system might trivially realize a specification by disallowing most or all interactions with the environment (cf. [2, 6]). Obliging games address this problem by requiring the system player to have a strategy that not only always guarantees that the system’s strong objective (say α_S) is achieved, but to also always keep an interaction possible in which intuitively both players cooperate to achieve a second, weak, objective (say α_W). Such a strategy then is called *graciously winning* for the system player. For example, the generalized reactivity (GR[1]) setting (cf. [4]) incorporates k different *requests* and k corresponding *grants* (R_i, G_i for $1 \leq i \leq k$). Then the objective α_S states that “if all R_i (requests) hold infinitely often, then all G_i (grants) hold infinitely often”. In this setting, player \exists may satisfy the objective α_S trivially by simply ensuring that, for each interaction, there is some R_i that is eventually avoided forever. The obliging game setting allows to address this situation by taking α_W to require that all R_i hold infinitely often. Then a gracious strategy for player \exists has to ensure α_S (possibly by avoiding, on most interactions, some R_i), but it also has to enable player \forall to additionally realize α_W , thereby always enabling at least one “interesting” interaction on which all R_i and also all G_i hold infinitely often.

Previous approaches to the analysis of obliging games [6, 20] have been largely based on a reduction to equivalent non-obliging games in which the players still take single steps on the original game graph, but in addition to that, game nodes are annotated with auxiliary memory that is used to deal with the more involved obliging game’s objectives. Obliging GR[1] games, as in the example above, have been considered under the names of “cooperative” [3] and “environmentally-friendly” [19]. Independent bespoke symbolic algorithms of slightly better complexity than the general solution have been suggested.

In this context, the contributions of the current work are threefold:

- We propose a novel perspective on obliging games that is based on considering multi-step strategies of players in the original game, rather than emulating single-step moves. In more detail, we provide an alternative reduction that takes obliging games to equivalent non-obliging games in which the system player commits to certain long-term behaviors, encoded by so-called witnesses, which are just plays of the original game (we therefore call the resulting games *witness games*). The environment player in turn can either check whether a given witness is valid, that is, satisfies both α_S and α_W , or accept the witness and exit it at any game node that occurs in the witness, thereby intuitively challenging the system player to still win when a play only traverses the witness up to the exit node and then continues on outside of the proposed witness; in the latter case, the system player has to provide a new witness for the challenged game node, and so on. We use the reduction to witness games to show determinacy of obliging games with Borel objectives (however, with respect to strategies of type $V^* \rightarrow V^\omega$ for the system player, and strategies of type $V^\omega \rightarrow V \cup V^\omega$ for the environment player).

- We show that witnesses for obliging games with ω -regular objectives α_S and α_W have finite representations, as they correspond to (accepting) runs of ω -automata with acceptance condition $\alpha_S \wedge \alpha_W$; we call such representations *certificates*. Using certificates in place of infinite witnesses, we modify witness games to obtain an alternative reduction that takes ω -regular obliging games to *finite* ω -regular non-obliging games, called *certificate games*. Technically, we present the reduction for obliging games with Emerson-Lei objectives. During the correctness proof for this reduction, we show that the memory requirements of graciously winning strategies for Emerson-Lei obliging games depend only linearly on the size of α_W , improving significantly upon existing upper bounds [6, 20] that are, in general, exponential in the size of α_W .
- The certificate games that we propose contain an explicit game node for any possible certificate within an obliging game. Hence they are prohibitively large and it is not viable to solve them naively. However, we show how a technique of fixpoint acceleration can be used to speed up the solving process of certificate games, replacing the exploration of all certificate nodes with emptiness checking for suitable ω -automata; this technique solves certificate games by computing nested fixpoints of a function that checks for the existence of suitable certificates. Thereby we are able to improve previous upper runtime bounds for the solution of obliging games; in many cases, our algorithm outperforms existing algorithms by an exponential factor.

Summing up, we propose a novel approach to the analysis of obliging games that provides more insight in their determinacy, and for obliging games with Emerson-Lei objectives, we significantly improve existing upper bounds both on strategy sizes and solution time.

Structure. We introduce obliging games and related notions in Section 2. In Section 3, we reduce obliging games to witness games and use the reduction to show that obliging games are determined (for strategies with additional information). Subsequently, we restrict our attention to ω -regular obliging games with Emerson-Lei objectives. In Section 4 we reduce witness games with such objectives to certificate games and use the latter to obtain improved upper bounds on strategy sizes in obliging games. In Section 5 we show how certificate games can be solved efficiently, in consequence improving previous upper bounds on the runtime complexity of solving obliging games. Full proofs and additional details can be found in an extended version of this paper.

2 Preliminaries

We start by recalling obliging games and extend the setup from previous work to use general Borel objectives in place of Muller objectives; we also introduce the special case of obliging games with Emerson-Lei objectives, and recall the definition of Emerson-Lei automata.

Arenas, plays, strategies. An *arena* is a graph $A = (V, V_{\exists}, E)$, consisting of a set V of *nodes* and a set $E \subseteq V \times V$ of *moves*; furthermore, we assume that the set of nodes is partitioned into the sets V_{\exists} and $V_{\forall} := V \setminus V_{\exists}$ of nodes *owned* by player \exists and by player \forall , respectively. We write $E(v) = \{w \in V \mid (v, w) \in E\}$ for the set of nodes reachable from node $v \in V$ by a single move. We assume without loss of generality that every node has at least one outgoing edge, that is, that $E(v) \neq \emptyset$ for all $v \in V$. A *play* $\pi = v_0 v_1 \dots$ on A is a (finite or infinite) sequence of nodes such that $v_{i+1} \in E(v_i)$ for all $i \geq 0$. The length $|\pi| = n + 1$ of a finite play $\pi = v_0 v_1 \dots v_n$ is the number of vertices it contains; throughout, we denote the set $\{0, \dots, n\}$ for $n \in \mathbb{N}$ by $[n]$. By abuse of notation, we denote by A^ω the set of infinite plays on A and by A^* and A^+ the set of finite (nonempty) plays on A . A *strategy*

for player $i \in \{\exists, \forall\}$ is a function $\sigma : A^* \cdot V_i \rightarrow V$ that assigns a node $\sigma(\pi v) \in V$ to every finite play πv that ends in a node $v \in V_i$. A strategy σ for player i is said to be *positional* if the moves that it prescribes do not depend on previously visited game nodes. Formally this is the case if we have $\sigma(\pi v) = \sigma(\pi' v)$ for all $v \in V_i$ and all $\pi, \pi' \in A^*$. A play $\pi = v_0 v_1 \dots$ is *compatible* with a strategy σ for player $i \in \{\exists, \forall\}$ if for all $j \geq 0$ such that $v_j \in V_i$, we have $v_{j+1} = \sigma(v_0 v_1 \dots v_j)$.

Objectives and games. In this work we consider two types of objectives: *Borel objectives* and *Emerson-Lei objectives*. Borel objectives are explicit sets of infinite sequences of vertices. A set is *Borel definable* if it is in the σ -algebra over the open subsets of infinite sequences of vertices V^ω . That is, sets that can be obtained by countable unions, countable intersections, and complementations from the open sets (sets of the form wV^ω for $w \in V^*$). A play π on A *satisfies* a Borel objective B if $\pi \in B$.

Emerson-Lei objectives are specified relative to a coloring function $\gamma_C : E \rightarrow 2^C$ (for some set C of colors) that assigns a set $\gamma_C(v, w) \subseteq C$ of colors to every move $(v, w) \in E$; we note that our setup is more succinct than the one from [6] where each edge has (at most) one color. A play $\pi = v_0 v_1 \dots$ then induces a sequence $\gamma_C(\pi) = \gamma_C(v_0, v_1) \gamma_C(v_1, v_2) \dots$ of sets of colors. Emerson-Lei objectives are given as positive Boolean formulas $\varphi_C \in \mathbb{B}_+(\{\text{Inf } c, \text{Fin } c \mid c \in C\})$ over atoms of the shape $\text{Inf } c$ and $\text{Fin } c$. Such formulas are interpreted over infinite sequences $\gamma_0 \gamma_1 \dots$ of sets of colors. We put $\gamma_0 \gamma_1 \dots \models \text{Inf } c$ if and only if there are infinitely many positions i such that $c \in \gamma_i$, and $\gamma_0 \gamma_1 \dots \models \text{Fin } c$ if there are only finitely many such positions; satisfaction of Boolean operators is defined in the usual way. Then an infinite play π on A *satisfies* the formula φ_C if and only if $\gamma_C(\pi) \models \varphi_C$ and we define the Emerson-Lei objective induced by γ_C and φ_C by putting

$$\alpha_{\gamma_C, \varphi_C} = \{\pi \in A^\omega \mid \gamma_C(\pi) \models \varphi_C\}.$$

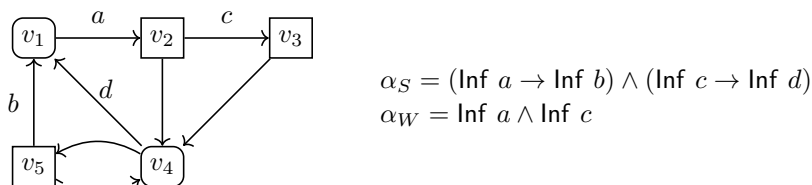
Parity objectives are a special case of Emerson-Lei objectives with set $C = \{p_0, \dots, p_k\}$ of colors, where each edge has exactly one color (also called *priority*), and where $\varphi = \bigvee_{i \text{ even}} \text{Inf } p_i \wedge \bigwedge_{i < j \leq k} \text{Fin } p_j$, stating that the maximal priority that is visited infinitely often has an even index. We can denote such objectives by just a single function $\Omega : E \rightarrow [k]$. Further standardly used conditions include *Büchi*, *generalized Büchi*, *generalized reactivity* (*GR[1]*), *Rabin* and *Streett* objectives, all of which are special cases of Emerson-Lei objectives (see e.g. [15]); the memory required for winning in such games has been investigated in [7].

We note that neither Borel nor Emerson-Lei objectives enable a simple distinction between finite plays ending in existential and universal nodes; hence we will avoid deadlocks in our game reductions, ensuring that all games in this work allow only infinite plays.

A *standard game* is a tuple (A, α) , where $A = (V, V_\exists, E)$ is an arena and α is an objective. A strategy σ is *winning* for player \exists at some node $v \in V$ if all plays that start at v and are compatible with σ satisfy the objective α . A strategy τ for player \forall is defined winning dually.

An *obliging game* is a tuple (A, α_S, α_W) , consisting of an arena $A = (V, V_\exists, E)$ together with two objectives α_S and α_W , called the *strong* and *weak* objective, respectively; we also refer to such games as α_S / α_W obliging games. A strategy σ for player \exists is *graciously winning* for $v \in V$ if all plays that start at v and are compatible with σ satisfy the strong objective α_S and furthermore every finite prefix $\pi \in A^*$ of a play that is compatible with σ can be extended to an infinite play $\pi\tau$ that is compatible with σ and satisfies the weak objective α_W . We sometimes refer to the infinite plays $\pi\tau$ witnessing the satisfaction of the weak objective as *obliging* plays. It follows immediately that player \exists can only win graciously at nodes at which at least one obliging play (satisfying $\alpha_S \wedge \alpha_W$) starts, so we assume without loss of generality that this is the case for all nodes in V .

► **Example 1.** We consider the Emerson-Lei obliging game depicted below with the set $\{a, b, c, d\}$ of colors, a Streett condition (with two pairs (a, b) and (c, d)) as strong objective α_S , and generalized Büchi condition enforcing visiting both a and c as weak objective α_W . In all examples in this work, nodes belonging to player \exists are depicted with rounded corners, while \forall -nodes are depicted by rectangles; edges may have several colors in Emerson-Lei games, but in this example, each edge has at most one color. We use the dashed edge to illustrate both winning and losing in an obliging game.



Consider the strategy σ with which player \exists alternately moves to v_5 and to v_1 when node v_4 is reached, depending on where they moved from v_4 the last time it has been visited. Without the dashed edge, σ is graciously winning: every play compatible with σ visits the colors a, b and d infinitely often and hence satisfies α_S ; also, σ allows player \forall to visit color c arbitrarily often by moving from v_2 to v_3 , so every finite prefix of a σ -play can be continued to an obliging σ -play that infinitely often visits v_3 and therefore satisfies $\alpha_S \wedge \alpha_W$. If the dashed edge is added to the arena, then σ is no longer graciously winning. Indeed, when playing against σ , player \forall can prevent b from ever being visited by always moving from v_5 to v_4 . However, the modified strategy σ' that moves from v_4 to v_1 only if the last visited node is *not* v_5 and also b has been visited more recently than d (and otherwise moves back to v_5) is graciously winning. Every σ' -play that ends in $(v_4 v_5)^\omega$ satisfies α_S but also can be made into an obliging play by having \forall move to v_1 whenever v_5 is reached.

Emerson-Lei automata. Given an Emerson-Lei objective $\alpha_{\gamma_C, \varphi_C}$, an *Emerson-Lei automaton* is a tuple $A = (\Sigma, Q, \delta, q_0, \alpha_{\gamma_C, \varphi_C})$, where Σ is the alphabet, Q is a set of *states*, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $q_0 \in Q$ is the initial state; in this context, we assume that $\gamma_C : \delta \rightarrow 2^C$ assigns sets of colors to transitions in A . A *run* of A on some infinite word $w = a_0 a_1 \dots \in \Sigma^\omega$ is a sequence $\pi = q_0 q_1 \dots \in Q^\omega$ such that $(q_i, a_i, q_{i+1}) \in \delta$ for all $i \geq 0$. A run π is *accepting* if and only if $\gamma_C(\pi) \models \varphi_C$, and A *recognizes* the language

$$L(A) = \{w \in \Sigma^\omega \mid \text{there is an accepting run of } A \text{ on } w\}.$$

An Emerson-Lei automaton A is *non-empty* if and only if $L(A) \neq \emptyset$. All automata that we consider in this work will have a single-letter alphabet $\Sigma = \{*\}$ so that they can read just the single infinite word $*^\omega$.

3 Determinacy of Obliging Games

Chatterjee et al. [6], when formalizing obliging games, stated that the standard shape of strategies (that is, $A^* \cdot V_i \rightarrow V$) does not allow player \forall to counteract player \exists 's strategy with a single strategy. We show that, for a more general form of strategy (of type $A^\omega \rightarrow V \cup A^\omega$), this is possible. Furthermore, with these more general strategies, the games are determined. That is, players are not disadvantaged by revealing their strategy first and one of the players always has a winning strategy. This insight allows offering alternative (more efficient) solutions to obliging games in the next sections.

Fix an obliging game $G = (A, \alpha_S, \alpha_W)$. Let $A = (V, V_{\exists}, E)$ and let α_S and α_W be Borel sets. A *witness* for vertex $v \in V$ is an infinite play $\pi = v_0 v_1 \dots \in A^\omega$ such that $v_0 = v$; we write $\text{witness}(v)$ to denote the set of witnesses for $v \in V$. The *witness game* $W(G) = (A', \alpha')$ captures the obliging game by allowing player \exists to choose an explicit witness for a given vertex. Player \forall can then either check whether the witness satisfies both α_S and α_W , or stop at some point verifying the witness and ask for a new witness. If player \forall changes witnesses infinitely often, they still check that the resulting play satisfies α_S .

Formally, we have $A' = (V', V'_{\exists}, E')$, where $V' = V \cup A^\omega$, $V'_{\exists} = V$ and E' is as follows.

$$E' = \{(v, \pi) \mid v \in V, \pi \in \text{witness}(v)\} \cup \{(v\pi, \pi) \mid v\pi \in A^\omega\} \cup \{(v\pi, v'') \mid v\pi \in A^\omega, v \in V_{\forall}, v'' \in E(v)\}.$$

Given $v\pi \in V'_{\forall}$ put $\text{head}(v\pi) = v$. We extend head to sequences over V' in the natural way. Consider a play $\pi' \in (A')^\omega$. Let $\pi' \downarrow_{V'_{\forall}}$ denote the projection of π' to the elements of V'_{\forall} , that is, $\pi' \downarrow_{V'_{\forall}}$ is obtained from π' by removing all elements in $V'_{\exists} = V$. Then put $\text{seq}_A(\pi') = \text{head}(\pi' \downarrow_{V'_{\forall}})$. Clearly, $\text{seq}_A(\pi') \in A^\omega$, that is, $\text{seq}_A(\pi')$ extracts from π' the infinite play in A that is followed in π' . The winning condition α' consists of plays π' that remain eventually in V'_{\forall} forever such that $\text{seq}_A(\pi')$ satisfies both α_S and α_W , or plays π' that visit V'_{\exists} infinitely often such that $\text{seq}_A(\pi')$ satisfies α_S . Formally, we have the following:

$$\alpha' = \{\pi' \in V'^* \cdot (V'_{\forall})^\omega \mid \text{seq}_A(\pi') \in \alpha_S \cap \alpha_W\} \cup \{\pi' \in V'^* \cdot (V'_{\exists} \cdot (V'_{\forall})^*)^\omega \mid \text{seq}_A(\pi') \in \alpha_S\}$$

► **Example 2.** For brevity, we refrain from showing the complete witness game associated to the obliging game from Example 1 and instead consider just two witnesses for the node v_1 , namely $(v_1 v_2 v_4 v_5)^\omega$ and $(v_1 v_2 v_3 v_4 v_1 v_2 v_4 v_5)^\omega$. The former satisfies the Streett objective α_S as it visits the colors a and b . However, it does not satisfy the generalized Büchi objective α_W as color c is not visited infinitely often. The latter witness, contrarily, visits all colors infinitely often and hence satisfies $\alpha_S \wedge \alpha_W$. In the witness game, player \exists can move from v_1 to these two witnesses (and to many more). Player \forall in turn can win the first witness by exploring it indefinitely, thereby showing that it does not satisfy α_W ; doing the same for the second witness, player \forall loses. In both certificates, we have exit moves from every position such that the node at the position is owned by player \forall , that is, moves from positions with node v_2 to $E(v_2)$, and similarly for v_3 and v_5 .

► **Lemma 3.** *Player \exists is graciously winning in G at v iff player \exists is winning in $W(G)$ at v .*

► **Corollary 4.** *Obliging games are determined.*

Proof. This follows immediately from Lemma 3 and the determinacy of games with Borel winning conditions [21]. Notice that Martin's determinacy result holds also for games with continuous vertex-spaces and continuous branching degrees, as in the witness game. ◀

4 Reducing ω -Regular Obliging Games to Finite Games

From this point on, we restrict our attention to obliging games in which both objectives are Emerson-Lei objectives; we note that every ω -regular objective can be transformed to an Emerson-Lei objective. It turns out that due to the ω -regularity of Emerson-Lei objectives, obliging plays in such games have finite witnesses that take on the form of lassos that are built over the game arena at hand. Formally, we show that for every such game we can create a game that is smaller than the witness game by restricting attention to witnesses of this specific form that satisfy the acceptance conditions. We call such witnesses *certificates*, which we define next.

4.1 Certificates from Witnesses

We fix an Emerson-Lei obliging game $G = (A, \alpha_S, \alpha_W)$ with arena $A = (V, V_\exists, E)$, objectives $\alpha_S = (\gamma_S, \varphi_S)$ and $\alpha_W = (\gamma_W, \varphi_W)$, and put $n := |V|$, $d := |S|$ and $k := |W|$.

► **Definition 5 (Certificate).** *Given a node $v \in V$, a certificate for v (in G) is a witness for v that is of the form $c = wu^\omega$; if c satisfies $\varphi_S \wedge \varphi_W$, then we say that c is a valid certificate.*

We equally represent $c = wu^\omega$ by the pair $c = (w, u)$. Let $w = w_0w_1 \dots w_m$ and $u = u_0u_1 \dots u_r$. We refer to w as the *stem* and to u as the *loop* of c , and to m and r as the length of the stem and the loop, respectively. When the partition of c is not important we sometimes just write $c = v_0 \dots v_{m+r+1}$. Clearly, as satisfaction of Emerson-Lei objectives depends only on the infinite suffixes of a play, it follows that in a valid certificate, u^ω also satisfies φ_S and φ_W .

Given a coloring function γ_C over some set C of colors, the C -*fingerprint* of a finite play $\pi = v_0v_1 \dots v_j \in A^*$ is the set $\bigcup_{0 \leq i < j} \gamma_C(v_i, v_{i+1})$ of colors visited by π , according to γ_C .

Next we show that given a witness in G that satisfies $\alpha_S \wedge \alpha_W$, we can always construct a valid certificate of size at most $\text{certLen} := n \cdot d + (d + k + 1) \cdot (n + 1) \in \mathcal{O}(n \cdot (\max(d, k)))$.

► **Lemma 6 (Certificate existence).** *Let $v \in V$ and let $\pi = \pi_0\pi_1 \dots$ be a witness for $\pi_0 = v$ that satisfies $\varphi_S \wedge \varphi_W$. Then there is a valid certificate $c = (w, u)$ for v with stem length at most $n \cdot d$ and loop length at most $(d + k + 1) \cdot (n + 1)$, such that for all positions i in c there is a position j in π such that $v_i = \pi_j$ and the S -fingerprints of $v_0 \dots v_i$ and $\pi_0 \dots \pi_j$ coincide.*

We let $\text{Cert}(v)$ and Cert denote the sets of all valid certificates for some $v \in V$ in G and all valid certificates for all $v \in V$ in G , respectively, subject to the size bounds obtained in Lemma 6. Then we have $|\text{Cert}(v)| \leq |\text{Cert}| \leq n^{\text{certLen}} \in 2^{\mathcal{O}(n \cdot (\max(d, k)) \cdot \log n)}$.

4.2 Certificate Games and Smaller Winning Strategies

Next we adapt the witness games from Section 3 to use finite certificates in place of witnesses, which leads to *certificate games* that have finite game arenas. In a certificate game, player \exists has to pick a single valid certificate at each node $v \in V$, thereby committing to a long-term future behavior that satisfies $\alpha_S \wedge \alpha_W$. Player \forall in turn can either accept the certificate (and thereby lose the play), or pick some position i in the certificate and challenge whether player \exists still has a strategy to graciously win when player \forall exits the certificate at position i . All plays then either end with player \forall eventually losing by accepting some certificate, or player \forall infinitely often exits certificates, in which case the winner of the play is determined using just the strong objective α_S .

Formally, the certificate game associated to G is a (non-obliging) Emerson-Lei game $C(G) = (B, \beta)$ with arena $B = (N, N_\exists, R)$. The game is played over the sets $N_\exists = V$ and $N_\forall = \text{Cert} \cup \text{Cert} \times [\text{certLen}] \times V$ of nodes. The moves in $C(G)$ are defined by

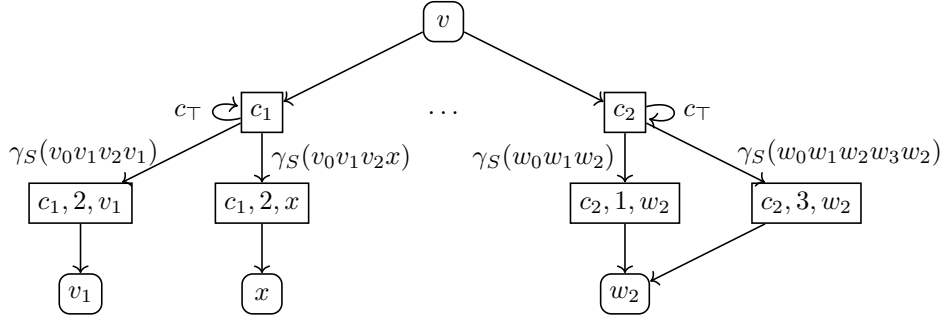
$$R(v) = \text{Cert}(v) \quad R(c) = \{(c, i, v) \in N_\forall \mid v_i \in V_\forall, v \in E(v_i)\} \cup \{c\} \quad R(c, i, v) = \{v\}$$

for $v \in V$, $c = v_0v_1 \dots v_m \in \text{Cert}$ and $i \in [\text{certLen}]$. Thus player \exists has to provide a valid certificate for v whenever a play reaches a node $v \in V$. Player \forall in turn can challenge the way certificates are combined by exiting them from universal nodes in their stem or loop, and continuing the game at the exit node; if a certificate c does not contain a universal node, then player \forall has to take the loop at c , intuitively accepting the certificate c . Intermediate nodes (c, i, v) are used to make explicit the S -fingerprint of the path through a certificate c

that is taken before exiting it by moving from v_i to v ; this is necessary since a certificate may contain universal nodes that allow moving from different positions in the certificate to a single exit node v , potentially giving player \forall a choice on the path that is taken (and on the S -fingerprint that is accumulated) through the certificate before exiting to v .

As our notion of games does not support deadlocks, we annotate trivial loops with an additional color c_\top to encode the situation that player \forall accepts a certificate and thereby loses. The coloring function $\gamma' : R \rightarrow 2^{S \cup \{c_\top\}}$ also keeps track of the S -fingerprints when passing through certificates and is defined by $\gamma'(c, (c, i, v)) = \gamma_S(v_0 \dots v_i v)$, $\gamma'(v, c) = \emptyset$ and $\gamma'((c, i, v), v) = \emptyset$ for the non-looping moves and by $\gamma'(c, c) = c_\top$ for looping moves at certificates $c \in \text{Cert}$. We then define $\varphi' = \text{Inf}(c_\top) \vee \varphi_S$, intuitively expressing that player \exists can win either according to φ_S or by forcing player \forall to get stuck in a trivial loop. Then we define β to be the objective induced by φ' and γ' .

► **Example 7.** Below we depict the construction for a single node $v \in V$, showing, as examples, just two valid certificates $c_1 = v_0 v_1 v_2 \in \text{Cert}(v)$ and $c_2 = w_0 w_1 w_2 w_3 \in \text{Cert}(v)$ (in particular, we assume $v_0 = w_0 = v$ and that the certificates c_1 and c_2 satisfy both φ_S and φ_W). We further assume that $v_2 \in V_\forall$, $E(v_2) = \{v_1, x\}$, and that $w_1 = w_3 \in V_\forall$ and $E(w_1) = E(w_3) = \{w_2\}$; all other nodes in c_1 and c_2 are assumed to be contained in V_\exists .



At node v , player \exists has to provide some valid certificate for v ; assuming that player \exists picks the certificate $c_1 = v_0 v_1 v_2$, player \forall in turn can exit the certificate at position 2 (as $v_2 \in V_\forall$) by moving to either $(c_1, 2, v_1)$ or $(c_1, 2, x)$ (as $E(v_2) = \{v_1, x\}$) thereby triggering S -fingerprint $\gamma_S(v_0 v_1 v_2 v_1)$ or $\gamma_S(v_0 v_1 v_2 x)$; player \forall intuitively cooperates during the play that leads from v_0 to v_2 , but may stop cooperating at node v_2 by moving to either v_1 or x . Similarly, player \forall can exit the certificate c_2 at positions 1 or 3, both leading to the node w_2 , but with different S -fingerprints, that is, with different sets of visited colors.

As the above example shows, certificate games have a three-stepped structure. Plays progress from nodes v of the original game on to certificate nodes c , and then onwards to nodes (c, i, v') that encode the part of c that is visited before exiting c , and then proceed back to some node v' of the original game. We refer to these three-step subgames a *gadgets*; each gadget has a starting node v and a (possibly empty) set of exiting nodes. We point out that, crucially, gadgets do not contain (non-trivial) loops.

We state the correctness of the reduction to certificate games as follows.

► **Theorem 8.** *Let G be an Emerson-Lei obliging game and let v be a node in G , and recall that $W(G)$ and $C(G)$ respectively are the witness game and the certificate game associated with G . The following are equivalent:*

1. *player \exists graciously wins v in G ;*
2. *player \exists wins v in $W(G)$;*
3. *player \exists wins v in $C(G)$.*

Proof. The equivalence of the first two items is stated by Lemma 3 and the proof of the implication from item two to item three is technical but straight-forward. Therefore we show just the implication from item three to item one, which also shows how to construct a graciously winning strategy in G from a winning strategy in $C(G)$. For this proof, we use *strategies with memory*, introduced next. A strategy for player $i \in \{\exists, \forall\}$ with memory M is a tuple $\sigma = (M, m_0, \text{update} : M \times E \rightarrow M, \text{move} : V_i \times M \rightarrow V)$, where M is some set of *memory values*, $m_0 \in M$ is the initial memory value, the *update function* update assigns the outcome $\text{update}(m, e) \in M$ of updating the memory value $m \in M$ according to the effects of taking the move $e \in E$, and the moving function move prescribes a single move $(v, \text{move}(v, m)) \in E$ to every game node $v \in V_i$ that is owned by player i , depending on the memory value m . We extend update to finite plays π by putting $\text{update}(m, \pi) = m$ in the base case that π consists of a single node, and by putting $\text{update}(m, \pi) = \text{update}(\text{update}(m, \tau), (v, w))$ if π is of the shape τvw , that is, contains at least two nodes.

Let $\sigma = (M, m_0, \text{update}, \text{move})$ be a winning strategy with memory M for player \exists in the game $C(G)$. We construct a strategy $\tau = (M', (m_0, v, 1), \text{update}', \text{move}')$ with memory $M' = V \times M \times \{1, \dots, 2|S| + |W|\}$ for player \exists in the game G such that τ graciously wins v . To this end, we note that σ provides, for each node $w \in V$ and memory value $m \in M$, a certificate $c(w, m) := \text{move}(w_{\exists}, m) \in \text{Cert}(w)$ for w in G .

The strategy τ uses memory values (w, m, i) , where w and m identify a current certificate $c(w, m)$ and i is a counter used for the construction of this certificate. We define τ such that player \exists starts by building the certificate $c(v, m_0)$ for v and m_0 . This process continues as long as player \forall obliges, that is, does not move outside of this certificate. Assuming that player \forall obliges, the memory required to construct the certificate is bounded by $2|S| + |W|$, walking, in the prescribed order, through the certificate. In the case that player \forall eventually stops obliging and takes a move to some node w that is not the next node on the path prescribed by the certificate, the memory for the strong objective is updated according to the play from v to w , resulting in a new memory value m , and the memory for certificate construction is reset. Then the certificate construction starts again, this time for the certificate $c(w, m)$ prescribed by σ for the new starting values w and m .

In more detail, every node from V is visited at most $2|S| + |W|$ times within a certificate $c(v, m) = (v_0 v_1 \dots v_n)$ before the end of the loop v_n is reached. To each position i within $c(v, m)$, we associate the number $\#i$ such that v_i is the $\#i$ -th occurrence of the node v_i in $c(v, m)$; we note that for all $1 \leq i \leq n$, we have $1 \leq \#i \leq 2|S| + |W|$. Conversely, given a node w and a number j between 1 and $2|S| + |W|$, we let $\text{pos}(w, j)$ denote the position of the j -th occurrence of w in $c(v, m)$. In the case w occurs less than j times in $c(v, m)$, we leave $\text{pos}(w, j)$ undefined; below we make sure that this value is always defined when it is used.

Let j be the starting position of the loop of $c(v, m)$. Given a position i in $c(v, m)$, we abuse notation to let $i + 1$ denote just $i + 1$ if $i < n$, and to denote j if $i = n$; in this way, we encode taking single steps within a certificate, wrapping back to the start of the certificate loop, once the end of the loop is reached.

We define the strategy τ to always move to the next node in the current certificate, using the memory value i together with the current node w to find the current position in the certificate $c(v, m)$, that is, we put

$$\text{move}'(w, (v, m, i)) = v_{\text{pos}(w, i)+1}$$

for $w \in V_{\exists}$ and $(v, m, i) \in M'$. The memory update in τ incorporates the memory update function from σ , but additionally also keeps track of the memory for certificate construction. For moves (w, w') that stay within the current certificate (that is, $w' = v_{\text{pos}(w, i)+1}$), this

memory is updated according to proceeding one step within the certificate; for moves that leave the current certificate, the memory for certificate construction is reset while the memory for σ is updated according to the path taken through the certificate before exiting it. Formally, we put

$$\text{update}'((v, m, i), (w, w')) = \begin{cases} (v, m, \#(\text{pos}(w, i) + 1)) & w' = v_{\text{pos}(w, i)+1} \\ (w', \text{update}(m, (v_0 v_1 \dots w w')), 1) & w' \neq v_{\text{pos}(w, i)+1} \end{cases}$$

for $(v, m, i) \in M'$ such that $c(v, m) = v_0 v_1 \dots v_n$ and $(w, w') \in E$; notice that in plays that adhere to τ , the latter case can, by definition of move' , only happen for $w \in V_\forall$.

To see that player \exists graciously wins v using the constructed strategy τ , let π be a play that adheres to τ . Then π either eventually stays within one certificate $c(w, m)$ forever, or π induces an infinite play ρ of $C(G)$ that adheres to σ . In the latter case, π changes the certificate infinitely often, and the S -fingerprints of π and ρ coincide by construction, showing that π satisfies φ_S ; if π eventually stays within one certificate $c(w, m)$ forever, we note that the loop of $c(w, m)$ satisfies φ_S so that π satisfies φ_S as well. Thus every play that adheres to τ satisfies the strong objective φ_S . Next, let π_f be a finite prefix of some play that adheres to τ , and let π_f end in some node w . We have to show that there is an infinite play π that adheres to τ , extends π_f and satisfies φ_W . Let m be the value of the memory for the strong objective at the end of π_f . We consider the play of G in which player \forall obliges from the end of π_f on, forever. By construction, this play is π_f extended with the certificate $c(w, m)$ which adheres to τ and satisfies φ_W . \blacktriangleleft

The strategy construction given in the proof of Theorem 8 yields:

► **Corollary 9.** *Given an obliging game with Emerson-Lei objectives α_S and α_W and n nodes that contains a node v at which \exists is graciously winning, there is a graciously winning strategy for v that uses memory at most $n \cdot (2|S| + |W|) \cdot m$, where m is the amount of memory required by winning strategies for player \exists in standard games with objective α_S .*

► **Remark 10 (Canonical certificates).** With the proposed strategy extraction, certificate strategies memorize the starting point of the certificate that player \exists currently attempts to construct, leading to an additional linear factor n in strategy size. We conjecture that winning strategies in certificate games can be transformed to make them *reuse* certificates (so that the choice of the certificate does not depend on the starting point). Strategies with such canonical certificate choices would allow for removing the additional factor n in Corollary 9.

In particular, our result shows the existence of graciously winning strategies with quadratic sized strategies for all obliging games that have a half-positional strong objective (i.e. one for which standard games have positional winning strategies for player \exists); this covers, e.g., obliging games in which α_S is a parity or Rabin objective.

In the table below we compare the solution via certificate games to a previous solution method [6, 20] reduces α_S / α_W obliging games to standard games with an objective of type $\alpha_S \wedge$ Büchi. In this approach, the weak objective is encoded by a non-deterministic Büchi automaton and graciously winning strategies obtained in this way incorporate the state space of the automaton. The encoding of α_W by a Büchi automaton leads to linear blow-up if α_W is a Rabin objective, but is exponential if α_W is a Streett or general Emerson-Lei objective.

In contrast to this, the certificate games that we propose here always have (essentially) just α_S as objective, and the dependence of strategy size on the weak objective α_W is in all cases linear in $|W|$. In comparison to [6], our approach hence leads to significantly smaller

■ **Table 1** Comparison of upper bounds on strategy sizes for various types of obliging games.

type of α_S	type of α_W	objective red. [6]	strategy size [6]	strategy size
parity(d)	Rabin(k)	parity(d) \wedge Büchi	$d(4k + 2)$	$(2d + 2k)n$
	Streett(k)		$2^{k+1}dk$	$(2d + 2k)n$
	EL(k)		$2^{k+1}dk$	$(2d + k)n$
Rabin(d)	Rabin(k)	Rabin(d) \wedge Büchi	$d(4k + 2)$	$(4d + 2k)n$
	Streett(k)		$2^{k+1}dk$	$(4d + 2k)n$
	EL(k)		$2^{k+1}dk$	$(4d + k)n$
Streett(d)	Rabin(k)	Streett($d + 1$)	$(d + 1)!(4k + 2)$	$(4d + 2k)d!n$
	Streett(k)		$(d + 1)!2^{k+2}k$	$(4d + 2k)d!n$
	EL(k)		$(d + 1)!2^{k+2}k$	$(4d + k)d!n$
EL(d)	EL(k)	EL($d + 1$)	$(d + 1)!2^{k+2}k$	$(2d + k)d!n$

strategies for obliging games in which α_W is at least a Streett objective. In the cases where α_W is a Rabin objective, strategies obtained from certificate games are slightly larger than in [6]; we conjecture that this can be improved by sharing certificates (cf. Remark 10).

For instance, for obliging games with $\alpha_S = \text{Rabin}(d)$ and $\alpha_W = \text{Streett}(k)$, the approach from [6] uses nondeterministic Büchi automata with $2^k k$ states to encode the weak (Streett) objective. The reduction leads to a game with objective $\text{Rabin}(d) \wedge \text{Büchi}$; winning strategies in such games require $2d$ additional memory values for each automaton state (cf. [7]), resulting in an overall memory requirement of $2^{k+1}dk$. In contrast, the reduced certificate game in this case is a $\text{Rabin}(d)$ game, and the extracted gracious strategies require memory only to identify the current certificate and a position in it; the overall memory requirement for strategies obtained through our approach thus is $(4d + 2k)n$.

5 Solving Certificate Games Efficiently

In the previous section we have shown how obliging games with Emerson-Lei objectives α_S and α_W can be reduced to certificate games. This reduction not only yields games with a simpler objective (essentially just α_S) than in the previously known alternative reduction from [6], but it also shows that the memory required for graciously winning strategies in obliging games always depends only linearly on $|W|$. However, the proposed reduction to certificate games makes explicit all possible certificates that exist in the original obliging game and therefore incurs exponential blowup. In more detail, given an Emerson-Lei obliging game G with n nodes and sets S and W of colors, the certificate game $C(G)$ from the previous section is of size $2^{\mathcal{O}(n \cdot \max(|S|, |W|) \cdot \log n)}$ and uses $|S|$ many colors; solving it naively does not improve upon previously known solution algorithms for obliging games.

In this section, we show that the gadget constructions in certificate games essentially encode non-emptiness checking for non-deterministic ω -automata; intuitively, a certificate for a game node is a witness for the non-emptiness of an Emerson-Lei automaton with acceptance condition $\alpha_S \wedge \alpha_W$ that lives over (parts of) the original game arena. We show that it suffices to check for the existence of a single suitable certificate, rather than exploring all possible certificates that occur as explicit nodes in the reduced game.

As pointed out above, the gadget parts in a certificate game do not have non-trivial cycles (that is, we can regard them as directed acyclic graphs, DAGs). Consequently, it is possible to simplify the solution process of these (potentially) exponential-sized parts of the game by instead using non-emptiness checking for suitable ω -automata. If for instance both α_S and α_W are Streett conditions (so that $\alpha_S \wedge \alpha_W$ again is a Streett condition), then the

solution of every gadget part in the game can be reduced to non-emptiness checking of Streett automata. As non-emptiness checking for Streett automata can be done in polynomial time, this trick in effect removes the exponential runtime factor that originates from the large number of certificates when solving such games (and in general, whenever non-emptiness of $\alpha_S \wedge \alpha_W$ -automata can be checked efficiently).

Our program is as follows: We first show how Emerson-Lei certificate games can be reduced to parity games using a tailored later-appearance-record (LAR) construction, incurring blow-up $|S|!$ in game size, but retaining the DAG structure of gadget subgames. Importantly, this is required to retain the dependence of non-DAG nodes on a small number of (post DAG) successors. Then we show the relation between attractor computation in gadget subgames within the obtained parity games and the non-emptiness problem for specific Emerson-Lei automata. Finally, we apply a fixpoint acceleration method from [14] to show that during the solution of parity games, the solution of DAG substructures can be replaced with a procedure that decides attraction to (subsets of) the exit nodes of the DAG. Overall, we therefore show that the winning regions of certificate games can be computed as nested fixpoints of a function that checks certificate existence by nonemptiness checking suitable Emerson-Lei automata.

5.1 Lazy parity transform

We intend to transform $C(G)$ to an equivalent parity game, using a lazy variant of the later-appearance-record (LAR) construction (cf. [12, 17]). To this end, we fix a set C of colors and introduce notation for permutations over C . We let $\Pi(C)$ denote the set of permutations over C , and for a permutation $\pi \in \Pi(C)$ and a position $1 \leq i \leq |C|$, we let $\pi(i) \in C$ denote the element at the i -th position of π . For $D \subseteq C$ and $\pi \in \Pi(C)$, we let $\pi@D$ denote the permutation that is obtained from π by moving *the* element of D that occurs at the right-most position in π to the front of π ; for instance, for $C = \{a, b, c, d\}$ and $\pi = (a, d, c, b) \in \Pi(C)$, we have $\pi@\{a, d\} = \pi@\{d\} = (d, a, c, b)$ and $(d, a, c, b)@\{a, d\} = \pi$. Crucially, restricting the reordering to single colors, rather than sets of colors, ensures that for each $\pi \in \Pi(C)$ and all $D \subseteq C$, there are only $|C|$ many π' such that $\pi@D = \pi'$. Given a permutation $\pi \in \Pi(C)$ and an index $1 \leq i \leq |C|$, we furthermore let $\pi[i]$ denote the set of colors that occur in one of the first i positions in π .

Next we show how permutations over C can be used to transform Emerson-Lei games with set C of colors to parity games; the reduction annotates nodes from the original game with permutations that serve as a memory, encoding the order in which colors have recently been visited. The transformation is lazy as it just moves the most significant color that is visited by a set of colors C rather than the entire set.

► **Definition 11.** *Let $G = (A, \alpha_C)$ be an Emerson-Lei game with arena $A = (V, V_\exists, E)$, set C of colors and objective α_C induced by γ_C and φ_C . We define the parity game*

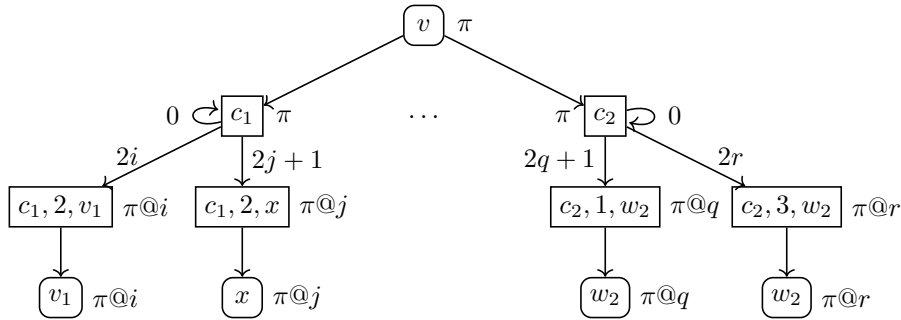
$$P(G) = (V \times \Pi(C), V_\exists \times \Pi(C), E', \Omega : E' \rightarrow \{1, \dots, 2|C| + 1\})$$

by putting $E'(v, \pi) = \{(w, \pi@\gamma_C(v, w)) \mid (v, w) \in E\}$ for $(v, \pi) \in V \times \Pi(C)$, as well as $\Omega((v, \pi), (w, \pi')) = 2p$ if $\pi[p] \models \varphi_C$ and $\Omega((v, \pi), (w, \pi')) = 2p + 1$ if $\pi[p] \not\models \varphi_C$, for $((v, \pi), (w, \pi')) \in E'$. In the definition of $\Omega((v, \pi), (w, \pi'))$, we write p to denote the right-most position in π that contains some color from $\gamma_C(v, w)$. For a finite play $\tau = (v_0, \pi_0)(v_1, \pi_1) \dots (v_n, \pi_n)$ of $P(G)$, we let $p(\tau)$ denote the right-most position in π_0 such that $\pi_0(p(\tau)) \in \gamma_C(v_0 v_1 \dots v_n)$. The game $P(G)$ has $|V| \cdot |C|!$ nodes and $2|C| + 1$ priorities.

► **Lemma 12.** *For all $v \in V$ and $\pi \in \Pi(C)$, player \exists wins from v in G iff player \exists wins from (v, π) in $P(G)$.*

Now we consider the parity game $P(C(G))$ that is obtained by applying the above LAR construction to the certificate game $C(G)$ from Section 4. It is a parity game with $2^{\mathcal{O}(n \cdot \max(|S|, |W|) \cdot \log n)} \cdot |S|!$ many nodes and $2|S| + 1$ priorities. We recall that all nodes in $C(G)$ that are of the shape c or (c, i, v) such that $c, (c, i, v) \in N_{\forall}$ are inner nodes of gadget subgames that consist of three layers. Each such subgame has exactly one entry node and at most n exit nodes, all contained in $N_{\exists} = V$. The LAR construction preserves this general structure as it simply annotates game nodes with permutations of colors. Specifically, $P(G)$ has $|S|! \cdot n$ entry and exit nodes for all such subgames together. Furthermore, for all entry nodes (v, π) of a subgame in $P(G)$, we have that every exit node (that can be reached from (v, π) with exactly three moves, not accounting for trivial loops) is of the shape $(w, \pi @ i)$ where $w \in V$ and $0 \leq i \leq |C|$. While an entry node for an individual subgame in $C(G)$ has at most n exit nodes, each entry node for a subgame in $P(C(G))$ has at most $n(|S| + 1)$ exit nodes. Using the classical LAR would result in a potentially exponential number of exit nodes. Indeed, for every possible subset $C' \subseteq C$ there could be a different exit node.

► **Example 13.** Below, we depict (part of) the parity game that is obtained from the certificate game from Example 7 by using the proposed LAR construction.



Here $i = p(v_0 v_1 v_2 v_1)$, $j = p(v_0 v_1 v_2 x)$, $q = p(w_0 w_1 w_2)$, $r = p(w_0 w_1 w_2 w_3 w_2)$ and we assume that the sets $\pi[i]$ and $\pi[r]$ satisfy φ_S and that the sets $\pi[j]$ and $\pi[q]$ do not satisfy φ_S , leading to the respective even and odd priorities. In particular, we have $2q + 1 \neq 2r$ so that $(w_2, \pi @ q)$ and $(w_2, \pi @ r)$ are two distinct exit nodes from the certificate c_2 . Intuitively they correspond to two different paths through c_2 with different S -fingerprints; while in the Emerson-Lei game $C(G)$, the different fingerprints are dealt with by signalling different sets of colors, in the parity game $P(C(G))$, the two paths have different effects on the later-appearance memory π and thus lead to different outcome nodes. The self-loops at nodes c_1 and c_2 correspond to player \forall giving up, so we assign priority 0 to these moves.

5.2 Solving parity games using DAG attraction

A standard way of solving parity games is by computing a nested fixpoint of a function that encodes one-step attraction in the game; the domain of this fixpoint computation then is the set of all game nodes. For parity games that contain cycle-free parts (DAGs), this process can be improved by instead computing a nested fixpoint of a function that encodes multi-step attraction along the DAG parts of the game. The domain of the latter fixpoint computation then does not contain the internal nodes of the DAG parts, which leads to accelerated fixpoint stabilization. We formalize this idea as follows.

► **Definition 14** (DAGs in games). Let $G = (A, \Omega)$ be a parity game with $A = (V, V_{\exists}, E)$ and $k+1$ priorities $0, \dots, k$. We refer to a set $W \subseteq V$ of nodes as a DAG (directed acyclic graph) if it does not contain an E -cycle; then there is no play of G that eventually stays within W forever. A DAG need not be connected, that is, it may consist of several cycle-free subgames of G . Given a DAG $W \subseteq V$, we write $V' = V \setminus W$ and refer to the set V' as real nodes (with respect to W). A DAG is positional if for each existential node $w \in W \cap V_{\exists}$ in it, there is exactly one real node $v \in V'$ from which w is reachable without visiting other real nodes.

► **Definition 15** (DAG attraction). Given a DAG W and $k+1$ sets $\bar{V} = (V_0, \dots, V_k)$ of real nodes and a real node $v \in V'$, we say that player \exists can attract to \bar{V} from v within W if they have a strategy σ such that for all plays π that start at v and adhere to σ , the first real node v' in π such that $v' \neq v$ is contained in V_p , where p is the maximal priority that is visited by the part of π that leads from v to v' . Given a dag W , we define the dag attractor function $\text{DAttr}_{\exists}^W : \mathcal{P}(V')^k \rightarrow \mathcal{P}(V')$ by $\text{DAttr}_{\exists}^W(\bar{V}) = \{v \in V' \mid \text{player } \exists \text{ can attract to } \bar{V} \text{ from } v \text{ within } W\}$ for $\bar{V} = (V_0, \dots, V_k) \in \mathcal{P}(V')^k$. We denote by $t_{\text{Attr}_{\exists}^W}$ the time required to compute, for every input $\bar{V} \in \mathcal{P}(V')^k$, the dag attractor of \bar{V} through W .

► **Remark 16.** The sets V_i in the above definition correspond to valuations of fixpoint variables in the nested fixpoint computation that is used by the fixpoint acceleration method in Lemma 17 below to solve games via DAG attraction. These sets monotonically increase or decrease during the solution process and at each point of the solution process, a set V_i intuitively holds game nodes for which it currently is assumed that player \exists wins if they can force the game to reach a node from V_i via a partial play in which the maximal priority is i .

During the computation of the DAG attractor to a tuple $\bar{V} = (V_0, \dots, V_k)$, we therefore intuitively consider the argument nodes \bar{V} to be *safe* in the sense that in order to win from a node v , it suffices that existential player has a strategy that ensures that every partial play through the DAG exits it to a node from V_i , where i the maximal priority visited by that play along the DAG. Thus if player \exists can win from all nodes in \bar{V} , then they can win from all nodes in $\text{DAttr}_{\exists}^W(\bar{V})$.

In our case, from a node v , a strategy σ corresponds to choosing one certificate. Then, player \forall can attract to all successors of \forall -nodes on the certificate. The path through the certificate to this \forall -node and to its successor outside the certificate shows a certain priority j that implies the successor must be in the set V_j .

► **Lemma 17** ([13]). Let G be a parity game with priorities 0 to k and set V of nodes, let W be a positional DAG in G , and let $n = |V|$ and $m = n - |W|$. Then G can be solved with $\mathcal{O}(m^{\log k+1})$ computations of a DAG attractor; if $k+1 < \log m$, then G can be solved with a number of DAG attractor computations that is polynomial in m (specifically: in $\mathcal{O}(m^5)$).

We always have $t_{\text{Attr}_{\exists}^W} \leq |E|$, using a least fixpoint computation to check for alternating reachability, thereby possibly exploring all DAG edges of the game. However, in the case that $m < \log n$ and $t_{\text{Attr}_{\exists}^W} \in \mathcal{O}(\log n)$ (that is, when most of the game nodes are part of a DAG, and DAG attractability can be decided without exploring most of the DAG nodes), Lemma 17 enables exponentially faster game solving.

5.3 Checking certificate existence efficiently

The parity game $P(C(G))$ that is obtained by applying the LAR construction from Subsection 5.1 to the certificate game $C(G)$ has $2^{\mathcal{O}(n \cdot \max(d,k) \cdot \log n)} \cdot |S|!$ nodes, but only $|S|!n$ of these are real nodes: all certificate nodes are internal nodes in a gadget that has a DAG structure. In this section, we show that DAG attractors in $P(C(G))$ can be computed efficiently relying on non-emptiness checking of suitable Emerson-Lei automata.

In $P(C(G))$, the DAG attractor to a tuple $\bar{V} = (V_1, \dots, V_{2|C|+1})$ consists of the nodes (v, π) such that there is a valid certificate starting at v such that all exits points of the certificate are safe in the sense of Remark 16, that is, exiting the certificate with fingerprint i is only possible to nodes $w \in V_i$; we refer to such certificates as *valid and safe*.

Next we show how the existence of valid and safe certificates can efficiently be checked by using non-emptiness checking for Emerson-Lei automata to find valid and safe certificate loops that reside over single sets V_i (as the fingerprint does not increase within certificate loops, by the construction of certificates as in the proof of Lemma 6), and then using a reachability analysis that keeps track of fingerprints to compute safe stems leading (with fingerprint i) to some loop over V_i . In more detail, we check for the existence of valid and safe certificates as follows, fixing a permutation π and a tuple $(V_1, \dots, V_{2|C|+1})$, where each V_i is a set of real nodes in $P(C(G))$.

- *Make the fingerprints explicit.* Define the graph $M = (V \times [2|C| + 1], R)$ as follows. Vertices of the graph are pairs (v, i) consisting of a game node $v \in V$ and a priority $i \in [2|C| + 1]$, intuitively encoding the largest priority that has been visited since a DAG has been entered; edges in this graph correspond to game moves but also update the priority value according to visited priorities, that is, R contains exactly the edges $((v, i), (w, j))$ such that $w \in E(v)$ and $j = \max(i, \Omega_\pi(v, w))$, where $\Omega_\pi(v, w)$ denotes priority associated to seeing the set $\gamma_C(v, w)$ of colors on memory π (cf. Definition 11).
- *Remove unsafe vertices.* A vertex (u, i) such that $(u, \pi @ i)$ is contained neither in V_{2i} nor in V_{2i+1} is *unsafe*. Remove from M all vertices (v, i) such that $v \in V_\forall$ and there is $w \in E(v)$ such that (w, i) is unsafe; these are vertices from where player \forall can access an unsafe exit point of the DAG.
- *Find safe and valid certificate loops:* Define, for all $i \in [2|C| + 1]$, a nondeterministic Emerson-Lei automaton $A_i = (Q_i, \delta, \alpha_S \wedge \alpha_W)$ (with singleton alphabet $\{*\}$) by putting $Q_i = \{(v, i) \mid (v, i) \text{ still exists in } M\}$ and $\delta((v, i), *) = \{(w, i) \mid w \in E(v) \text{ and } \Omega_\pi(v, w) \leq i\}$ for $(v, i) \in Q_i$. Compute the non-emptiness region of A_i and call it N_i .
- *Find safe stems:* Remove from M all vertices $(v, 0)$ for which there is no j such that some vertex from N_j is reachable (in M) from $(v, 0)$.

For all vertices $(v, 0)$ that are contained in M after this procedure terminates, there is a safe stem w leading (with maximal priority j) to some safe and valid loop u over N_j ; (w, u) is a valid certificate for v . We state the correctness of the described procedure as follows.

► **Lemma 18.** *Given subsets $\bar{V} = V_0, \dots, V_{2k}$ of the real nodes in $P(C(G))$ and a real node (v, π) in $P(C(G))$, we have that player \exists can attract to \bar{V} from (v, π) within $\text{Cert} \times \Pi(S)$ if and only if the set M contains the pair $(v, 0)$ after execution the above procedure (for parameters \bar{V} and (v, π)).*

► **Corollary 19.** *DAG attractors in $P(C(G))$ can be computed in time $\mathcal{O}(d!dt)$ where t denotes the time it takes to check $\alpha_S \wedge \alpha_W$ automata of size n for non-emptiness.*

Proof. In order to compute a DAG attractor in $P(C(G))$, it suffices to execute the above procedure once for each $\pi \in \Pi(S)$, that is, $d!$ many times; a single execution of the procedure can be implemented in time $\mathcal{O}(dt)$. ◀

5.4 Faster solution of obliging games

We are now ready to state the main result of this section.

► **Theorem 20.** *Certificate games for objectives α_S and α_W with n nodes and $d := |S|$ colors for the strong objective can be solved in time $\mathcal{O}((d!n)^5 d!dt)$, where t denotes the time it takes to check Emerson-Lei automata of size n and with acceptance condition $\alpha_S \wedge \alpha_W$ for non-emptiness. If α_S is a parity objective, then the runtime bound is $\mathcal{O}(n^{\log(2d+1)} dt)$.*

Proof. By Lemma 12, it suffices to solve the paritized version $P(C(G))$ of $C(G)$. By Lemma 18, computing DAG attractors in $P(C(G))$ can be done in time $\mathcal{O}(d!dt)$. As we have $d < \log(d! \cdot n)$, $P(C(G))$ can be solved with $(d!n)^5$ computations of a DAG attractor by Lemma 17. If α_S is a parity objective, then the LAR construction is not necessary as $C(G)$ already is a parity game; DAG attractors then can be computed in time $\mathcal{O}(dt)$ and $C(G)$ can be solved with $\mathcal{O}(n^{\log(2d+1)})$ computations of a DAG attractor. ◀

We collect results on the complexity of non-emptiness checking of Emerson-Lei automata with acceptance condition $\alpha_S \wedge \alpha_W$ for specific α_S and α_W . It is known (cf. Table 2. in [1]) that while the problem is in P for most frequently used objectives (subsuming automata with generalized Büchi, Rabin or Streett conditions, with linear or quadratic dependence on the number of colors), it is NP-complete for Emerson-Lei conditions. For combinations of such objectives, the problem remains in P unless one of the objectives is of type Emerson-Lei:

► **Lemma 21.** *The $\text{Rabin}(d) \wedge \text{Streett}(k)$ non-emptiness problem is in P (more precisely: in $\mathcal{O}(mdk^2)$ for automata with m edges).*

Proof. Let A be an Emerson-Lei automaton with n nodes, m edges and acceptance condition $\text{Streett}(d) \wedge \text{Rabin}(k)$. We check A for non-emptiness as follows. Let the Rabin(k) condition be encoded by k Rabin pairs (E_i, F_i) . For each $1 \leq i \leq k$, check the same automaton but with acceptance condition $\text{Streett}(d) \wedge \text{Inf}(F_i) \wedge \text{Fin}(E_i)$ for emptiness; call this automaton A_i . The acceptance condition of A_i can be treated as a Streett($d+2$) condition where the two additional Streett pairs are (\top, F_i) and (E_i, \perp) . As a state in A is non-empty if and only if there is some i such the state is non-empty in A_i , it suffices to check the k many Streett($d+2$) automata for emptiness. The claim follows from the bound on emptiness checking for Streett automata given in [1]. ◀

Using the equivalence of certificate games and obliging games (Theorem 8), Theorem 20 together with the described complexities of emptiness checking yields improved upper bounds on the runtime complexity of solving obliging games, shown in the table below; we let n denote the number of nodes; m the number of edges; b the size of a nondeterministic Büchi automaton accepting α_W ; and t the time required for emptiness checking an Emerson-Lei automaton of size n with acceptance condition $\alpha_S \wedge \alpha_W$; finally we let o abbreviate $\max(|W|, |S|)$. Recall that the approach from [6] reduces an obliging game to a standard game in which the objective α' is the conjunction of α_S with a Büchi objective, incurring blowup b on the arena size. This game then can be transformed to a parity game \mathcal{G} (with parameters v, e and r) incurring additional blowup for the LAR transformation of α' to a parity objective. Notice the definition of e (an underapproximation of the number of edges in \mathcal{G}) in column 4 and its usage in column 5.

For instance for an obliging game with objectives $\alpha_S = \text{Streett}(d)$ and $\alpha_W = \text{generalized Büchi}(d)$ consisting of the Streett requests, the approach from [6] reduces the game to a parity game \mathcal{G} with $d!nd$ nodes, at least $d!md$ edges, and $2d$ priorities; such a game can be solved in time $\mathcal{O}(d!md^6 (d!n)^5)$. In contrast, emptiness checking for $\alpha_S \wedge \alpha_W$ -automata is just emptiness checking for generalized Büchi automata so that our new algorithm solves such games in time $\mathcal{O}(d!md^2 (d!n)^5)$. For objectives $\alpha_S = \text{Rabin}(d)$ and $\alpha_W = \text{Streett}(k)$, the approach from [6] has time complexity $\mathcal{O}(m(d!k2^k)^6 n^5)$ while our algorithm has time

■ **Table 2** Comparison of runtime complexities for solving obliging games of various types.

type of α_S	type of α_W	b	$ \mathcal{G} (v, e, r)$ [6]	time [6]	t [1]	time here
parity(d)	Rabin(k)	$k + 1$	dnb, dmb, d	$\mathcal{O}(e(dnb)^{\log d})$	$\mathcal{O}(mo^2)$	$\mathcal{O}(n^{\log d+1}dt)$
	Streett(k)	$2^k k$			$\mathcal{O}(mo^2)$	
	EL(k)	$2^k k$			$\mathcal{O}(mo^2 2^o)$	
Rabin(d) or Streett(d)	Rabin(k)	$k + 1$	$d!nb, d!mb, 2d$	$\mathcal{O}(e(d!nb)^5)$	$\mathcal{O}(mo^3)$	$\mathcal{O}((d!n)^5 d!dt)$
	Streett(k)	$2^k k$			$\mathcal{O}(mo^3)$	
	EL(k)	$2^k k$			$\mathcal{O}(mo^2 2^o)$	
EL(d)	EL(k)	$2^k k$	$d!nb, d!mb, 2d$	$\mathcal{O}(e(d!nb)^5)$	$\mathcal{O}(mo^2 2^o)$	$\mathcal{O}((d!n)^5 d!dt)$
Streett(d)	g.Büchi(d)	d	$d!nb, d!mb, 2d$	$\mathcal{O}(e(d!nb)^5)$	$\mathcal{O}(md)$	$\mathcal{O}((d!n)^5 d!dt)$
GR[1](d, k)	g.Büchi(d)	d	$dknb, dkmb, 3$	$\mathcal{O}(e(dknb)^2)$	$\mathcal{O}(md)$	$\mathcal{O}((dk)^3 n^2 dt)$

complexity just $\mathcal{O}(m(d!)^6 n^5 do^3)$; this is due to the fact that Büchi automata that recognize Streett objectives are of exponential size (as they have to guess a set of Streett pairs and then verify their satisfaction), while emptiness checking for Streett(d) \wedge Rabin(k)-automata can be done in time cubic in o .

6 Conclusion

We propose a new angle of looking at the solution of obliging games. In contrast to previous approaches that have been based on single-step game reasoning, our method requires players to make promises about their long-term future behavior, which we formalize using the concept of certificates (or, more generally, witnesses). This new approach to obliging games enables us to not only show their determinacy (with strategies that contain additional information), but to also significantly improve previously existing upper bounds both on the size of graciously winning strategies, and on the worst-case runtime complexity of the solution of such games.

Technically, we use our new approach to show that the strategy sizes for Emerson-Lei obliging games with strong objective α_S and weak objective α_W are linear in the number $|W|$ of colors used in the weak objective; we obtain a similar polynomial dependence on $|W|$ for the runtime of solving obliging games, however with the important exception of the case where α_W is a full Emerson-Lei objective that cannot be expressed by a simpler (e.g. Rabin or Streett) objective. In previous approaches, those dependencies on $|W|$ have been, in general, exponential. Thereby we show that the strategy complexity of α_S / α_W obliging games is not significantly higher than that of standard games with objective just α_S , and that in many cases, this holds for the runtime complexity as well.

We leave the existence of canonical certificates (cf. Remark 10) as an open question for future work; such canonical certificates would allow for the extraction of yet smaller graciously winning strategies for obliging games.

We solve certificate games by using the LAR reduction to obtain equivalent parity games and then solving these parity games by fixpoint acceleration, computing nested fixpoints of a function that checks for certificate existence. We conjecture that it is possible to directly compute the more involved nested fixpoint associated to Emerson-Lei objectives (as given in [15]) over the original game arena; this would avoid the LAR reduction step in the solution process.

References

- 1 Christel Baier, Frantisek Blahoudek, Alexandre Duret-Lutz, Joachim Klein, David Müller, and Jan Strejcek. Generic emptiness check for fun and profit. In *Automated Technology for Verification and Analysis, ATVA 2019*, volume 11781 of *LNCS*, pages 445–461. Springer, 2019. doi:10.1007/978-3-030-31784-3_26.
- 2 Roderick Bloem, Rüdiger Ehlers, Swen Jacobs, and Robert Könighofer. How to handle assumptions in synthesis. In *Workshop on Synthesis, SYNT 2014*, volume 157 of *EPTCS*, pages 34–50, 2014. doi:10.4204/EPTCS.157.7.
- 3 Roderick Bloem, Rüdiger Ehlers, and Robert Könighofer. Cooperative reactive synthesis. In *Automated Technology for Verification and Analysis, ATVA 2015*, volume 9364 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2015. doi:10.1007/978-3-319-24953-7_29.
- 4 Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa’ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012. doi:10.1016/J.JCSS.2011.08.007.
- 5 Julian C. Bradfield and Igor Walukiewicz. The μ -calculus and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 871–919. Springer, 2018. doi:10.1007/978-3-319-10575-8_26.
- 6 Krishnendu Chatterjee, Florian Horn, and Christof Löding. Obliging games. In *Concurrency Theory, 21th International Conference, CONCUR 2010*, volume 6269 of *LNCS*, pages 284–296. Springer, 2010. doi:10.1007/978-3-642-15375-4_20.
- 7 Stefan Dziembowski, Marcin Jurdzinski, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Logic in Computer Science, LICS 1997*, pages 99–110. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614939.
- 8 Javier Esparza, Jan Kretínský, Jean-François Raskin, and Salomon Sickert. From linear temporal logic and limit-deterministic Büchi automata to deterministic parity automata. *Int. J. Softw. Tools Technol. Transf.*, 24(4):635–659, 2022. doi:10.1007/S10009-022-00663-1.
- 9 Oliver Friedmann and Martin Lange. Deciding the unguarded modal μ -calculus. *J. Appl. Non Class. Logics*, 23(4):353–371, 2013. doi:10.1080/11663081.2013.861181.
- 10 Oliver Friedmann, Markus Latte, and Martin Lange. Satisfiability games for branching-time logics. *Log. Methods Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:5)2013.
- 11 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.
- 12 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *Symposium on Theory of Computing, STOC 1982*, pages 60–65. ACM, 1982. doi:10.1145/800070.802177.
- 13 Daniel Hausmann. Faster game solving by fixpoint acceleration. *CoRR*, abs/2404.13687, 2024. arXiv:2404.13687.
- 14 Daniel Hausmann. Faster game solving by fixpoint acceleration. In *Fixed Points in Computer Science, FICS 2024*, EPTCS, 2024, to appear.
- 15 Daniel Hausmann, Mathieu Lehaut, and Nir Piterman. Symbolic solution of Emerson-Lei games for reactive synthesis. In *Foundations of Software Science and Computation Structures, FoSSaCS 2024*, volume 14574 of *LNCS*, pages 55–78. Springer, 2024. doi:10.1007/978-3-031-57228-9_4.
- 16 Daniel Hausmann and Lutz Schröder. Game-based local model checking for the coalgebraic μ -calculus. In *Concurrency Theory, CONCUR 2019*, volume 140 of *LIPICs*, pages 35:1–35:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.35.
- 17 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *Mathematical Foundations of Computer Science, MFCS 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 495–506. Springer, 2005. doi:10.1007/11549345_43.

- 18 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, 57(1-2):3–36, 2020. doi:10.1007/s00236-019-00349-3.
- 19 Rupak Majumdar, Nir Piterman, and Anne-Kathrin Schmuck. Environmentally-friendly GR(1) synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2019*, volume 11428 of *LNCS*, pages 229–246. Springer, 2019. doi:10.1007/978-3-030-17465-1_13.
- 20 Rupak Majumdar and Anne-Kathrin Schmuck. Supervisory controller synthesis for nonterminating processes is an obliging game. *IEEE Trans. Autom. Control.*, 68(1):385–392, 2023. doi:10.1109/TAC.2022.3143108.
- 21 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. doi:10.2307/1971035.
- 22 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Principles of Programming Languages, POPL 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.
- 23 Tom van Dijk, Feije van Abbema, and Naum Tomov. Knor: reactive synthesis using oink. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2024*, volume 14570 of *LNCS*, pages 103–122. Springer, 2024. doi:10.1007/978-3-031-57246-3_7.

Strategic Dominance: A New Preorder for Nondeterministic Processes

Thomas A. Henzinger   

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Nicolas Mazzocchi   

Slovak University of Technology in Bratislava, Slovak Republic

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

N. Ege Saraç   

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

We study the following refinement relation between nondeterministic state-transition models: model \mathcal{B} *strategically dominates* model \mathcal{A} iff every deterministic refinement of \mathcal{A} is language contained in some deterministic refinement of \mathcal{B} . While language containment is trace inclusion, and the (fair) simulation preorder coincides with tree inclusion, strategic dominance falls strictly between the two and can be characterized as “strategy inclusion” between \mathcal{A} and \mathcal{B} : every strategy that resolves the nondeterminism of \mathcal{A} is dominated by a strategy that resolves the nondeterminism of \mathcal{B} . Strategic dominance can be checked in 2-EXPTIME by a decidable first-order Presburger logic with quantification over words and strategies, called *resolver logic*. We give several other applications of resolver logic, including checking the co-safety, co-liveness, and history-determinism of boolean and quantitative automata, and checking the inclusion between hyperproperties that are specified by nondeterministic boolean and quantitative automata.

2012 ACM Subject Classification Theory of computation \rightarrow Formal languages and automata theory; Theory of computation \rightarrow Logic and verification; Theory of computation \rightarrow Program reasoning

Keywords and phrases quantitative automata, refinement relation, resolver, strategy, history-determinism

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.29

Funding This work was supported in part by the ERC-2020-AdG 101020093. N. Mazzocchi was affiliated with ISTA when this work was submitted for publication.

Acknowledgements We thank the anonymous reviewers for their helpful comments.

1 Introduction

Nondeterminism is a powerful mechanism for varying the degree of detail shown in a state-transition model of a system. Intuitively, a nondeterministic model captures the set of possible deterministic implementations. Consider the process model $ab + ac$. This model allows two possible deterministic implementations, ab and ac . Mathematically, each deterministic implementation of a nondeterministic model corresponds to a *strategy* for resolving the nondeterminism, i.e., a function f that maps a finite run h through the model (the “history”) and a new letter σ to a successor state $f(h, \sigma)$ allowed by the model. Consider the recursive process model $X = abX + acX$, which repeatedly chooses either the ab branch or the ac branch. There are infinitely many different strategies to resolve the repeated nondeterminism, e.g., the strategy f_{prime} whose n th choice is the ab branch if $n = 1$ or n is prime, and whose n th choice is the ac branch when $n > 1$ and n is composite.

Two models that describe the same system at different levels of detail are related by a preorder. Milner’s simulation preorder and the trace-inclusion preorder represent two particularly paradigmatic and widely studied examples of preorders on state-transition



© Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 29; pp. 29:1–29:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

models, but many variations and other refinement preorders can be found in the literature [30, 29, 4, 9, 26]. In its purest form, the *linear-time* view of model refinement postulates that for two models \mathcal{A} and \mathcal{B} , for \mathcal{B} to describe the same system as \mathcal{A} at a higher level of abstraction, the less precise model \mathcal{B} must allow all traces that are possible in the more refined model \mathcal{A} while potentially allowing more traces. In contrast, the pure *branching-time* view of model refinement postulates that for two models \mathcal{A} and \mathcal{B} , for \mathcal{B} to describe the same system as \mathcal{A} at a higher level of abstraction, the less precise model \mathcal{B} must allow all subtrees of the full computation tree of the more refined model \mathcal{A} (modulo its isomorphic subtrees) while potentially allowing more trees. This tree-inclusion view of branching time corresponds to the simulation preorder and its generalization to fair simulation [20].

In this paper, we introduce and study a third fundamental view of model refinement – *strategy inclusion* – which lies strictly between trace inclusion and tree inclusion. According to the strategic view, for \mathcal{B} to describe the same system as \mathcal{A} at a higher level of abstraction, the less precise model \mathcal{B} must allow all deterministic implementations that are possible for the more refined model \mathcal{A} (but \mathcal{B} may allow strictly more implementations than \mathcal{A}). Mathematically, we say that \mathcal{B} (*strategically*) *dominates* \mathcal{A} if for every strategy f resolving the nondeterminism of \mathcal{A} , there exists a strategy g for resolving the nondeterminism of \mathcal{B} such that every trace of \mathcal{A}^f (the deterministic result of applying the strategy f to the nondeterministic model \mathcal{A}) corresponds to a trace of \mathcal{B}^g .

Strategic dominance, like simulation, is a branching-time preorder, but while simulation corresponds to inclusion of all subtrees of the full computation tree of a model, dominance corresponds to the inclusion of all *deterministic* subtrees. To see that dominance is strictly coarser than simulation, recall the model $X = abX + acX$ and compare it with the model $Y = ababY + abacY + acabY + acacY$. Both models X and Y have the same traces, but Y does not simulate X . Nonetheless, for every strategy resolving the nondeterminism of X there exists a strategy resolving the nondeterminism of Y that produces same infinite trace (and vice versa). In particular, the strategy g_{prime} for Y that dominates the strategy f_{prime} for X makes the following choices: initially g_{prime} chooses the $abab$ branch (since 2 is prime) and thereafter, for all $n > 1$, the n th choice of g_{prime} is $abac$ if $2n + 1$ is prime, and $acac$ otherwise (because $2n + 2$ is never prime). To see that strategic dominance is strictly finer than trace inclusion, the standard example of comparing $a(b + c)$ with $ab + ac$ will do; these two process models are trace equivalent, but only the former has a deterministic implementation that includes both traces.

Relation to prior work. Our motivation for studying the strategic view of model refinement originated from the definition of *history-determinism* [22, 8]. A state-transition model \mathcal{A} is history-deterministic if there exists a strategy for resolving the nondeterminism of \mathcal{A} which captures exactly the language of \mathcal{A} . In other words, the nondeterminism of a history-deterministic model can be resolved on-the-fly, by looking only at the history, without making guesses about the future.

Strategies (a.k.a. policies) are a central concept of game theory, and our resolvers correspond mathematically to deterministic strategies of games played on graphs. However, we study the *single-player* case – where the player resolves the nondeterminism of a state-transition model – and not the multi-player case that arises in game models [3, 12]. Besides general strategies, we consider the special cases of *finite-state* strategies (which can remember only a finite number of bits about the history), and of *positional* strategies (which have no memory about the history). Positional (a.k.a. memoryless) strategies correspond, in our setting, to the removal of nondeterministic edges from a model (“edge pruning”). To the best of our knowledge, strategic dominance is a novel relation that has not been studied before.

In particular, the seminal works on the linear time-branching time spectrum of sequential processes [30, 29] do not present a comparable relation. In general, these works do not involve a game-theoretic view of nondeterminism, which is a crucial aspect captured through our use of resolvers. In this way, the game-theoretic view adds a new dimension to the spectrum of process preorders.

For generality, we study strategies for state-transition models in a *quantitative* setting, where all states have outgoing transitions and all transitions have numeric weights [11]. In this setting, every finite path through a state-transition model can be extended, every infinite path is assigned a numeric value, and the values of different paths can be compared. The trace preorder between \mathcal{A} and \mathcal{B} requires that for every infinite word w , for every run of \mathcal{A} on w there exists a run of \mathcal{B} on w of equal or greater value. All refinement relations we consider are refinements of the trace preorder. The quantitative setting generalizes the boolean setting (take $\{0, 1\}$ as the value set, and assign the value 1 to a run iff the run is accepted by the model), and it generalizes many common acceptance conditions on finite and infinite runs (Sup and Inf values correspond to reachability and safety acceptance; LimSup and LimInf values to Büchi and coBüchi acceptance). Extensions of simulation and inclusion preorders to this setting have been studied in [11].

Contributions of this paper. Our results are threefold. First, we define *resolver logic* as a first-order logic that is interpreted over a set QA of quantitative finite automata over infinite words. Resolver logic quantifies over infinite words w , strategies f that resolve the nondeterminism of automata $\mathcal{A} \in \text{QA}$ (so-called “resolvers”), and natural numbers. The existentially quantified formulas are built from terms of the form $\mathcal{A}^f(w)$ using Presburger arithmetic. The term $\mathcal{A}^f(w)$ denotes the value of the unique run of the automaton \mathcal{A} over the word w when all nondeterministic choices are resolved by the strategy f . We show that model-checking problem for resolver logic – i.e., the problem of deciding if a closed formula φ is true over a given set \mathcal{A} of automata – can be solved in $d\text{-EXPTIME}$, where d is the number of quantifier switches in φ . Our model-checking algorithm uses automata-theoretic constructions over parity tree automata that represent resolvers. A main difference of resolver logic to strategy logics [12, 25], besides the handling of quantitative constraints and Presburger arithmetic, lies in the quantification over infinite words. This quantification is not present in strategy logics, and lets us define strategic dominance and other inclusion-based relations.

Second, we define in resolver logic *eight different resolver relations* between quantitative automata, including the strategic-dominance relation explained above. The eight relations are obtained from each other by reordering quantifiers in resolver logic. We show that six of the eight relations are preorders: one coincides with simulation, four coincide with trace inclusion, and the sixth – strategic dominance – lies strictly between simulation and trace inclusion. The remaining two relations are finer than simulation and transitive, but not reflexive. Since all eight relations are defined in resolver logic, they can be decided using the model-checking algorithm for resolver logic. We also specialize our expressiveness and decidability results to specific value functions (Sup; Inf; LimSup; LimInf) and to restricted classes of strategies (positional; finite-state).

Third, we provide three more applications for resolver logic, in addition to checking strategic dominance. These applications show that resolvers play a central role in many different automata-theoretic problems.

Three more applications. Our first application concerns the *co-safety and co-liveness* of boolean and quantitative automata [21, 5]. In [5], while the authors solved the problems of deciding safety or liveness, they left open the *bottom-value problem* for quantitative automata, which needs to be solved when checking if a quantitative automaton specifies a co-safety or co-liveness property. The bottom value of a quantitative automaton \mathcal{A} can be defined as the infimum over all words w of the supremum over all resolvers f of the value $\mathcal{A}^f(w)$. The bottom value, and all similarly defined values, can therefore be computed using the model-checking algorithm for resolver logic.

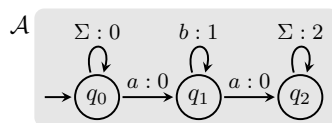
Our second application concerns the *history-determinism* of boolean and quantitative automata. For a model \mathcal{B} to strategically dominate \mathcal{A} , different resolvers for \mathcal{A} may give rise to different resolvers for \mathcal{B} . Alternatively, one may postulate that in refinement, all resolvers for \mathcal{A} should be “dominated” by the same resolver for \mathcal{B} , that is, a single deterministic implementation of \mathcal{B} should capture all possible deterministic implementations of \mathcal{A} . Such *blind domination* gives rise to a nonreflexive relation on quantitative automata which implies simulation. We show that a quantitative automaton is history-deterministic iff the automaton blindly dominates itself. Consequently, our model-checking algorithm for resolver logic can be used to check history-determinism and generalizes the algorithm provided in [22] for checking the history determinism of boolean automata.

Our third application concerns the *specification of hyperproperties*. A hyperproperty is a set of properties [14]; hyperproperties occur in many different application contexts such as system security. A nondeterministic boolean automaton \mathcal{A} can be viewed as specifying a hyperproperty $\llbracket \mathcal{A} \rrbracket$ in the following natural way: if $R(\mathcal{A})$ is the set of all possible resolvers for the nondeterminism of \mathcal{A} , then $\llbracket \mathcal{A} \rrbracket = \{\mathcal{A}^f \mid f \in R(\mathcal{A})\}$, where \mathcal{A}^f is the property (or “language”) that is specified by \mathcal{A} if its nondeterminism is resolved by f . In the same way, nondeterministic quantitative automata can be used to specify quantitative hyperproperties as sets of quantitative languages [11]. We show that there are simple automata that specify the boolean hyperproperties that contain all safety (resp. co-safety) properties, which cannot be specified in HyperLTL [13]. However, there are also boolean hyperproperties that can be specified in HyperLTL but not by applying resolvers to nondeterministic automata. Finally, we show how resolver logic can decide the inclusion problem for resolver-specified hyperproperties.

2 Definitional Framework

Let $\Sigma = \{a, b, \dots\}$ be a finite alphabet of letters. An infinite (resp. finite) word (a.k.a. trace or execution) is an infinite (resp. finite) sequence of letters $w \in \Sigma^\omega$ (resp. $u \in \Sigma^*$). Given $u \in \Sigma^*$ and $w \in \Sigma^* \cup \Sigma^\omega$, we write $u \prec w$ when u is a strict prefix of w . We denote by $|w|$ the length of $w \in \Sigma^* \cup \Sigma^\omega$ and, given $a \in \Sigma$, by $|w|_a$ the number of occurrences of a in w . We assume that the reader is familiar with formal language theory.

► **Definition 2.1** (automaton). A (quantitative) automaton is a tuple $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ where Σ is a finite alphabet, Q is a finite set of states, $s \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $\mu: \Delta \rightarrow \mathbb{Q}$ is a weight function, and $\nu: \mathbb{Q}^\omega \rightarrow \mathbb{R}$ is a value function. The size of \mathcal{A} is defined by $|Q| + \sum_{\delta \in \Delta} 1 + \log_2(\mu(\delta))$. A run π over a finite (resp. an infinite) word $w = \sigma_0\sigma_1\dots$ is a finite (resp. an infinite) sequence $\pi = q_0q_1q_2\dots$ such that $q_0 = s$ is the initial state of \mathcal{A} and $(q_i, \sigma_i, q_{i+1}) \in \Delta$ holds for all $0 \leq i < |w|$ (resp. $i \in \mathbb{N}$). A history is a run over a finite word, and we denote the set of histories of \mathcal{A} by $\Pi_{\mathcal{A}}$. The weight sequence $\mu(\pi)$ of a run π is defined by $x_0x_1\dots$ where $x_i = \mu(q_i, \sigma_i, q_{i+1})$ for all $i \in \mathbb{N}$. The value of a run π is defined as $\nu(\mu(\pi))$. In this paper, we consider automata with $\nu \in \{\text{Inf}, \text{Sup}, \text{LimInf}, \text{LimSup}\}$ and without loss of generality $\mu: \Delta \rightarrow \mathbb{N}$ since a finite set of rational weights can be scaled and shifted to naturals and back.



■ **Figure 1** A LimSup-automaton \mathcal{A} over $\Sigma = \{a, b\}$ with $R^{\text{pos}}(\mathcal{A}) \subsetneq R^{\text{fin}}(\mathcal{A}) \subsetneq R(\mathcal{A})$.

In general, automata resolve their nondeterminism by taking the sup over its set of runs on a given word. We take an alternative view and pair every automaton with a *resolver* – an explicit description of how nondeterminism is resolved, which is a central concept in this work. Given a finite prefix of a run and the next input letter, a resolver determines the next state of the automaton.

► **Definition 2.2** (resolver). Let $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ be an automaton. A resolver for \mathcal{A} is a function $f: \Pi_{\mathcal{A}} \times \Sigma \rightarrow Q$ such that for every history $h = q_0\sigma_0q_1\sigma_1 \dots q_n \in \Pi_{\mathcal{A}}$ and every $\sigma \in \Sigma$ we have $(q_n, \sigma, f(h, \sigma)) \in \Delta$. A resolver f for \mathcal{A} and a word $w = \sigma_1\sigma_2 \dots \in \Sigma^\omega$ produce a unique infinite run $\pi_{f,w} = q_0\sigma_1q_1\sigma_2 \dots$ of \mathcal{A} such that $q_0 = s$ and $f(q_0\sigma_1 \dots q_{i-1}, \sigma_i) = q_i$ for all $i \geq 1$. Given an automaton \mathcal{A} , we denote by $R(\mathcal{A})$ the set of its resolvers. Given a resolver $f \in R(\mathcal{A})$, we define the quantitative language $\mathcal{A}^f: w \mapsto \nu(\mu(\pi_{f,w}))$ where $w \in \Sigma^\omega$. We also define the quantitative language $\mathcal{A}_{\text{sup}}: w \mapsto \sup_{f \in R(\mathcal{A})} \mathcal{A}^f(w)$, which is the standard interpretation of nondeterminism for automata.

We define finite-memory and positional resolvers the usual way: a resolver f is *finite-memory* iff it can be implemented by a finite-state machine, and it is *positional* iff the output of f only depends on the last state in the input history and the incoming letter (see [17, Sec. 1.5] for the formal definitions). Given an automaton \mathcal{A} , we denote by $R^{\text{fin}}(\mathcal{A})$ the set of its finite-memory resolvers, and by $R^{\text{pos}}(\mathcal{A})$ set of its positional resolvers. Let us demonstrate the notion of resolvers.

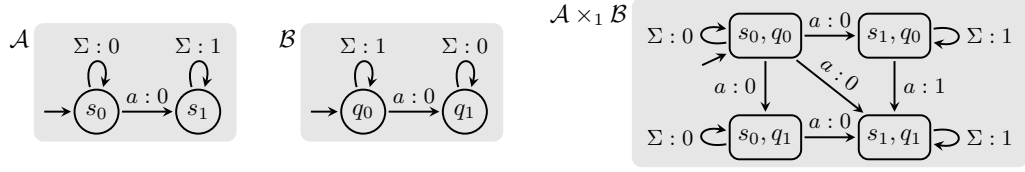
► **Example 2.3.** Let \mathcal{A} be a LimSup-automaton over the alphabet $\Sigma = \{a, b\}$ as in Figure 1 and observe that the only source of nondeterminism is the transitions (q_0, a, q_0) and (q_0, a, q_1) .

Consider a resolver f_1 that maps any history of the form $q_0(bq_0)^*$ followed by a to q_1 . Intuitively, it is a *positional* resolver because it ignores the transition (q_0, a, q_0) and its output only depends on the current state and the next letter. It denotes the language \mathcal{A}^{f_1} that maps a given word w to 0 if w has no a , to 1 if w has exactly one a , and to 2 otherwise.

Now, consider a resolver f_2 that maps the histories of the form $q_0(bq_0)^*$ followed by a to q_0 , and those of the form $q_0(bq_0)^*aq_0(bq_0)^*$ followed by a to q_1 . Intuitively, it is a *finite-state* resolver because it only distinguishes between a occurring once or twice or neither. The language \mathcal{A}^{f_2} then maps a word w to 0 if w has at most one a , to 1 if w has exactly two a s, and to 2 otherwise.

Finally, consider a resolver f_3 that maps every history that ends at q_0 with the incoming letter a to q_1 if the given history is of the form $q_0(\Sigma q_0)^*(bq_0)^p$ where p is a prime, and to q_0 otherwise. Intuitively, it is an *infinite-memory* resolver because it needs to store the length of every block of bs , which is not bounded, and check whether it is prime. The language \mathcal{A}^{f_3} maps a word w to 0 if it has no prefix $u = vb^p a$ with $v \in \Sigma^*$, to 1 if it has such a prefix u and $w = ub^\omega$, and to 2 otherwise.

Next, we introduce the notion of *partial resolvers*. In contrast to (nonpartial) resolvers, partial resolvers output a set of successor states for a given history and letter.



■ **Figure 2** Two LimSup-automata \mathcal{A} and \mathcal{B} and the product $\mathcal{A} \times_1 \mathcal{B}$.

► **Definition 2.4** (partial resolver). Let $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ be an automaton. A partial resolver for \mathcal{A} is a function $f: \Pi_{\mathcal{A}} \times \Sigma \rightarrow 2^Q$ such that for every history $h = q_0 \sigma_0 q_1 \sigma_1 \dots q_n \in \Pi_{\mathcal{A}}$ and every $\sigma \in \Sigma$ we have $\{(q_n, \sigma, q) \mid q \in f(h, \sigma)\} \subseteq \Delta$. A collection of partial resolvers f_1, \dots, f_n for \mathcal{A} is said to be conclusive when $|\bigcap_{i=1}^n f_i(h, \sigma)| = 1$ for all $h \in \Pi_{\mathcal{A}}$ and all $\sigma \in \Sigma$. Given a conclusive collection of resolvers f_1, \dots, f_n , we denote by $\mathcal{A}^{f_1, \dots, f_n}$ the quantitative language \mathcal{A}^f where f is a resolver defined by $f(h, \sigma) = \bigcap_{i=1}^n f_i(h, \sigma)$ for all $h \in \Pi_{\mathcal{A}}$ and all $\sigma \in \Sigma$.

Partial resolvers are particularly useful when we consider products of automata. In particular, we will use these objects to capture simulation-like relations in Section 3.

► **Definition 2.5** (synchronized product). Let \mathcal{A}_1 and \mathcal{A}_2 be two automata such that $\mathcal{A}_i = (\Sigma, Q_i, s_i, \Delta_i, \mu_i, \nu_i)$ for $i \in \{1, 2\}$. For $k \in \{1, 2\}$, the (synchronized) product $\mathcal{A}_1 \times_k \mathcal{A}_2$ corresponds to the input-synchronization of \mathcal{A}_1 and \mathcal{A}_2 where the transition weights are taken from \mathcal{A}_k . Formally, $\mathcal{A}_1 \times_k \mathcal{A}_2 = (\Sigma, Q_1 \times Q_2, (s_1, s_2), \Delta, \mu, \nu_k)$ where the transition relation is such that $((q_1, q_2), \sigma, (q'_1, q'_2)) \in \Delta$ if and only if $(q_i, \sigma, q'_i) \in \Delta_k$ for $i \in \{1, 2\}$, and the weight function is such that $\mu((q_1, q_2), \sigma, (q'_1, q'_2)) = \mu_k(q_k, \sigma, q'_k)$.

Given $i \in \{1, 2\}$, we denote by $R_i^S(\mathcal{A}_1, \mathcal{A}_2)$ the set of partial resolvers operating non-partially on the i th component of any product between \mathcal{A}_1 and \mathcal{A}_2 . Formally, every $f \in R_i(\mathcal{A}_1, \mathcal{A}_2)$ is a partial resolver that satisfies the following: for all $h \in \Pi_{\mathcal{A}_1 \times_i \mathcal{A}_2}$, all $(q_1, q_2) \in Q_1 \times Q_2$ and all $\sigma \in \Sigma$, there exists $(q_i, \sigma, q'_i) \in \Delta_i$ such that $f(h \cdot (q_1, q_2), \sigma) = \{(q'_1, q'_2) \mid (q_{3-i}, \sigma, q'_{3-i}) \in \Delta_{3-i}\}$.

Let us demonstrate the notions of partial resolvers and synchronized products.

► **Example 2.6.** Let \mathcal{A} and \mathcal{B} be as in Figure 2. Because of the loop in their initial states, as in Example 2.3, both automata have infinitely many resolvers. Their product $\mathcal{A} \times_1 \mathcal{B}$ is also shown in Figure 2. Intuitively, the product is similar to the boolean construction with difference of handling the transition weights instead of the accepting states. For $\mathcal{A} \times_1 \mathcal{B}$, the transition weights are obtained from the corresponding ones in \mathcal{A} .

Consider the positional resolver f of \mathcal{A} that moves from s_0 to s_1 as soon as a occurs. Observe that there is a partial resolver f' of $\mathcal{A} \times_1 \mathcal{B}$, namely $f' \in R_1(\mathcal{A}, \mathcal{B})$, that imitates f . In particular, $f'((s_0, q_0), a) = \{(s_1, q_0), (s_1, q_1)\}$ since f moves \mathcal{A} from s_0 to s_1 with a , but it is not specified how the \mathcal{B} -component of the product resolves this nondeterministic transition. Now, suppose we have $g' \in R_2(\mathcal{A}, \mathcal{B})$ such that $g'((s_0, q_0), a) = \{(s_0, q_1), (s_1, q_1)\}$. The two partial resolvers f' and g' together are conclusive for $\mathcal{A} \times_1 \mathcal{B}$, as witnessed by $f'((s_0, q_0), a) \cap g'((s_0, q_0), a) = \{(s_1, q_1)\}$.

► **Remark 2.7.** As demonstrated in Example 2.6, when we consider partial resolvers over product automata, the partial resolvers operating on the distinct components are conclusive for the product. Formally, let \mathcal{A} and \mathcal{B} be two automata and consider one of their products. Every pair of partial resolvers $f \in R_1(\mathcal{A}, \mathcal{B})$ and $g \in R_2(\mathcal{A}, \mathcal{B})$ is conclusive for the product automaton by definition, and thus corresponds to a (non-partial) resolver over the product.

3 Strategic Dominance and Other Resolver-Based Relations

Given two automata \mathcal{A} and \mathcal{B} , we investigate the relations between the problems defined in Figure 3. We denote the *strategic dominance* relation by \preceq : the automaton \mathcal{A} is strategically dominated by \mathcal{B} , denoted $\mathcal{A} \preceq \mathcal{B}$, iff for all resolvers $f \in R(\mathcal{A})$ there exists a resolver $g \in R(\mathcal{B})$ such that $\mathcal{A}^f(w) \leq \mathcal{B}^g(w)$ for all words $w \in \Sigma^\omega$. Intuitively, this holds when each deterministic implementation of \mathcal{A} can be countered by some deterministic implementation of \mathcal{B} that provides a value at least as large for each word.

We define and study other resolver-based relations and compare their expressiveness. In \sqsubseteq , the automaton \mathcal{B} has the freedom to choose a resolver per input word (unlike in \preceq). The relations \blacktriangleleft and \blacksquare can be seen as variants of \preceq and \sqsubseteq where the resolvers of \mathcal{B} are *blind*, meaning that they cannot depend on the resolvers of \mathcal{A} .

To capture simulation-like relations, we additionally define the relations \preceq_\times , $\blacktriangleleft_\times$, \sqsubseteq_\times , \blacksquare_\times that relate product automata through their partial resolvers. In particular, we show that \preceq_\times , the product-based strategic dominance, coincides with simulation. Thanks to Remark 2.7, we are able to reason about these relations the same way we do for non-partial resolvers.

In addition to these resolver relations, we denote the trace-inclusion preorder by \subseteq and the simulation by \preceq . To define simulation formally, let us recall quantitative simulation games [11]: Let $\mathcal{A} = (\Sigma, Q_{\mathcal{A}}, s_{\mathcal{A}}, \Delta_{\mathcal{A}}, \mu_{\mathcal{A}}, \nu_{\mathcal{A}})$ and $\mathcal{B} = (\Sigma, Q_{\mathcal{B}}, s_{\mathcal{B}}, \Delta_{\mathcal{B}}, \mu_{\mathcal{B}}, \nu_{\mathcal{B}})$. A strategy τ for Challenger is a function from $(Q_{\mathcal{A}} \times Q_{\mathcal{B}})^*$ to $\Sigma \times Q_{\mathcal{A}}$ satisfying for all $\pi = (q_1, p_1) \dots (q_n, p_n) \in (Q_{\mathcal{A}} \times Q_{\mathcal{B}})^*$, if $\tau(\pi) = (\sigma, q)$ then $(q_n, \sigma, q) \in \Delta_{\mathcal{A}}$. Given a strategy τ for Challenger, the set of outcomes is the set of pairs $(q_0 \sigma_1 q_1 \sigma_2 q_2 \dots, p_0 \sigma_1 p_1 \sigma_2 p_2 \dots)$ of runs such that $q_0 = s_{\mathcal{A}}$, $p_0 = s_{\mathcal{B}}$, and for all $i \geq 0$ we have $(\sigma_{i+1}, q_{i+1}) = \tau((q_0, p_0) \dots (q_i, p_i))$ and $(p_i, \sigma_i, p_{i+1}) \in \Delta_{\mathcal{B}}$. A strategy τ for Challenger is winning iff $\nu_{\mathcal{A}}(\mu_{\mathcal{A}}(r_1)) > \nu_{\mathcal{B}}(\mu_{\mathcal{B}}(r_2))$ for all outcomes (r_1, r_2) of τ .

Given a problem instance $\mathcal{A} \sim \mathcal{B}$ where \sim is one of the relations in Figure 3, we write $\mathcal{A} \sim^{\text{fin}} \mathcal{B}$ (resp. $\mathcal{A} \sim^{\text{pos}} \mathcal{B}$) for the restriction of the corresponding problem statement to finite-memory (resp. positional) resolvers. For example, $\mathcal{A} \preceq^{\text{fin}} \mathcal{B}$ iff for all $f \in R^{\text{fin}}(\mathcal{A})$ there exists $g \in R^{\text{fin}}(\mathcal{B})$ such that $\mathcal{A}^f(w) \leq \mathcal{B}^g(w)$ for all $w \in \Sigma^\omega$.

► **Remark 3.1.** The results in this paper hold for quantitative Inf-, Sup-, LimSup-, and LimInf-automata as well as boolean safety, reachability, Büchi, and coBüchi automata. Moreover, they also hold when restricted to finite-state or positional resolvers.

We start with a short lemma showing that the supremum over the values of any word w is attainable by some run over w , which follows from the proof of [11, Thm. 3].

► **Lemma 3.2.** *Let $\nu \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$ and \mathcal{A} be a ν -automaton. For every $w \in \Sigma^\omega$ there exist $f \in R(\mathcal{A})$ such that $\mathcal{A}^f(w) = \mathcal{A}_{\text{sup}}(w)$.*

3.1 Implications Between Resolver Relations

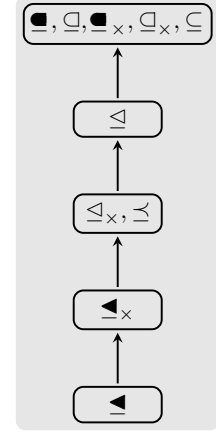
We now prove the implications in Figure 3 without any memory constraints on the resolvers. We start with showing that some of these relations coincide with inclusion.

► **Proposition 3.3.** *Let $\nu_1, \nu_2 \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. For all ν_1 -automata \mathcal{A} and all ν_2 -automata \mathcal{B} , we have $\mathcal{A} \sqsubseteq \mathcal{B}$ iff $\mathcal{A} \blacksquare \mathcal{B}$ iff $\mathcal{A} \sqsubseteq_\times \mathcal{B}$ iff $\mathcal{A} \blacksquare_\times \mathcal{B}$ iff $\mathcal{A} \subseteq \mathcal{B}$. This is equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

Next, we show an equivalent formulation for simulation.

► **Proposition 3.4.** *Let $\nu_1, \nu_2 \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. For all ν_1 -automata \mathcal{A} and all ν_2 -automata \mathcal{B} , we have $\mathcal{A} \preceq_\times \mathcal{B}$ iff $\mathcal{A} \preceq \mathcal{B}$. This is equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

Notation	Problem statement
$\mathcal{A} \subseteq \mathcal{B}$	$\forall w \in \Sigma^\omega : \mathcal{A}_{\text{sup}}(w) \leq \mathcal{B}_{\text{sup}}(w)$
$\mathcal{A} \preceq \mathcal{B}$	no winning strategy for Challenger in the simulation game for \mathcal{A} and \mathcal{B}
$\mathcal{A} \trianglelefteq \mathcal{B}$	$\forall f \in R(\mathcal{A}) : \exists g \in R(\mathcal{B}) : \forall w \in \Sigma^\omega : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \triangleleft \mathcal{B}$	$\exists g \in R(\mathcal{B}) : \forall f \in R(\mathcal{A}) : \forall w \in \Sigma^\omega : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \sqsubseteq \mathcal{B}$	$\forall w \in \Sigma^\omega : \forall f \in R(\mathcal{A}) : \exists g \in R(\mathcal{B}) : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \blacksquare \mathcal{B}$	$\forall w \in \Sigma^\omega : \exists g \in R(\mathcal{B}) : \forall f \in R(\mathcal{A}) : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \trianglelefteq_x \mathcal{B}$	$\forall f \in R_1(\mathcal{A}, \mathcal{B}) : \exists g \in R_2(\mathcal{A}, \mathcal{B}) : \forall w \in \Sigma^\omega$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$
$\mathcal{A} \triangleleft_x \mathcal{B}$	$\exists g \in R_2(\mathcal{A}, \mathcal{B}) : \forall f \in R_1(\mathcal{A}, \mathcal{B}) : \forall w \in \Sigma^\omega$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$
$\mathcal{A} \sqsubseteq_x \mathcal{B}$	$\forall w \in \Sigma^\omega : \forall f \in R_1(\mathcal{A}, \mathcal{B}) : \exists g \in R_2(\mathcal{A}, \mathcal{B})$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$
$\mathcal{A} \blacksquare_x \mathcal{B}$	$\forall w \in \Sigma^\omega : \exists g \in R_2(\mathcal{A}, \mathcal{B}) : \forall f \in R_1(\mathcal{A}, \mathcal{B})$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$



■ **Figure 3** Left: The definitions of inclusion (denoted \subseteq), simulation (denoted \preceq), and the resolver relations we study in Section 3 including strategic dominance (denoted \trianglelefteq). Right: The implications between these relations as proved in Propositions 3.3–3.7 and Corollary 3.8.

We proceed to show the implications posed in Figure 3.

► **Proposition 3.5.** *Let $\nu_1, \nu_2 \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. For all ν_1 -automata \mathcal{A} and all ν_2 -automata \mathcal{B} , the following statements hold. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

1. $\mathcal{A} \trianglelefteq \mathcal{B} \Rightarrow \mathcal{A} \subseteq \mathcal{B}$
2. $\mathcal{A} \preceq \mathcal{B} \Rightarrow \mathcal{A} \trianglelefteq \mathcal{B}$
3. $\mathcal{A} \triangleleft_x \mathcal{B} \Rightarrow \mathcal{A} \preceq \mathcal{B}$
4. $\mathcal{A} \triangleleft \mathcal{B} \Rightarrow \mathcal{A} \triangleleft_x \mathcal{B}$

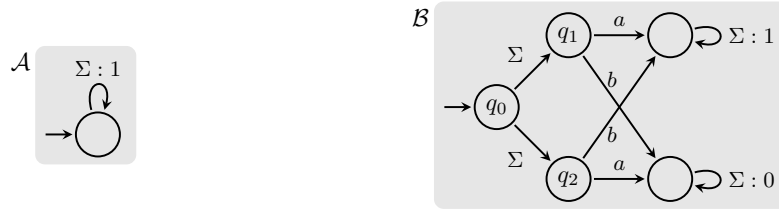
The implications given in Figure 3 (and proved in Propositions 3.3–3.5) also hold when the problem statements are restricted to only finite-memory resolvers or positional resolvers.

► **Proposition 3.6.** *Let $\nu_1, \nu_2 \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. For all ν_1 -automata \mathcal{A} and all ν_2 -automata \mathcal{B} and each $r \in \{\text{fin}, \text{pos}\}$ the following statements hold. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

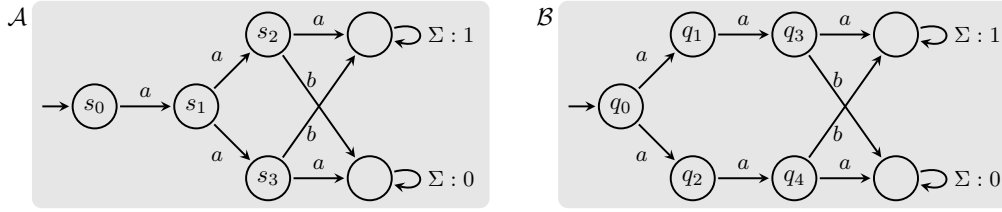
1. $\mathcal{A} \sqsubseteq^r \mathcal{B} \Leftrightarrow \mathcal{A} \blacksquare^r \mathcal{B} \Leftrightarrow \mathcal{A} \sqsubseteq_x^r \mathcal{B} \Leftrightarrow \mathcal{A} \blacksquare_x^r \mathcal{B} \Leftrightarrow \mathcal{A} \subseteq^r \mathcal{B}$
2. $\mathcal{A} \trianglelefteq_x^r \mathcal{B} \Leftrightarrow \mathcal{A} \preceq^r \mathcal{B}$
3. $\mathcal{A} \trianglelefteq^r \mathcal{B} \Rightarrow \mathcal{A} \subseteq^r \mathcal{B}$
4. $\mathcal{A} \preceq^r \mathcal{B} \Rightarrow \mathcal{A} \trianglelefteq^r \mathcal{B}$
5. $\mathcal{A} \triangleleft_x^r \mathcal{B} \Rightarrow \mathcal{A} \preceq^r \mathcal{B}$
6. $\mathcal{A} \triangleleft^r \mathcal{B} \Rightarrow \mathcal{A} \triangleleft_x^r \mathcal{B}$

3.2 Separating Examples for Resolver Relations

In this part, we provide separating examples for the implications we proved above, establishing a hierarchy of relations given in Figure 3. The counter-examples we used in the proofs below are displayed in Figures 4 and 5.



■ **Figure 4** Two automata \mathcal{A} and \mathcal{B} such that $\mathcal{A} \subseteq^{\text{pos}} \mathcal{B}$ but $\mathcal{A} \not\leq^{\text{pos}} \mathcal{B}$. Note that \mathcal{A} and \mathcal{B} can be Inf-, Sup-, LimSup-, or LimInf-automata as well as safety, reachability, Büchi, or coBüchi automata. The exact values of the omitted weights depend on the considered value function. For example, we can take as weight 1 for Inf and 0 for Sup while both choices work for LimSup and LimInf.



■ **Figure 5** Two automata \mathcal{A} and \mathcal{B} such that (i) $\mathcal{A} \leq^{\text{pos}} \mathcal{B}$ but $\mathcal{A} \not\leq^{\text{pos}} \mathcal{B}$, (ii) $\mathcal{A} \leq^{\text{pos}} \mathcal{A}$ but $\mathcal{A} \not\leq_{\times}^{\text{pos}} \mathcal{A}$, and (iii) $\mathcal{B} \leq_{\times}^{\text{pos}} \mathcal{A}$ but $\mathcal{B} \not\leq^{\text{pos}} \mathcal{A}$. The transitions that are not shown lead to a sink state with a self-loop on every letter with weight 0. Note that \mathcal{A} and \mathcal{B} can be Inf-, Sup-, LimSup-, or LimInf-automata as well as safety, reachability, Büchi, or coBüchi automata. The exact values of the omitted weights depend on the considered value function. For example, we can take as weight 1 for Inf and 0 for Sup while both choices work for LimSup and LimInf.

► **Proposition 3.7.** *Let $\nu_1, \nu_2 \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. For each statement below, there exist a ν_1 -automaton \mathcal{A} and a ν_2 -automaton \mathcal{B} to satisfy it. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

1. $\mathcal{A} \subseteq^{\text{pos}} \mathcal{B} \wedge \mathcal{A} \not\leq^{\text{pos}} \mathcal{B}$
2. $\mathcal{A} \leq^{\text{pos}} \mathcal{B} \wedge \mathcal{A} \not\leq^{\text{pos}} \mathcal{B}$
3. $\mathcal{A} \leq^{\text{pos}} \mathcal{B} \wedge \mathcal{A} \not\leq_{\times}^{\text{pos}} \mathcal{B}$
4. $\mathcal{A} \leq_{\times}^{\text{pos}} \mathcal{B} \wedge \mathcal{A} \not\leq^{\text{pos}} \mathcal{B}$

Since $R^{\text{pos}}(\mathcal{A}) = R^{\text{fin}}(\mathcal{A}) = R(\mathcal{A})$ for each automaton \mathcal{A} we consider in this section (see Figures 4 and 5), the statements above also hold for the finite-memory and the general cases.

► **Corollary 3.8.** *Let $\nu_1, \nu_2 \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. For each statement below, there exist a ν_1 -automaton \mathcal{A} and a ν_2 -automaton \mathcal{B} to satisfy it. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

1. $\mathcal{A} \subseteq^{\text{fin}} \mathcal{B} \wedge \mathcal{A} \not\leq^{\text{fin}} \mathcal{B}$
2. $\mathcal{A} \leq^{\text{fin}} \mathcal{B} \wedge \mathcal{A} \not\leq^{\text{fin}} \mathcal{B}$
3. $\mathcal{A} \leq^{\text{fin}} \mathcal{B} \wedge \mathcal{A} \not\leq_{\times}^{\text{fin}} \mathcal{B}$
4. $\mathcal{A} \leq_{\times}^{\text{fin}} \mathcal{B} \wedge \mathcal{A} \not\leq^{\text{fin}} \mathcal{B}$
5. $\mathcal{A} \subseteq \mathcal{B} \wedge \mathcal{A} \not\leq \mathcal{B}$
6. $\mathcal{A} \leq \mathcal{B} \wedge \mathcal{A} \not\leq \mathcal{B}$
7. $\mathcal{A} \leq \mathcal{B} \wedge \mathcal{A} \not\leq_{\times} \mathcal{B}$
8. $\mathcal{A} \leq_{\times} \mathcal{B} \wedge \mathcal{A} \not\leq \mathcal{B}$

► **Remark 3.9.** The relations \leq and \leq_{\times} are not reflexive (and thus not a preorder). For the automaton \mathcal{A} given in Figure 5, we have $\mathcal{A} \not\leq_{\times}^{\text{pos}} \mathcal{A}$ as shown in the proof of Item (3) of Proposition 3.7, and thus $\mathcal{A} \not\leq \mathcal{A}$ since all of its resolvers are positional. Then, by Item (4) of Proposition 3.5, we also have $\mathcal{A} \not\leq \mathcal{A}$.

4 Resolver Logic

We dedicate this section to describing resolver logic, which intuitively extends Presburger arithmetic by introducing variables evaluated by automata parameterized by quantified words and resolvers. We formally define resolver logic and show that the model-checking of a resolver logic formula is decidable.

► **Definition 4.1** (resolver logic). *Let $QA = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ be a finite set of automata over the same alphabet Σ . For all $k \in \{1, \dots, n\}$, let F_k be a set of resolver variables ranging over $R(\mathcal{A}_k)$, and let $W \subseteq \Sigma^\omega$ be a set of word variables ranging over Σ^ω . We define the set $V = \{v_{(x,y)} \mid x \in W, y \in \bigcup_{k=1}^n F_k\}$ of variables ranging over non-negative integers. A resolver logic formula on the automata domain QA is a term generated by the grammar $\Psi ::= \exists x : \Psi \mid \forall x : \Psi \mid \varphi$, where $x \in W \cup \bigcup_{k=1}^n F_k$ and $\varphi \in \exists FO(\mathbb{N}, =, +, 1)$ is an existential Presburger formula whose set of free variables is V .*

We write $|\Psi|$ to denote the size of Ψ defined as $|\Psi| = n + m + |\varphi|$ where $n = |QA|$ is the cardinality of the automata domain, $m = |W| + \sum_{i=1}^n |F_i|$ is the number of word and resolver variables in Ψ , and $|\varphi|$ is the number of (existential) quantifiers in φ . Note that the strategic dominance and the other relations defined in Section 3 are examples of resolver logic formulas. An assignment α maps variables of W to words in Σ^ω , variables of F_k to resolvers in $R(\mathcal{A}_k)$ for all $k \in \{1, \dots, n\}$, and variables of $v \in V$ to values in \mathbb{N} . In particular, $\alpha(v_{(x,x')}) = \mathcal{A}_k^f(w)$ where $x \in W$ and $x' \in F_k$ such that $\alpha(x) = w \in \Sigma^\omega$ and $\alpha(x') = f \in R(\mathcal{A}_k)$. The semantics of Ψ is defined as follows.

$$\begin{aligned} (QA, \alpha) \models \varphi &\text{ iff } \varphi[\forall v \in V : v \leftarrow \alpha(v)] \text{ holds} \\ (QA, \alpha) \models \exists x \in W : \Psi &\text{ iff for some } w \in \Sigma^\omega \text{ we have } \alpha[x \leftarrow w] \models \Psi \\ (QA, \alpha) \models \forall x \in W : \Psi &\text{ iff for all } w \in \Sigma^\omega \text{ we have } \alpha[x \leftarrow w] \models \Psi \\ (QA, \alpha) \models \exists y \in F_k : \Psi &\text{ iff for some } f \in R(\mathcal{A}_k) \text{ we have } \alpha[y \leftarrow f] \models \Psi \\ (QA, \alpha) \models \forall y \in F_k : \Psi &\text{ iff for all } f \in R(\mathcal{A}_k) \text{ we have } \alpha[y \leftarrow f] \models \Psi \end{aligned}$$

► **Theorem 4.2.** *The model-checking of a given resolver logic formula Ψ is decidable. When $|\Psi|$ is fixed, model-checking is in $d\text{-EXPTIME}$ if there are $d > 0$ quantifier alternations and $P\text{TIME}$ if there is no quantifier alternation.*

The decision procedure relies on the following: (1) Each variable assignment can be represented by a single tree. (2) Each resolver logic formula admits a parity tree automata that accepts all tree-encoded assignments that satisfy its inner Presburger formula. (3) The model-checking of a given resolver formula can be decided based on nested complementations and projections over the above parity tree automaton. Below we provide an overview of the corresponding constructions (the full proof is deferred to the appendix due to space constraints). We start by defining trees and describing to use them to encode assignments.

Trees. Let Σ be an alphabet for the structure of trees. We view the set Σ^* of finite words as the domain of an infinite $|\Sigma|$ -ary tree. The root is the empty word ε , and for a node $u \in \Sigma^*$ together with some letter $\sigma \in \Sigma$ we call $u\sigma$ the σ -successor of u . Let Λ be an alphabet for the labeling of nodes. An infinite Λ -labeled Σ -structured tree is a function $t: \Sigma^* \rightarrow \Lambda$. We denote by Λ_Σ^ω the set of all such trees. In a tree $t \in \Lambda_\Sigma^\omega$, the word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ induces the branch $t(w)$ defined as the infinite sequence $t(\varepsilon)\sigma_0 t(\sigma_0)\sigma_1 t(\sigma_0\sigma_1) \cdots \in (\Lambda \times \Sigma)^\omega$.

Let $\text{QA} = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ be a set of automata over the same alphabet Σ . Consider the resolver logic formula over QA of the form $\Psi = \nabla_1 x_1 : \dots : \nabla_m x_m : \varphi$, where $\nabla_i \in \{\exists, \forall\}$. The decision procedure encodes assignments for resolver and word variables of Ψ into Λ -labeled Σ -structured trees, where $\Lambda = (\{0, 1\} \cup \bigcup_{i=1}^n Q_n)^m$. All dimensions of the tree that correspond to a word must have exactly one branch labeled by 1 (which encodes the word), and all other nodes are labeled by 0. Formally, the assignment α_t encoded by a tree $t \in \Lambda_\Sigma^\omega$ maps the word variable $x_j \in W$ to the unique word $\alpha_t(x_j) = \sigma_1 \sigma_2 \dots \in \Sigma^\omega$ for which the j th dimension of the branch $t(\alpha_t(x_j))$ is $t_j(\alpha_t(x_j)) = 1\sigma_1 \sigma_2 \dots \in (\{1\} \times \Sigma)^\omega$. All dimensions of the tree that correspond to a resolver of \mathcal{A}_k must respect its transition relation. Formally, the assignment α_t encoded by a tree $t \in \Lambda_\Sigma^\omega$ maps the resolver variable $x_i \in F_k$ to the unique resolver $\alpha_t(x_i) \in R(\mathcal{A}_k)$ defined by $\alpha_t(x_i)(\pi_k, \sigma) = t_i(u\sigma)$ where π_k is the finite run of $\mathcal{A}_k^{\alpha_t(x_i)}$ over $u \in \Sigma^*$. Consequently, for all $x_i \in F_k$ and all $x_j \in W$, the assignment α_t encoded by a tree $t \in \Lambda_\Sigma^\omega$ maps the variable $v_{(x_i, x_j)} \in V$ to the unique non-negative integer $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$. Next, we describe the parity tree automaton constructed in the decision procedure.

Parity Tree Automata. A (nondeterministic) parity tree automaton \mathcal{T} over Λ_Σ^ω is a tuple $(\Lambda, \Sigma, Q, I, \Delta, \theta)$ where Λ is a finite labeling alphabet, Σ is a finite structure alphabet, Q is a finite set of states, $I \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Lambda \times (\Sigma \rightarrow Q)$ is a transition relation, and $\theta: Q \rightarrow \mathbb{N}$ is the priority function. Note that the arity of the trees is $|\Sigma|$ and is statically encoded in the transition relation. A run of \mathcal{T} over $t \in \Lambda_\Sigma^\omega$ is a Q -labeled Σ -structured tree $\pi \in Q_\Sigma^\omega$ such that $\pi(\varepsilon) \in I$ and for each $u \in \Sigma^*$ we have $(\pi(u), t(u), \sigma \mapsto \pi(u\sigma)) \in \Delta$. The set of runs of \mathcal{T} over t is denoted $\Pi_t(\mathcal{T})$. A run π is accepting if, for all $w \in \Sigma^\omega$, the maximal priority that appears infinitely often along the branch $t(w)$, namely $\limsup_{i \rightarrow \infty} \theta(\pi(\sigma_0 \dots \sigma_i))$, is even. The language of \mathcal{T} is $T(\mathcal{T}) = \{t \in \Lambda_\Sigma^\omega \mid \pi \in \Pi_t(\mathcal{T}), \limsup_{i \rightarrow \infty} \theta(\pi(\sigma_0 \dots \sigma_i)) \equiv 0 \pmod{2}\}$, i.e., the set of all trees that admit an accepting run.

Parity tree automata are expressive enough to recognize the language of tree-encoded assignments for a given resolver logic formula. We describe an automaton with three computational phases. In first phase, the automaton guesses its initial state. All states hold a vector \vec{z} of m^2 weights appearing in $\mathcal{A}_1, \dots, \mathcal{A}_n$ and satisfying φ , i.e., such that $\varphi[\forall v_{(x, x')} \in V : v_{(x, x')} \leftarrow \vec{z}[x][x']]$ is true. Such a vector is guessed at the root of the run tree and carried in all nodes thanks to the states. Since there are finitely many weights and free variables in φ , there are also finitely many vectors \vec{z} . In the second phase, the automaton “waits” for finitely many transitions. This is important for LimInf and LimSup automata, because the run of $\mathcal{A}_k^{\alpha_t(x_i)}$ over $\alpha_t(x_j)$ may visit finitely many times some weights independent of the long-run value. In the third phase, the automaton checks whether the guessed vector \vec{z} is coherent with the tree-encoded assignment that it reads. Given a run π over the tree $t \in \Lambda_\Sigma^\omega$ that carries the vector \vec{z} , for all word variable $x_j \in W$ and all resolver variable $x_i \in F_k$, the value $\vec{z}[x_i][x_j]$ is a coherent assignment for the variable $v_{(x_i, x_j)} \in V$ when $\vec{z}[x_i][x_j] = \mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$. Consider the assignment α_t given as an input tree $t \in \Lambda_\Sigma^\omega$. The run produced by the resolver $\alpha_t(x_i) \in R(\mathcal{A}_k)$ over the word $\alpha_t(x_j) \in \Sigma^\omega$ corresponds, by construction, to the branch $t_i(\alpha_t(x_j))$. To check $\vec{z}[x_i][x_j] = \mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$ at runtime, the automaton ensures that: (1) the weight $\vec{z}[x_i][x_j]$ is visited infinitely often along the run $t_i(\alpha_t(x_j))$, and (2) it is never dismissed by another weight (e.g., for LimInf value function, the guessed weight should be the smallest visited after the waiting phase). Hence, the accepting condition of the automaton is such that, for all $x_j \in W$ and all $x_i \in F_k$, if the automaton accepts $t \in \Lambda_\Sigma^\omega$ then $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j)) = \vec{z}[x_i][x_j]$. Next, we describe how the quantifiers of a resolver logic formula are handled base on the automaton corresponding to its inner Presburger formula.

Handling Quantifiers. To decide the model-checking of a resolver logic formula of fixed size, we first construct a parity tree automata as presented above. Its size is at most $\mathcal{O}(\max_{1 \leq i \leq n} |\mathcal{A}_i|)^{\mathcal{O}(m^2+n)}$. Since, the satisfiability of an existential Presburger formula with a fixed number of quantifiers is in PTIME [27], the automaton can be constructed in polynomial time when $|\Psi|$ is fixed (i.e., n , m and $|\varphi|$ are fixed). Then, we release the existentially quantified variables through projections, i.e., leaving the automaton a non-deterministic choice while relaxing a dimension of the input tree. Universal quantifiers $\forall x : \Psi'$ are treated as $\neg \exists x : \neg \Psi'$, where each negation \neg requires the complementation of the current tree automaton, and then induce an exponential blow up of the computation time [24]. Ultimately, we obtain a tree automaton that does not read labels and the model-checking of the resolver logic formula reduces to its language non-emptiness. It is worth emphasizing that, when a universal quantifier appears at the edge of the quantifier sequence, some complementations can be avoided (e.g., when all quantifiers are universal). When the innermost quantifier is universal, the automaton is constructed over $\neg \varphi$ instead of φ . When the outermost quantifier is universal, we leverage the parity acceptance condition of the tree automata to perform a final complementation in PTIME. Formally, we increase all state priority by 1 and we check the language emptiness instead of non-emptiness.

► **Remark 4.3.** Resolver logic, as presented above, quantifies over non-partial resolvers (called *full* here to improve clarity). We can extend it to handle partial resolvers over product automata. The key observation is that, as long as a collection of partial resolvers is conclusive, they collectively define a full resolver. Moreover, partial resolvers over the components of product automata are conclusive by definition (see Remark 2.7). Hence, the parity tree automaton in the proof of Theorem 4.2 is constructed similarly. Some modifications are necessary to reason about the full resolver defined by the conclusive collection of partial resolvers, but the size of the parity tree automaton constructed above and the overall complexity will not change. This is because the partial resolvers are defined over a product of automata, which is already taken into account for full resolvers in the construction above.

To conclude, let us note that our construction allows checking inclusion or simulation at a high cost. For example, although checking the inclusion of two LimSup-automata is PSPACE-complete, using an equivalent formulation from Proposition 3.3, we obtain a 2-EXPTIME algorithm using the construction presented in this section. Similarly, while checking simulation for LimSup-automata can be done in $\text{NP} \cap \text{co-NP}$, we obtain a 3-EXPTIME algorithm using Proposition 3.4.

5 Applications of Resolver Logic

In this section, we explore various applications of resolver logic, highlighting its versatility in addressing problems in automata theory and system verification. We begin by presenting how resolver logic can be used for checking strategic dominance and other relations we studied in Section 3. Next, we examine its role in checking the bottom value of automata, which is a crucial problem for deciding co-safety and co-liveness of automata and was left open in [5]. Then, we explore its application for checking history-determinism of automata, and finally discuss its relevance in checking hyperproperty inclusion. As in the previous sections, we note that the results of this section also hold when we only consider boolean safety, reachability, Büchi, and coBüchi automata.

5.1 Checking Strategic Dominance and Other Resolver-Based Relations

Let \mathcal{A} and \mathcal{B} be two automata and recall that \mathcal{B} strategically dominates \mathcal{A} , denoted $\mathcal{A} \leq \mathcal{B}$, iff for all resolvers $f \in R(\mathcal{A})$ there exists a resolver $g \in R(\mathcal{B})$ such that $\mathcal{A}^f(w) \leq \mathcal{B}^g(w)$ for all words $w \in \Sigma^\omega$. Formulating this condition as a resolver logic formula, we obtain a 2-EXPTIME algorithm for checking strategic dominance thanks to Theorem 4.2.

► **Corollary 5.1.** *Let $\nu_1, \nu_2 \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. For all ν_1 -automata \mathcal{A} and all ν_2 -automata \mathcal{B} , checking whether $\mathcal{A} \leq \mathcal{B}$ can be done in 2-EXPTIME.*

Note that other relations introduced in Section 3 can be also checked similarly.

5.2 Checking the Bottom Value of Automata

Safety and liveness [1, 2], as well as co-safety and co-liveness, are fundamental concepts in specification of system properties and their verification. These concepts have been recently extended to quantitative properties [21], and safety and liveness have been studied in the context of quantitative automata [5]. Note that quantitative automata resolve nondeterminism by sup, i.e., given an automaton \mathcal{A} and a word w , we have $\mathcal{A}(w) = \mathcal{A}_{\text{sup}}(w)$.

For deciding the safety or liveness of a given automaton \mathcal{A} , computing its *top value*, namely the value of $\top_{\mathcal{A}} = \sup_{w \in \Sigma^\omega} \sup_{f \in R(\mathcal{A})} \mathcal{A}^f(w)$, is shown to be a central step. The PTIME algorithm provided in [5] is used as a subroutine for computing the safety closure of a given automaton \mathcal{A} , which is used for checking both safety and liveness of \mathcal{A} . In particular, \mathcal{A} is safe iff the safety closure of \mathcal{A} maps every word w to the same value as \mathcal{A} , and \mathcal{A} is live iff the safety closure of \mathcal{A} maps every word w to $\top_{\mathcal{A}}$.

Given an automaton \mathcal{A} , we can solve the top-value problem by simply iterating over its weights in decreasing order and checking for each weight k whether there exist $f \in R(\mathcal{A})$ and $w \in \Sigma^\omega$ with $\mathcal{A}^f(w) \geq k$. The largest k for which this holds is the top value of \mathcal{A} . Note that thanks to Theorem 4.2 we can achieve this in PTIME as it is only an existential formula, which gives us a new algorithm for this problem.

However, the problems of deciding the co-safety and co-liveness of automata were left open. For these, one needs to compute *bottom value* of a given automaton \mathcal{A} , namely $\perp_{\mathcal{A}} = \inf_{w \in \Sigma^\omega} \sup_{f \in R(\mathcal{A})} \mathcal{A}^f(w)$. Similarly as above, using the computation of the bottom value of \mathcal{A} as a subroutine, we can decide its co-safety and co-liveness: \mathcal{A} is co-safe iff the co-safety closure of \mathcal{A} maps every word w to the same value as \mathcal{A} , and \mathcal{A} is co-live iff the co-safety closure of \mathcal{A} maps every word w to $\perp_{\mathcal{A}}$.

For the classes of automata we consider, we can compute the bottom value in PSPACE by repeated universality checks over its finite set of weights: the largest weight k for which the automaton is universal is its bottom value. We remark that the bottom value of limit-average automata is uncomputable since their universality is undecidable [16, 10].

Together with Theorem 4.2, the theorem below provides us with a 2-EXPTIME algorithm for computing the bottom value of Inf-, Sup-, LimInf-, and LimSup-automata.

► **Theorem 5.2.** *Let $\nu \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$. Let \mathcal{A} be a ν -automaton and x be an integer. Then, the bottom value of \mathcal{A} is x iff $\exists w_1 \in \Sigma^\omega : \exists f_1 \in R(\mathcal{A}) : \forall w_2 \in \Sigma^\omega : \forall f_2 \in R(\mathcal{A}) : \exists f_3 \in R(\mathcal{A}) : \mathcal{A}^{f_1}(w_1) = x \wedge \mathcal{A}^{f_1}(w_1) \geq \mathcal{A}^{f_2}(w_1) \wedge \mathcal{A}^{f_1}(w_1) \leq \mathcal{A}^{f_3}(w_2)$. Moreover, given \mathcal{A} and x , this can be checked in 2-EXPTIME.*

5.3 Checking History-Determinism of Automata

History-determinism [22, 15] lies between determinism and nondeterminism. Intuitively, an automaton is history-deterministic if there exists a way of resolving its nondeterminism based on the current execution prefix (i.e., only the past) while ensuring that the value of the

resulting run equals the value assigned to the word by resolving its nondeterminism by sup. Although the concept of history-determinism first appeared as “good-for-gameness” in [22], following the distinction made in [6], we use the definition of history-determinism in [15].

► **Definition 5.3** (history-determinism [15]). *Let $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ be an automaton. Then, \mathcal{A} is history-deterministic iff Player-2 wins the letter game defined below.*

The letter game on \mathcal{A} is played as follows: The game begins on the initial state $q_0 = s$. At each turn $i \geq 0$ that starts in a state q_i , Player-1 first chooses a letter $\sigma_i \in \Sigma$, then Player-2 chooses a transition $d_i = (q_i, \sigma_i, q_{i+1}) \in \Delta$, and the game proceeds to state q_{i+1} . The corresponding infinite play is an infinite run π over the word $w = \sigma_0\sigma_1\dots$, and Player-2 wins the game iff $\mathcal{A}_{\text{sup}}(w) \leq \nu(\mu(\pi))$.

► **Remark 5.4.** An automaton \mathcal{A} is history-deterministic iff there exists a resolver $f \in R(\mathcal{A})$ such that $\mathcal{A}_{\text{sup}}(w) \leq \mathcal{A}^f(w)$ for all $w \in \Sigma^\omega$. One can verify that the resolver f is exactly the winning strategy for Player-2 in the letter game on \mathcal{A} .

History-deterministic automata offer a balance between deterministic and nondeterministic counterparts, with notable advantages. For instance, history-deterministic LimInf-automata are exponentially more concise than deterministic ones [23], and history-deterministic push-down automata exhibit both increased expressiveness and at least exponential succinctness compared to their deterministic counterparts [19]. Further exploration is detailed in [8, 7].

In [7], algorithms are presented to determine whether an automaton is history-deterministic. The approach involves solving a token game that characterizes history-determinism for the given automata type. The procedure is in PTIME for Inf- and Sup-automata, quasipolynomial time for LimSup, and EXPTIME for LimInf. Combined with Theorem 4.2, the theorem below presents a new EXPTIME algorithm for checking history-determinism across all these automata types, providing competitive complexity with [7] for LimInf-automata.

► **Theorem 5.5.** *Let $\nu \in \{\text{Inf}, \text{Sup}, \text{LimSup}, \text{LimInf}\}$ and \mathcal{A} be a ν -automaton. Then, \mathcal{A} is history-deterministic iff $\mathcal{A} \triangleleft \mathcal{A}$. Moreover, given \mathcal{A} , this can be checked in EXPTIME.*

5.4 Checking Hyperproperty Inclusion

We have focused on trace properties – functions mapping words to values, either 0 or 1 in the boolean setting. While adept at representing temporal event orderings, trace properties lack the capacity to capture dependencies among multiple system executions, such as noninterference in security policies [18] or fairness conditions for learning-based systems [28].

This limitation is addressed by hyperproperties [14]. Unlike trace properties, hyperproperties encompass global characteristics applicable to sets of traces. This enables the specification of intricate relationships and constraints beyond temporal sequencing. Formally, while a trace property is a set of traces, a hyperproperty is a set of trace properties.

In this subsection, we use nondeterministic automata as a specification language for hyperproperties. A deterministic automaton defines a trace property where each word has a single run, yielding a unique value. In contrast, a nondeterministic automaton specifies a trace property only when equipped with a resolver, representing a function from its resolvers to trace properties. Formally, a nondeterministic automaton \mathcal{A} specifies the hyperproperty $H_{\mathcal{A}} = \{\mathcal{A}^f \mid f \in R(\mathcal{A})\}$. An illustrative example is presented in Figure 6 and Proposition 5.6.

► **Proposition 5.6.** *The nondeterministic automata \mathcal{A} and \mathcal{B} in Figure 6 respectively specify the hyperproperties $SP = \{P \subseteq \Sigma^\omega \mid P \text{ is safe}\}$ and $CP = \{P \subseteq \Sigma^\omega \mid P \text{ is co-safe}\}$.*



■ **Figure 6** Two nondeterministic automata \mathcal{A} and \mathcal{B} over a finite alphabet Σ that respectively specify the hyperproperties $\text{SP} = \{P \subseteq \Sigma^\omega \mid P \text{ is safe}\}$ and $\text{CP} = \{P \subseteq \Sigma^\omega \mid P \text{ is co-safe}\}$.

HyperLTL [13] extends linear temporal logic (LTL) only with quantification over traces, and therefore cannot express the hyperproperty SP specifying the set of all safety trace properties. However, using HyperLTL over the alphabet $\Sigma = \{i, s, o, x\}$, one can express the noninterference between a secret input s and a public output o as follows: $\forall \pi, \pi' : \Box(i_\pi \leftrightarrow i_{\pi'}) \rightarrow \Box(o_\pi \leftrightarrow o_{\pi'})$, i.e., for every pair π, π' of traces, if the positions of the public input i coincide in π and π' , then so do the positions of the public output o . We show below that a simpler variant of this property cannot be specified by nondeterministic automata, separating them as a specification language for hyperproperties from HyperLTL.

► **Proposition 5.7.** *Let $\Sigma = \{a, b, c\}$ and let $\phi = \forall \pi, \pi' : \Box(b_\pi \leftrightarrow b_{\pi'})$ be a HyperLTL formula. Neither $H_1 = \{P \subseteq \Sigma^\omega \mid P \text{ satisfies } \phi\}$ nor $H_2 = \{P \subseteq \Sigma^\omega \mid P \text{ satisfies } \neg\phi\}$ is expressible by an automaton.*

Together with Theorem 4.2, the following theorem gives us a 2-EXPTIME algorithm for checking if the hyperproperty specified by an automaton is included in another one.

► **Theorem 5.8.** *Let \mathcal{A} and \mathcal{B} be two nondeterministic automata respectively denoting the hyperproperties $H_{\mathcal{A}}$ and $H_{\mathcal{B}}$. Then, $H_{\mathcal{A}} \subseteq H_{\mathcal{B}}$ iff $\forall f \in R(\mathcal{A}) : \exists g \in R(\mathcal{B}) : \forall w \in \Sigma^\omega : \mathcal{A}^f(w) = \mathcal{B}^g(w)$. Moreover, given \mathcal{A} and \mathcal{B} , this can be checked in 2-EXPTIME.*

Using \mathcal{A} for a deterministic automaton representing a system and \mathcal{B} for a nondeterministic automaton defining a hyperproperty, we solve the model checking problem by determining if there exists $g \in R(\mathcal{B})$ such that $\mathcal{A}(w) = \mathcal{B}^g(w)$ for all $w \in \Sigma^\omega$. This is solvable in EXPTIME (Theorem 4.2). Notably, for HyperLTL, which is incomparable to nondeterministic automata as a specification language, the complexity is PSPACE-hard in the system's size [13].

6 Conclusion

We introduced a novel perspective on model refinement, termed *strategic dominance*. This view, which falls between trace inclusion and tree inclusion, captures the relationship between two nondeterministic state-transition models by emphasizing the ability of the less precise model to accommodate all deterministic implementations of the more refined one. We formally defined strategic dominance and showed that it can be checked in 2-EXPTIME using *resolver logic* – a decidable extension of Presburger logic we developed in this work. Resolver logic is a powerful tool for reasoning about nondeterministic boolean and quantitative finite automata over infinite words. We provided a model-checking algorithm for resolver logic which, besides the verification of resolver-based refinement relations such as strategic dominance, allows the checking of co-safety, co-liveness, and history-determinism of quantitative automata, and the inclusion of hyperproperties specified by nondeterministic automata. There are some problems we have left open, including the study of resolver logic for other value functions as well as lower bounds for the model-checking problem of resolver logic and its fragments. Future research should also extend resolver logic and its model-checking algorithm to handle the settings of partial information, of multiple agents, and of probabilistic strategies.

References

- 1 Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985. doi:10.1016/0020-0190(85)90056-0.
- 2 Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987. doi:10.1007/BF01782772.
- 3 Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998. doi:10.1007/BFB0055622.
- 4 Ralph-Johan Back and Joakim von Wright. *Refinement Calculus - A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998. doi:10.1007/978-1-4612-1674-2.
- 5 Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Safety and liveness of quantitative automata. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPICs*, pages 17:1–17:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.CONCUR.2023.17.
- 6 Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPICs*, pages 38:1–38:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.FSTTCS.2021.38.
- 7 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative-automata. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:8)2023.
- 8 Udi Boker and Karoliina Lehtinen. When a little nondeterminism goes a long way: An introduction to history-determinism. *ACM SIGLOG News*, 10(1):24–51, 2023. doi:10.1145/3584676.3584682.
- 9 Stephen Brookes. A semantics for concurrent separation logic. *Theor. Comput. Sci.*, 375(1-3):227–270, 2007. doi:10.1016/J.TCS.2006.12.034.
- 10 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2010. doi:10.1007/978-3-642-15375-4_19.
- 11 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010. doi:10.1145/1805950.1805953.
- 12 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010. doi:10.1016/J.IC.2009.07.004.
- 13 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.
- 14 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
- 15 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2009. doi:10.1007/978-3-642-02930-1_12.

- 16 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2010. doi:10.1007/978-3-642-15205-4_22.
- 17 Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, and Mateusz Skomra. Games on graphs. *CoRR*, abs/2305.10546, 2023. doi:10.48550/arXiv.2305.10546.
- 18 Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.
- 19 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. *Log. Methods Comput. Sci.*, 20(1), 2024. doi:10.46298/LMCS-20(1:3)2024.
- 20 Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. *Inf. Comput.*, 173(1):64–81, 2002. doi:10.1006/INCO.2001.3085.
- 21 Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Quantitative safety and liveness. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 349–370. Springer, 2023. doi:10.1007/978-3-031-30829-1_17.
- 22 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006. doi:10.1007/11874683_26.
- 23 Denis Kuperberg and Michal Skrzypczak. On determinisation of good-for-games automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-47666-6_24.
- 24 Christof Löding. Automata on infinite trees. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 265–302. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/AUTOMATA-1/8.
- 25 Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning about strategies. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPICs*, pages 133–144. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICs.FSTTCS.2010.133.
- 26 Peter W. O’Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1-3):271–307, 2007. doi:10.1016/J.TCS.2006.12.035.
- 27 Bruno Scarpellini. Complexity of subcases of presburger arithmetic. *Transactions of the American Mathematical Society*, 284(1):203–218, 1984. URL: <http://www.jstor.org/stable/1999283>.
- 28 Sanjit A. Seshia, Ankush Desai, Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue. Formal specification for deep neural networks. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2018. doi:10.1007/978-3-030-01090-4_2.

- 29 Rob J. van Glabbeek. The linear time - branching time spectrum II. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993. doi:10.1007/3-540-57208-2_6.
- 30 Rob J. van Glabbeek. The linear time - branching time spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland / Elsevier, 2001. doi:10.1016/B978-044482830-9/50019-9.

A Appendix

Proof of Theorem 4.2. We assume that each automata $\mathcal{A}_k = (Q_k, s_k, \Delta_k, \mu_k, \nu_k)$ is either a LimSup-automaton or a LimInf-automaton. This is without loss of generality since Sup-automata and Inf-automata can be converted in PTIME into LimInf-automata [5, Proposition 2.1]. The proof goes as follows. First, we construct in polynomial time a parity tree automaton \mathcal{C} which read an assignment α for Ψ as input, such that its language is empty if and only if $\alpha \models \Psi$. Then we handle the quantifiers of Ψ based on nested complementations and projections applied on \mathcal{C} . Finally, we construct a parity game such that the even player wins iff Ψ is satisfiable over $\mathcal{A}_1, \dots, \mathcal{A}_n$.

Construction of \mathcal{C} . Let Ψ be of the form $\nabla_1 x_1 : \dots : \nabla_m x_m : \varphi$, where $\nabla_i \in \{\exists, \forall\}$. In this construction, we encode assignments for resolver and word variables of Ψ into single Σ -structured trees. The labeling alphabet is defined from the sets Q_1, \dots, Q_n and $\{0, 1\}$ in order to manipulate branches as runs. For all $i \in \{1, \dots, m\}$, we define Λ_i and $\ell_i \in \Lambda_i$ such that if $x_i \in W$ then $\Lambda_i = \{0, 1\}$ and $\ell_i = 1$; otherwise, $x_i \in F_k$ for some $k \in \{1, \dots, n\}$, and so $\Lambda_i = Q_k$ and $\ell_i = s_k$ where s_k is the initial state of \mathcal{A}_k . Let the labeling alphabet be $\Lambda = \Lambda_1 \times \dots \times \Lambda_m$ and let the label of roots be $\ell = \ell_1 \times \dots \times \ell_m$. For all $\lambda \in \Lambda$ and $1 \leq i \leq m$ we write $\lambda[i]$ to denote the dimension of λ corresponding to Λ_i . In the same way, we construct the value domains of the variables of V from the sets of weights of $\mathcal{A}_1, \dots, \mathcal{A}_k$. For all $x_i \in F_k$ and all $x_j \in W$, we define the value domain of the variable $v_{(x_i, x_j)} \in V$ as $Z_{(x_i, x_j)} = \{\mu_k(\delta) \in \mathbb{N} \mid \delta \in \Delta_k\}$. Let $Z = \prod_{k=1}^n \prod_{f \in F_k} \prod_{w \in W} Z_{(f, w)}$ be the set of assignment of the variable of V . For all $z \in Z$, all $x_i \in F_k$ and all $x_j \in W$, we write $z[x_i][x_j]$ to denote the dimension of z corresponding to $Z_{(f, w)}$.

We now construct the parity tree automaton $\mathcal{C} = (\Lambda, \Sigma, Q, I, \Delta, \theta)$. The set of \heartsuit -states is $Q_{\heartsuit} = \{(\heartsuit_{(y_1, y_2)}, z, \lambda) \mid y_1, y_2 \in \{1, \dots, m\}, z \in Z, \lambda \in \Lambda\}$, the set of \spadesuit -states is $Q_{\spadesuit} = \{(\spadesuit_{(y_1, y_2)}, z, \lambda) \mid y_1, y_2 \in \{1, \dots, m\}, z \in Z, \lambda \in \Lambda\}$, and the set of \perp -states is $Q_{\perp} = \{(\perp, z, \lambda) \mid z \in Z, \lambda \in \Lambda\}$. The set of states is $Q = Q_{\perp} \cup Q_{\heartsuit} \cup Q_{\spadesuit}$ and the set of initial states is $I = \{(\perp, z, \ell) \mid z \in Z, \varphi[\forall v_{(x, x')} \in V : v_{(x, x')} \leftarrow z[x][x']]\}$. The priority function $\theta: Q \rightarrow \{1, 2\}$ maps \heartsuit -states to 2 and all the other states to 1. The transition relation Δ is defined as follows.

■ $((\perp, z, \lambda), \lambda, \sigma \mapsto (S_{\sigma}, z, \lambda_{\sigma})) \in \Delta$ where $S_{\sigma} \in \{\perp, \spadesuit_{(1,1)}\}$ iff

$$\bigwedge \left\{ \begin{array}{l} \bigwedge_{j=1}^m \left(x_j \in W \Rightarrow \sum_{\sigma \in \Sigma} \lambda_{\sigma}[j] = \lambda[j] \right) \\ \bigwedge_{i=1}^m \bigwedge_{k=1}^n \bigwedge_{\sigma \in \Sigma} \left(x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_{\sigma}[i]) \in \Delta_k \right) \end{array} \right.$$

For all transitions, \mathcal{C} ensures that the encoding of its assignment for x_1, \dots, x_m is a coherent Λ -labeled Σ -structured tree. Above, the first constraint guarantees that all dimensions encoding a word have exactly one branch labeled by 1 (which encodes the word), and all other nodes are labeled by 0. Formally, each tree $t \in T(\mathcal{C})$ assigns the variable $x_j \in W$ to the unique word $\alpha_t(x_j) = \sigma_1 \sigma_2 \dots \in \Sigma^{\omega}$ for which the j th dimension of the branch $t(\alpha_t(x_j))$ equals $1\sigma_1 1\sigma_2 \dots \in (\{1\} \times \Sigma)^{\omega}$. The second constraint guarantees that all dimensions

encoding a resolver of \mathcal{A}_k respect its transition relation, i.e., a node labeled by λ and its σ -child labeled by λ_σ must encode a transition of \mathcal{A}_k in these dimensions. Formally, each tree $t \in T(\mathcal{C})$ assigns the variable $x_i \in F_k$ to the unique resolver $\alpha_t(x_i) \in R(\mathcal{A}_k)$ defined by $\alpha_t(x_i)(\pi_k, \sigma) = t(u\sigma)$ where π_k is the finite run of $\mathcal{A}_k^{\alpha_t(x_i)}$ over $u \in \Sigma^*$. In particular, for all $t \in T(\mathcal{C})$, all $x_i \in F_k$ and $x_j \in W$, the value $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$ is the correct assignment for the free variable $v_{(x_i, x_j)} \in V$ of φ .

■ $((\spadesuit_{(y_1, y_2)}, z, \lambda), \lambda, \sigma \mapsto (\spadesuit_{(y_1, y_2)}, z, \lambda_\sigma)) \in \Delta$ iff

$$\bigwedge \left\{ \begin{array}{l} \bigwedge_{j=1}^m (x_j \in W \Rightarrow \sum_{\sigma \in \Sigma} \lambda_\sigma[j] = \lambda[j]) \\ \bigwedge_{i=1}^m \bigwedge_{k=1}^n \bigwedge_{\sigma \in \Sigma} (x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k) \\ \bigwedge_{i=1}^m \bigwedge_{k=1}^n \bigwedge_{j=1}^m \bigwedge_{\sigma \in \Sigma} ((x_i \in F_k \wedge \lambda[j] = 1) \Rightarrow h_k(z[x_i][x_j], \mu_k(\lambda[i], \sigma, \lambda_\sigma[i])) = z[x_i][x_j]) \end{array} \right.$$

where $h_k = \max$ if \mathcal{A}_k is a **LimSup**-automaton and $h_k = \min$ if it is a **LimInf**-automaton.

Observe that \perp -states are reachable only from \perp -states and cannot lead to acceptance as their priority is odd. Once a $\spadesuit_{(1,1)}$ -state is reached, \mathcal{C} checks through the rest of the run tree whether z provides a correct assignment for the variable of V . By construction, z is guessed at the root of the run tree and carried in all its nodes. Given a run π of \mathcal{C} over $t \in \Lambda_\Sigma^\omega$ that carries $z \in Z$, for all $x_j \in W$, all $x_i \in F_k$, the value $z[x_i][x_j]$ is a correct assignment for $v_{(x_i, x_j)}$ when $z[x_i][x_j] = \mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$. Intuitively, $v_{(x_i, x_j)}$ requires \mathcal{C} to ensure that the weight $z[x_i][x_j]$ is (\dagger) visited infinitely often, (\ddagger) never dismissed by another weight, and so along the branch induced by the word $\alpha_t(x_j)$. The condition (\dagger) is handled by \mathcal{C} thanks to its acceptance condition that we explain below. Above, the last constraint guarantees the condition (\ddagger) , i.e., assuming that $x_j \in W$, $x_i \in R(\mathcal{A}_k)$ and that the current node belongs to the branch induced by $\alpha_t(x_j)$, if \mathcal{A}_k is a **LimSup**-automaton then the weight of the transition $(\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k$ from the node to its σ -child is at most $z[x_i][x_j]$, otherwise \mathcal{A}_k is a **LimInf**-automaton and the weight of this transition is at least $z[x_i][x_j]$. This constrain appears in all transitions outgoing from the \spadesuit -states and the \heartsuit -states.

■ $((\spadesuit_{(y_1, y_2)}, z, \lambda), \lambda, \sigma \mapsto (S_{\sigma(y_1, y_2)}, z, \lambda_\sigma)) \in \Delta$ where $S_{\sigma(y_1, y_2)} \in \{\heartsuit_{(y_1, y_2)}, \spadesuit_{(y_1, y_2)}\}$ iff

$$\bigwedge \left\{ \begin{array}{l} \bigwedge_{j=1}^m (x_j \in W \Rightarrow \sum_{\sigma \in \Sigma} \lambda_\sigma[j] = \lambda[j]) \\ \bigwedge_{i=1}^m \bigwedge_{k=1}^n \bigwedge_{\sigma \in \Sigma} (x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k) \\ \bigwedge_{i=1}^m \bigwedge_{k=1}^n \bigwedge_{j=1}^m \bigwedge_{\sigma \in \Sigma} ((x_i \in F_k \wedge \lambda[j] = 1) \Rightarrow h_k(z[x_i][x_j], \mu_k(\lambda[i], \sigma, \lambda_\sigma[i])) = z[x_i][x_j]) \\ \bigwedge_{k=1}^n \bigwedge_{\sigma \in \Sigma} ((x_{y_1} \in F_k \wedge \lambda[y_2] = 1 \wedge S_{\sigma(y_1, y_2)} = \heartsuit_{(y_1, y_2)}) \Rightarrow \mu_k(\lambda[y_1], \sigma, \lambda_\sigma[y_1]) = z[x_{y_1}][x_{y_2}]) \end{array} \right.$$

Given a run π of \mathcal{C} over $t \in \Lambda_\Sigma^\omega$ that carries $z \in Z$, assuming that $x_{y_2} \in W$ and $x_{y_1} \in F_k$, the condition (\dagger) asks \mathcal{C} to check whether the guessed value $z[x_{y_1}][x_{y_2}]$ is among the values visited infinitely many times along the branch induced by $\alpha_t(x_{y_2}) \in \Sigma^\omega$. Above, the last constraint guarantees that \mathcal{C} allows to move from a $\spadesuit_{(y_1, y_2)}$ -state to a $\heartsuit_{(y_1, y_2)}$ -state only if either $x_{y_1} \notin R(\mathcal{A}_k)$, or the current node does not belong to the branch induced by $\alpha_t(x_{y_2}) \in \Sigma^\omega$, or the guessed weight $z[x_{y_1}][x_{y_2}]$ is visited in the corresponding dimension. Observe that \heartsuit -states have priority 2 in \mathcal{C} , while \spadesuit -states have priority 1. The condition (\dagger) on $z[x_{y_1}][x_{y_2}]$ holds for all accepting runs because \mathcal{C} ensures that a $\heartsuit_{(y_1, y_2)}$ -state is visited infinitely many times on all branches of its accepting runs, as we explain below.



■ $((\heartsuit_{(y_1, y_2)}, z, \lambda), \lambda, \sigma \mapsto (\spadesuit_{(y'_1, y'_2)}, z, \lambda_\sigma)) \in \Delta$ iff

$$\bigwedge \left\{ \begin{array}{l} \bigwedge_{j=1}^m \left(x_j \in W \Rightarrow \sum_{\sigma \in \Sigma} \lambda_\sigma[j] = \lambda[j] \right) \\ \bigwedge_{i=1}^m \bigwedge_{k=1}^n \bigwedge_{\sigma \in \Sigma} \left(x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k \right) \\ \bigwedge_{i=1}^m \bigwedge_{k=1}^n \bigwedge_{j=1}^m \bigwedge_{\sigma \in \Sigma} \left((x_i \in F_k \wedge \lambda[j] = 1) \Rightarrow h_k(z[x_i][x_j], \mu_k(\lambda[i], \sigma, \lambda_\sigma[i])) = z[x_i][x_j] \right) \\ (y'_1 = y_1 \wedge y'_2 = y_2 + 1) \vee (y'_1 = y_1 + 1 \wedge y_2 = m \wedge y'_2 = 1) \vee (y_1 = y_2 = m \wedge y'_1 = y'_2 = 1) \end{array} \right.$$




We recall that $\mathcal{O}(m^2)$ values are checked by \mathcal{C} through its runs. To ensure that condition (†) holds for all dimensions of z , the transitions of \mathcal{C} enforces to visit cyclically all $\heartsuit_{(y_1, y_2)}$ -states in order to get a run of even priority. Above, the last constraint guarantees that \mathcal{C} allows to leave a $\heartsuit_{(y_1, y_2)}$ -state only toward a \spadesuit -state that regulates the next pair of index. Since there is no transition from a \heartsuit -state to a \heartsuit -state, a run of \mathcal{C} is accepting if and only if all branches visit a $\heartsuit_{(y_1, y_2)}$ -state infinitely often for all $y_1, y_2 \in \{1, \dots, m\}$. As final observation, we point out that, since z carried in all nodes of the run tree, the consistency of (†) and (‡) through branches is guaranteed. Hence, for all $t \in T(\mathcal{C})$, if $x_j \in W$ and $x_i \in F_k$ then the value $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$ equals $z[x_i][x_j]$ thanks to the conditions (†) and (‡).

Construction of the parity game. Note that the size of \mathcal{C} is at most $\mathcal{O}(|\max_{1 \leq i \leq n} |\mathcal{A}_i|)^{\mathcal{O}(m^2+n)}$. In particular, when $|\Psi|$ is fixed (i.e., n, m and $|\varphi|$ are fixed), \mathcal{C} can be constructed in polynomial time since the satisfiability of an existential Presburger formula with a fixed number of quantifiers is in PTIME [27]. To handle the quantifiers of Ψ , we construct a parity tree automaton \mathcal{C}' that do not take inputs. Essentially, \mathcal{C}' is constructed from \mathcal{C} by releasing the existentially quantified variables through projections, i.e., leaving the tree automaton a non-deterministic choice while relaxing a dimension of the input tree. Universal quantifiers $\forall x : \Psi'$ are treated as $\neg \exists x : \neg \Psi'$, where each negation \neg requires the computation of the complement of the current tree automaton, and then induce an exponential blow up of the computation time [24]. The parity game is constructed in PTIME from a tree automaton \mathcal{C}' . The game proceeds with the even player first choosing a transition in the tree automaton, and then the odd player choosing a subtree. The even player wins iff the language of the tree automaton is not empty. Naturally, a resolver logic formula with only existential quantifiers do not require tree automata complementations. However, with a naive approach, a formula with only universal quantifiers may requires two complementations while none are necessary. This is because if the innermost quantifier is universal then the first complementation can be avoided by using $\neg \varphi$ instead of φ to construct \mathcal{C} . Additionally, if the outermost quantifier is universal then the last complementation can be avoided by constructing a parity game that is winning for the player with even objective if and only if the current tree automaton is empty. This is doable in PTIME as before, but the players are swapped and the priority are increased by one. ◀

Around Classical and Intuitionistic Linear Processes

Juan C. Jaramillo  

University of Groningen, The Netherlands

Dan Frumin   

University of Groningen, The Netherlands

Jorge A. Pérez   

University of Groningen, The Netherlands

Abstract

Curry-Howard correspondences between Linear Logic (LL) and session types provide a firm foundation for concurrent processes. As the correspondences hold for intuitionistic and classical versions of LL (ILL and CLL), we obtain two different families of type systems for concurrency. An open question remains: *how do these two families exactly relate to each other?* Based upon a translation from CLL to ILL due to Laurent, we provide two complementary answers, in the form of full abstraction results based on a typed observational equivalence due to Atkey. Our results elucidate hitherto missing formal links between seemingly related yet different type systems for concurrency.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Type theory; Theory of computation → Process calculi

Keywords and phrases Process calculi, session types, linear logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.30

Related Version *Full Version:* <http://arxiv.org/abs/2407.06391>

Funding *Juan C. Jaramillo:* Ministry of Science of Colombia (Minciencias).

Jorge A. Pérez: Support of the Dutch Research Council (NWO) under project No.016.Vidi.189.046 (Unifying Correctness for Communicating Software) is gratefully acknowledged.

Acknowledgements We are most grateful to Bas van den Heuvel for initial discussions on the topic of this paper. We also thank the anonymous reviewers for their helpful suggestions.

1 Introduction

We address an open question on the logical foundations of concurrency, as resulting from Curry-Howard correspondences between linear logic (LL) and session types. These correspondences, often referred to as “*propositions-as-sessions*”, connect LL propositions and session types, proofs in LL and π -calculus processes, as well as cut-elimination in LL and process synchronization. The result is type systems that elegantly ensure important forms of communication correctness for processes. The correspondence was discovered by Caires and Pfenning, who relied on an *intuitionistic* presentation of LL (ILL) [8]; Wadler later presented it using *classical* LL (CLL) [24]. These two works triggered the emergence of multiple type systems for concurrency with firm logical foundations, based on (variants of) ILL (e.g., [21, 17, 3, 7]) and CLL (e.g., [6, 12, 15, 14, 19]). While key differences between these two families of type systems, intuitionistic and classical, have been observed [22], in this paper we ask: can we formally relate them from the standpoint of (typed) process calculi?

From a logical standpoint, the mere existence of two different families of type systems may seem surprising – after all, the relationship between ILL and CLL is well understood [20, 11, 16]. Laurent has given a thorough account of these relationships, including a translation from CLL to ILL [16]. A central insight in our work is the following: while translations from



© Juan C. Jaramillo, Dan Frumin, and Jorge A. Pérez;
licensed under Creative Commons License CC-BY 4.0

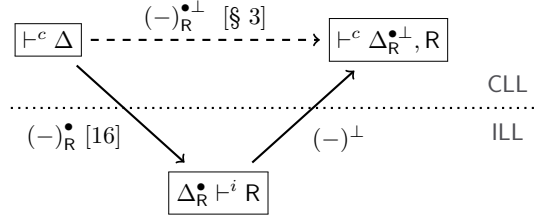
35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 30; pp. 30:1–30:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Translations between CLL and ILL. In this paper, we shall fix $R = \mathbf{1}$.

CLL to ILL are useful, they alone do not entail formal results for typed processes, and a satisfactory answer from the “propositions-as-sessions” perspective must include process calculi considerations.

Let us elaborate. Given some context Δ and a formula A , let us write $\vdash^c \Delta$ and $\Delta \vdash^i A$ to denote sequents in CLL and ILL, respectively. Under the concurrent interpretation induced by “propositions-as-sessions”, these sequents are annotated as $P \vdash^c \Delta$ and $\Delta \vdash^i P :: x : A$, respectively, where P is a process, x is a name, and Δ is now a finite collection of assignments $x_1 : A_1, \dots, x_n : A_n$. An assignment specifies a name’s intended session protocol. This way, e.g., “ $x : A \otimes B$ ” (resp. “ $x : A \wp B$ ”) says that x *outputs* (resp. *inputs*) a name of type A before continuing as described by B . Also, “ $x : \mathbf{1}$ ” says that x has completed its protocol and has no observable behavior. The judgment $\Delta \vdash^i P :: x : A$ has a rely-guarantee flavor: “ P relies on the behaviors described by Δ to offer a protocol A on x ”. Hence, the assignment $x : A$ in the right-hand side plays a special role: this is *the* only observation made about the behavior of P . Differently, the judgment $P \vdash^c \Delta$ simply reads as “ P implements the behaviors described by Δ ”; as such, all assignments in Δ are equally relevant for observing the behavior of P .

Unsurprisingly, these differences between intuitionistic and classical processes arise in their associated (typed) behavioral equivalences [18, 15, 1, 13]. For intuitionistic processes, theories of *logical relations* [18, 13] induce contextual equivalences in which only the right-hand side assignment matters in comparisons; the assignments in Δ are used to construct appropriate contexts. For classical processes, we highlight Atkey’s *observed communication semantics* [1], whose induced observational equivalence accounts for the entire typing context.

Laurent’s *negative translation* from CLL to ILL [16], denoted $(-)_R^\bullet$, translates formulas using the parameter R (an arbitrary formula in ILL) as a “residual” element. We have, e.g.,:

$$(A \otimes B)_R^\bullet = ((A_R^\bullet \multimap R) \otimes (B_R^\bullet \multimap R)) \multimap R$$

As Figure 1 shows, using $(-)_R^\bullet$ we can transform $\vdash^c \Delta$ into $(\Delta)_R^\bullet \vdash^i R$. Now, from the view of “propositions-as-sessions”, we see that $(-)_R^\bullet$ increases the size of formulas/protocols and that fixing $R = \mathbf{1}$ results into the simplest residual protocol. Given this, Laurent’s translation transforms $P \vdash^c \Delta$ into $(\Delta)_R^\bullet \vdash^i (P)^\bullet :: w : \mathbf{1}$ (for some fresh z), where $(P)^\bullet$ is a process that reflects the translation. The translation has an unfortunate effect, however: a classical process P with observable behavior given by Δ is transformed into an intuitionistic process $(P)^\bullet$ *without observable behavior* (given by $w : \mathbf{1}$). We conclude that, independently of the chosen R , the translation $(-)_R^\bullet$ alone does not adequately relate the concurrent interpretations of CLL and ILL, as it does not uniformly account for observable behavior in P and $(P)^\bullet$.

Our goal is to complement the scope of Laurent’s translation, in a way that is consistent with existing theories of (typed) behavioral equivalence for logic-based processes.

We proceed in two steps, shown in Figure 1. In the following, we shall fix $R = \mathbf{1}$ and omit “ R ” when clear from the context. First, there is a well-known translation from ILL to CLL , denoted $(-)^{\perp}$, under which a sequent $\Delta \vdash^i A$ is transformed into $\vdash^c \Delta^{\perp}, A$. Our observation is that $(-)^{\bullet\perp}$ (the composition of the two translations) goes from CLL into itself, translating $\vdash^c \Delta$ into $\vdash^c \Delta^{\bullet\perp}, \mathbf{1}$. At the level of processes, this allows us to consider the corresponding processes P and $(P)^{\bullet}$ in the common setting of classical processes. To reason about their observable behavior we employ Atkey’s observational equivalence [1], denoted “ \simeq ”.

Our second step leads to our **main contributions**: two full abstraction results that connect $\vdash^c \Delta$ and $\vdash^c \Delta^{\bullet\perp}, \mathbf{1}$ from the perspective of “propositions-as-sessions”.

- The first result, given in § 3, adopts a *denotational* approach to ensure that P (typable with $\vdash^c \Delta$) and $(P)^{\bullet}$ (typable with *both* $\Delta^{\bullet} \vdash^i \mathbf{1}$ and $\vdash^c \Delta^{\bullet\perp}, \mathbf{1}$) are behaviorally equivalent. This full abstraction result ensures that $P \simeq Q$ iff $(P)^{\bullet} \simeq (Q)^{\bullet}$ (Corollary 3.12).
- The second result, given in § 4, is an *operational* bridge between $\vdash^c \Delta$ and $\vdash^c \Delta^{\bullet\perp}, \mathbf{1}$: Corollary 4.15 ensures that $P \simeq Q$ iff $C[P] \simeq C[Q]$, where C is a so-called *transformer context*, which “adapts” observable behavior in processes using types in Δ .

Next, we recall CP (Wadler’s concurrent interpretation of CLL), Atkey’s observational equivalence, and Laurent’s translation. § 3 and § 4 develop our full abstraction results. § 5 further discusses our contributions; in particular, we discuss how they are related to the *locality* principle – one of the known distinguishing features between typed processes based on “propositions-as-sessions” [22, 9, 25].

2 Background

Propositions-as-Sessions / Classical Processes (CP). We shall work with *classical processes* (CP) as proposed by Wadler [24]. Assuming an infinite set of *names* (x, y, z, \dots) , the set of *processes* (P, Q, \dots) is defined as follows:

$$P, Q ::= \mathbf{0} \mid (\nu x)P \mid P \mid Q \mid [x \leftrightarrow y] \mid x[y].(P \mid Q) \mid x(y).P \mid !x(y).P \mid ?x[y].P \\ \mid x[i].P \mid x.\text{case}(P, Q) \mid x[] \mid x().P \quad \text{for } i \in \{1, 2\}$$

We write $P\{x/y\}$ to denote the capture-avoiding substitution of y for x in P . We have usual constructs for inaction, restriction, and parallel composition. The forwarder $[x \leftrightarrow y]$ equates x and y . We then have $x[y].(P \mid Q)$ (send the restricted name y along x , proceed as $P \mid Q$) and $x(y).P$ (receive a name z along x , proceed as $P\{z/y\}$). Processes $!x(y).P$ and $?x[y].P$ denote a replicated input (server) and a client request, respectively. Process $x[i].P$ denotes the selection of one of the two alternatives of a corresponding branching process $x.\text{case}(P, Q)$. Processes $x[]$ and $x().P$ enable coordinated closing of the session along x . In a statement, a name is *fresh* if it is not among the names of the objects of the statement (e.g., processes).

In $(\nu x)P$, name x is bound in P ; also, in $x[y].(P \mid Q)$, $x(y).P$, $!x(y).P$, and $?x[y].P$, name y is bound in P but not in Q .

The types are assigned to names and correspond to the following formulas of CLL:

$$A, B ::= \mathbf{1} \mid \perp \mid A \otimes B \mid A \wp B \mid A \oplus B \mid A \& B \mid !A \mid ?A$$

The assignment $x : A$ says that the session protocol through x goes as described by A . As we have seen, $x : A \otimes B$ and $x : A \wp B$ are read as sending and receiving along x , respectively. Also, $x : A \oplus B$ denotes the selection of either A or B along x , whereas $x : A \& B$ denotes the offer of A and B along x . Finally, $x : !A$ and $x : ?A$ assign server and client behaviors to x , respectively. There then is a clear duality in the interpretation of the following pairs: \otimes

$$\begin{array}{c}
 \frac{}{[x \leftrightarrow y] \vdash^c x : A, y : A^\perp} \text{ID} \quad \frac{}{x[] \vdash^c x : \mathbf{1}} \mathbf{1} \quad \frac{P \vdash^c \Gamma}{x().P \vdash^c \Gamma, x : \perp} \perp \\
 \\
 \frac{P \vdash^c \Gamma, y : A \quad Q \vdash^c \Delta, x : B}{x[y].(P \mid Q) \vdash^c \Gamma, \Delta, x : A \otimes B} \otimes \quad \frac{P \vdash^c \Gamma, y : A, x : B}{x(y).P \vdash^c \Gamma, x : A \wp B} \wp \quad \frac{P \vdash^c \Gamma, x : A_i}{x[i].P \vdash^c \Gamma, x : A_1 \oplus A_2} \oplus_i \\
 \\
 \frac{P \vdash^c \Gamma, x : A_1 \quad Q \vdash^c \Gamma, x : A_2}{x.\text{case}(P, Q) \vdash^c \Gamma, x : A \& B} \& \quad \frac{P \vdash^c ?\Delta, y : A}{!x(y).P \vdash^c ?\Delta, x : !A} ! \\
 \\
 \frac{P \vdash^c \Delta, x : A}{?x[y].P \vdash^c \Delta, x : ?A} ? \quad \frac{P \vdash^c \Delta, x_1 : ?A, x_2 : ?A}{P\{x_1/x_2\} \vdash^c \Delta, x_1 : ?A} \text{C} \quad \frac{P \vdash^c \Gamma}{P \vdash^c \Gamma, x : ?A} \text{W} \\
 \\
 \frac{P \vdash^c \Gamma, x : A \quad Q \vdash^c \Delta, x : A^\perp}{(\nu x)(P \mid Q) \vdash^c \Gamma, \Delta} \text{CUT} \quad \frac{P \vdash^c \Delta \quad Q \vdash^c \Gamma}{P \mid Q \vdash^c \Delta, \Gamma} \text{MIX}_2 \quad \frac{}{\mathbf{0} \vdash^c} \text{MIX}_0
 \end{array}$$

■ **Figure 2** Typing rules. CP does not include “mix”. CP₀ is CP + MIX₀, CP₀₂ is CP₀ + MIX₂.

and \wp ; \oplus and $\&$; and $!$ and $?$. It reflects reciprocity between the behavior of a name: when a process on one side sends, the process on the opposite side must receive, and vice versa. Formally, the dual type of A , denoted A^\perp , is defined as

$$\begin{array}{l}
 \mathbf{1}^\perp := \perp \quad (A \otimes B)^\perp := A^\perp \wp B^\perp \quad (A \& B)^\perp := A^\perp \oplus B^\perp \quad (!A)^\perp := ?A^\perp \\
 \perp^\perp := \mathbf{1} \quad (A \wp B)^\perp := A^\perp \otimes B^\perp \quad (A \oplus B)^\perp := A^\perp \& B^\perp \quad (?A)^\perp := !A^\perp
 \end{array}$$

Duality is an involution, i.e., $(A^\perp)^\perp = A$. We write Δ, Γ to denote *contexts*, a finite collection of assignments $x : A$. The empty context is denoted “.”. The typing judgments are then of the form $P \vdash^c \Delta$, with typing rules as in Figure 2. For technical convenience, we shall consider mix principles (rules MIX₀ and MIX₂, not included in [24]), which enable the typing of useful forms of process composition. This way, in § 3 we will consider CP₀: the variant of CP with MIX₀; in § 4 we will consider CP₀₂: the extension of CP₀ with MIX₂.

Note that the type system CP₀ corresponds exactly to the sequent calculus for CLL if we ignore name and process annotations. This correspondence goes beyond typing: an important aspect of “propositions-as-sessions” is that the dynamic behavior of processes (process reductions) corresponds to simplification of proofs (cut elimination). In the following we will not need this reduction semantics, which can be found in, e.g., [24]. Rather, we will use the denotational semantics of CP₀ as defined by Atkey [1], which we recall next.

Denotational Semantics for CP. We adopt Atkey’s denotational semantics for CP₀ [1], which allows us to reason about observational equivalence of processes. Here we recall the notions of configurations, observations, and denotations as needed for our purposes.

The observational equivalence on processes relies on the notion of *configuration*, which is a process that has some of its names selected for the purposes of “observations”. Configurations, defined in Figure 3, are typed as $C \vdash_{\text{cfg}} \Delta \mid \Theta$, where Δ contains free/unconnected names, and Θ contains the names that we intend to observe. Rule C:CUT, for example, states that we can compose two configurations along a name x , and make the name observable.

$$\begin{array}{c}
\text{C:PROC} \\
\frac{P \vdash^c \Gamma}{P \vdash_{\text{cfg}} \Gamma \mid \cdot} \\
\\
\text{C:W} \\
\frac{C \vdash_{\text{cfg}} \Gamma \mid \Theta}{C \vdash_{\text{cfg}} \Gamma, x : ?A \mid \Theta} \\
\\
\text{C:CUT} \\
\frac{C_1 \vdash_{\text{cfg}} \Gamma_1, x : A \mid \Theta_1 \quad C_2 \vdash_{\text{cfg}} \Gamma_2, x : A^\perp \mid \Theta_2}{C_1 \mid_x C_2 \vdash_{\text{cfg}} \Gamma_1, \Gamma_2 \mid \Theta_1, \Theta_2, x : A} \\
\\
\text{C:CON} \\
\frac{C \vdash_{\text{cfg}} \Gamma, x_1 : ?A, x_2 : ?A \mid \Theta}{C\{x_1/x_2\} \vdash_{\text{cfg}} \Gamma, x_1 : ?A \mid \Theta} \\
\\
\text{C:0} \\
\frac{}{\mathbf{0} \vdash_{\text{cfg}} \cdot \mid \cdot}
\end{array}$$

■ **Figure 3** Classical Processes: Configurations.

$$\begin{array}{c}
\frac{}{\mathbf{0} \Downarrow ()} \text{STOP} \quad \frac{C[C'\{\{x/y\}\}] \Downarrow \theta[x \mapsto a]}{C[[x \leftrightarrow y] \mid_x C'] \Downarrow \theta[x \mapsto a, y \mapsto a]} \text{LINK} \quad \frac{C[P \mid_x Q] \Downarrow \theta[x \mapsto a]}{C[(\nu x)(P \mid Q)] \Downarrow \theta} \text{COMM} \\
\\
\frac{C[\mathbf{0}] \Downarrow \theta}{C[\mathbf{0}] \Downarrow \theta} \mathbf{0} \quad \frac{C[P \mid_y (Q \mid_x R)] \Downarrow \theta[x \mapsto a, y \mapsto b]}{C[x[y].(P \mid Q) \mid_x x(y).R] \Downarrow \theta[x \mapsto (a, b)]} \otimes \otimes \\
\\
\frac{C[P] \Downarrow \theta}{C[x[] \mid_x x().P] \Downarrow \theta[x \mapsto *]} \mathbf{1} \perp \quad \frac{C[P \mid_x Q_i] \Downarrow \theta[x \mapsto a]}{C[x[i].P \mid_x x.\text{case}(Q_0, Q_1)] \Downarrow \theta[x \mapsto (i, a)]} \oplus \& \\
\\
\frac{C[P \mid_y Q] \Downarrow \theta[y \mapsto a]}{C[!x(y).P \mid_x ?x[y].Q] \Downarrow \theta[x \mapsto \{a\}]} \text{!} ? \quad \frac{C[C'] \Downarrow \theta}{C[!x(y).P \mid_x C'] \Downarrow \theta[x \mapsto \emptyset]} \text{!} W \\
\\
\frac{C[!x_1(y).P \mid_{x_1} (!x_2(y).P \mid_{x_2} C')] \Downarrow \theta[x_1 \mapsto \alpha, x_2 \mapsto \beta]}{C[!x_1(y).P \mid_{x_1} C'\{x_1/x_2\}] \Downarrow \theta[x_1 \mapsto \alpha \uplus \beta]} \text{!} C \quad \frac{C' \Downarrow \theta \quad C \equiv C'}{C \Downarrow \theta} \equiv
\end{array}$$

■ **Figure 4** Classical Processes: Observations.

Observations for configurations are given in Figure 4. The observation relation $C \Downarrow \theta$ is defined for closed configurations $C \vdash_{\text{cfg}} \cdot \mid \Theta$, and the shape observation $\theta \in \llbracket \Theta \rrbracket$ is defined based on the shape of types in Θ . For a type A , the set of observations $\llbracket A \rrbracket$ is defined as

$$\begin{aligned}
\llbracket \mathbf{1} \rrbracket &= \llbracket \perp \rrbracket = \{*\} & \llbracket !A \rrbracket &= \llbracket ?A \rrbracket = \mathcal{M}_f(\llbracket A \rrbracket) \\
\llbracket A \otimes B \rrbracket &= \llbracket A \otimes B \rrbracket = \llbracket A \rrbracket \times \llbracket B \rrbracket & \llbracket A_0 \oplus A_1 \rrbracket &= \llbracket A_0 \& A_1 \rrbracket = \sum_{i \in \{0,1\}} \llbracket A_i \rrbracket
\end{aligned}$$

and we set $\llbracket \Theta \rrbracket = \llbracket x_1 : A_1, \dots, x_n : A_n \rrbracket = \llbracket A_1 \rrbracket \times \dots \times \llbracket A_n \rrbracket$. Here $\mathcal{M}_f(X)$ denotes finite multisets with elements from X . We use the standard notations \emptyset , \uplus , and $\{a_1, \dots, a_n\}$ to denote the empty multiset, multiset union, and multiset literals, respectively.

If $\theta \in \llbracket x_1 : A_1, \dots, x_n : A_n \rrbracket$, then we write $\theta[x_i \mapsto \theta_i]$ for the observation which is identical to θ , except that its i th component is set to θ_i . In Figure 4, Rule Stop says that $\mathbf{0}$ has no observations; Rule $\otimes \otimes$ collects observations a and b into a single observation (a, b) .

Using these notions, there is an immediate canonical notion of observational equivalence. In the following, we write $P, Q \vdash^c \Gamma$ whenever $P \vdash^c \Gamma$ and $Q \vdash^c \Gamma$ hold.

► **Definition 2.1** (Observational equivalence [1]). *Let $P, Q \in \text{CP}_0$ such that $P, Q \vdash^c \Gamma$. They are observationally equivalent, written $P \simeq Q$, if for all configurations-process context $C[-]$ where $C[P], C[Q] \vdash^c \cdot \mid \Theta$, and all $\theta \in \llbracket \Theta \rrbracket$, $C[P] \Downarrow \theta \Leftrightarrow C[Q] \Downarrow \theta$.*

$$\begin{aligned}
 \llbracket [x \leftrightarrow y] \vdash^c x : A, y : A^\perp \rrbracket &= \{(a, a) \mid a \in \llbracket A \rrbracket\} & \llbracket x[] \vdash^c x : \mathbf{1} \rrbracket &= \{(*)\} & \llbracket \mathbf{0} \vdash^c \rrbracket &= \{()\} \\
 \llbracket x().P \vdash^c \Gamma, x : \perp \rrbracket &= \{(\gamma, *) \mid \gamma \in \llbracket P \vdash^c \Gamma \rrbracket\} \\
 \llbracket (\nu x)(P \mid Q) \vdash^c \Gamma, \Delta \rrbracket &= \{(\gamma, \delta) \mid (\gamma, a) \in \llbracket P \vdash^c \Gamma, x : A \rrbracket, (\delta, a) \in \llbracket Q \vdash^c \Delta, x : A^\perp \rrbracket\} \\
 \llbracket x[y].(P \mid Q) \vdash^c \Gamma, \Delta, x : A \otimes B \rrbracket &= \left\{ (\gamma, \delta, (a, b)) \mid \begin{array}{l} (\gamma, a) \in \llbracket P \vdash^c \Gamma, y : A \rrbracket, \\ (\delta, b) \in \llbracket Q \vdash^c \Delta, x : B \rrbracket \end{array} \right\} \\
 \llbracket x(y).P \vdash^c \Delta, x : A \wp B \rrbracket &= \{(\gamma, \delta, (a, b)) \mid (\gamma, a, b) \in \llbracket P \vdash^c \Delta, y : A, x : B \rrbracket\} \\
 \llbracket x[i].P \vdash^c \Gamma, x : A_1 \oplus A_2 \rrbracket &= \{(\gamma, (i, a)) \mid (\gamma, a) \in \llbracket P \vdash^c \Gamma, x : A_i \rrbracket\} \\
 \llbracket x.case(P_1, P_2) \vdash^c \Gamma, x : A_1 \& A_2 \rrbracket &= \bigcup_{i \in \{1, 2\}} \{(\gamma, (i, a)) \mid (\gamma, a) \in \llbracket P_i \vdash^c \Gamma, x : A_i \rrbracket\} \\
 \llbracket !x(y).P \vdash^c ?\Delta, x : !A \rrbracket &= \left\{ \begin{array}{l} (\uplus_{j=1}^k \alpha_j^1, \dots, \uplus_{j=1}^k \alpha_j^n, \uparrow a_1, \dots, a_n) \\ \mid \forall i \in \{1, \dots, k\}. (\alpha_i^1, \dots, \alpha_i^k, a_i) \in \llbracket P \vdash^c ?\Delta, y : A \rrbracket \end{array} \right\} \\
 \llbracket ?x[y].P \vdash^c \Gamma, x : ?A \rrbracket &= \{(\gamma, \uparrow a) \mid (\gamma, a) \in \llbracket P \vdash^c \Gamma, y : A \rrbracket\} \\
 \llbracket P \vdash^c \Gamma, x : ?A \rrbracket &= \{(\gamma, \emptyset) \mid \gamma \in \llbracket P \vdash^c \Gamma \rrbracket\} \\
 \llbracket P\{x_1/x_2\} \vdash^c \Gamma, x_1 : ?A \rrbracket &= \{(\gamma, \alpha_1 \uplus \alpha_2) \mid (\gamma, \alpha_1, \alpha_2) \in \llbracket P \vdash^c \Gamma, x_1 : ?A, x_2 : ?A \rrbracket\}
 \end{aligned}$$

■ **Figure 5** Classical Processes: Denotational Semantics.

We take this notion of observational equivalence as *the* equivalence of CP_0 , sometimes writing $P \simeq Q \vdash^c \Gamma$ to emphasize the typing of processes we are comparing. Establishing observational equivalence of two processes directly is complicated, due to the universal quantification over all potential configurations C . To establish equivalence in a compositional way, we recall Atkey's notion of denotational semantics for CP in Figure 5: it assigns to each process $P \vdash^c \Delta$ a denotation $\llbracket P \vdash^c \Delta \rrbracket$ as a subset of observations $\llbracket \Delta \rrbracket$ on its names. When the typing of a process P is clear from the context we simply write $\llbracket P \rrbracket \subseteq \llbracket \Delta \rrbracket$. The denotational semantics are sound and complete w.r.t. the observations:

► **Theorem 2.2** (Adequacy [1]). *If $C \vdash^c \cdot \mid \Theta$, then $C \Downarrow \theta$ iff $\theta \in \llbracket C \vdash^c \cdot \mid \Theta \rrbracket$.*

Hence, we can use denotational semantics to prove observational equivalence:

► **Corollary 2.3** ([1]). *If $P, Q \vdash^c \Gamma$ and $\llbracket P \rrbracket = \llbracket Q \rrbracket$, then $P \simeq Q$.*

Above, the condition $P, Q \vdash^c \Gamma$ is important, as there are processes with different types that have the same denotations. Examples are $x[1].x[] \vdash^c x : \mathbf{1} \oplus \mathbf{1}$ and $x.case(x[], x[]) \vdash^c x : \mathbf{1} \& \mathbf{1}$.

► **Remark 2.4.** Atkey shows that \simeq captures many equalities on processes induced by proof transformations, such as cut permutations and commuting conversions; see [1, Sect. 5].

Laurent's Translation $(-)_R^\bullet$. As mentioned above, Laurent gives a parametric translation from CLL to ILL. Here we recall this translation following [16, §2.1], considering only the class of formulas needed for our purposes. The formulas of ILL are built using the grammar:

$$I, J ::= \mathbf{1} \mid I \otimes J \mid I \multimap J \mid I \oplus J \mid I \& J \mid !I$$

■ **Table 1** Translations $(-)_R^\bullet$ and $(-)_R^{\bullet\perp}$.

F	F_R^\bullet (ILL)	F_R^\bullet (CLL)	$F_R^{\bullet\perp}$
\perp	$\mathbf{1}$	$\mathbf{1}$	\perp
$\mathbf{1}$	$\mathbf{1} \multimap R$	$\perp \wp R$	$\mathbf{1} \otimes R^\perp$
$A \otimes B$	$((A_R^\bullet \multimap R) \otimes (B_R^\bullet \multimap R)) \multimap R$	$((A_R^{\bullet\perp} \wp R) \otimes (B_R^{\bullet\perp} \wp R))^\perp \wp R$	$((A_R^{\bullet\perp} \wp R) \otimes (B_R^{\bullet\perp} \wp R)) \otimes R^\perp$
$A \wp B$	$A_R^\bullet \otimes B_R^\bullet$	$A_R^\bullet \otimes B_R^\bullet$	$A_R^{\bullet\perp} \wp B_R^{\bullet\perp}$
$A \oplus B$	$((A_R^\bullet \multimap R) \oplus (B_R^\bullet \multimap R)) \multimap R$	$((A_R^{\bullet\perp} \wp R) \oplus (B_R^{\bullet\perp} \wp R))^\perp \wp R$	$((A_R^{\bullet\perp} \wp R) \oplus (B_R^{\bullet\perp} \wp R)) \otimes R^\perp$
$A \& B$	$A_R^\bullet \oplus B_R^\bullet$	$A_R^\bullet \oplus B_R^\bullet$	$A_R^{\bullet\perp} \& B_R^{\bullet\perp}$
$!A$	$!(A_R^\bullet \multimap R) \multimap R$	$!(A_R^{\bullet\perp} \wp R)^\perp \wp R$	$!(A_R^{\bullet\perp} \wp R) \otimes R^\perp$
$?A$	$!((A_R^\bullet \multimap R) \multimap R)$	$!((A_R^{\bullet\perp} \wp R)^\perp \wp R)$	$?((A_R^{\bullet\perp} \wp R) \otimes R^\perp)$

The sequent calculus for ILL (omitted for space reasons) works on the judgments of the form $\Delta \vdash^i I$. Let R be a fixed but arbitrary formula in ILL. We have the following derivable rules:

$$\frac{\Gamma, I \vdash^i R}{\Gamma \vdash^i I \multimap R} R_R \qquad \frac{\Gamma \vdash^i I \quad R \vdash^i R}{\Gamma, I \multimap R \vdash^i R} R_L$$

By using Rule R_R , the formula I in the left-hand side of \vdash^i becomes $I \multimap R$ on the right-hand side. Similarly, by using Rule R_L , the formula I on the right-hand side of \vdash^i becomes $I \multimap R$ on the left-hand side. Moving the formula I from one side to the other of \vdash^i results in $I \multimap R$, which allows us to mimic in ILL the one-sided sequents of CLL. The translation in ILL of a CLL formula F , denoted $(F)_R^\bullet$, is inductively defined using this movement of formulas; see Table 1 (second column).

The amount of (nested) occurrences of “ $\multimap R$ ” indicates how many times a formula has to be moved. Not all connectives require such transformations; we will expand on this in § 3. This translation extends to contexts as expected; it is correct, in the following sense:

► **Theorem 2.5** ([16]). *If $\vdash^c \Delta$ is provable in CLL then $\Delta_R^\bullet \vdash^i R$ is provable in ILL.*

Given an R such that $\bigotimes_n R \vdash^i R$ (for all $n > 0$), the theorem extends to CLL_{02} – CLL with the corresponding MIX_0 and MIX_2 rules (obtained from CP_{02} in Figure 2). The following result considers the case $R = \mathbf{1}$; it will be useful in § 4, where we use CP_{02} .

► **Lemma 2.6** ([16]). *Let $R = \mathbf{1}$. $\vdash^c \Delta$ is provable in CLL_{02} iff $\Delta_R^\bullet \vdash^i R$ is provable in ILL.*

As already mentioned, since we interpret propositions as sessions, we pick the simplest residual formula/protocol that satisfies the premise of Lemma 2.6, i.e., we fix $R = \mathbf{1}$. Considering this, in the remainder of the paper we refer to the translation simply as $(-)_R^\bullet$.

3 A Denotational Characterization of Laurent’s Translation

Here we study the effect of Laurent’s translation $(-)_R^\bullet$ on processes typed under CP_0 . We prove our first full abstraction result (Corollary 3.12) by lifting $(-)_R^\bullet$ to the level of denotations.

The Composed Translation. As discussed in § 1, we wish to compare processes in the uniform setting of CLL. We know that if $\Delta \vdash^i A$ is provable in ILL, then $\vdash^c \Delta^\perp, A$ is provable in CLL (see, e.g., [16]). Hence, we can interpret sequents in ILL as sequents in CLL.

Notice that formulas in ILL can be treated as formulas in CLL by letting $A \multimap B := A^\perp \wp B$. Using this transformation within the composition of $(-)^{\bullet}$ and $(-)^{\perp}$, we obtain the desired transformation on CLL proofs. From now on, we shall write $(-)^{\bullet\perp}$ to denote the translation given in Table 1 (rightmost column).

► **Theorem 3.1.** *If $\vdash^c \Delta$ is provable in CLL_0 , then $\vdash^c \Delta^{\bullet\perp}, \mathbf{1}$ is provable in CLL_0 .*

We shall write $A \in \text{CP}_0$, when is clear from the context that A is a type. Similarly, $A \in \text{CLL}_0$ says that A is a formula in CLL_0 .

The Translation on Processes. $(-)^{\bullet\perp}$ induces a translation on processes, denoted $(-)^{\bullet}$, which is defined inductively on typing derivations (Definition 3.3). This translation is the computational interpretation of the composition of the two steps in Figure 1. Before detailing its definition, we examine two illustrative cases: output and input.

Let us first consider the process $P = x[y].(P_1 \mid P_2)$, which is typed as follows:

$$\frac{P_1 \vdash^c \Delta, y : A \quad P_2 \vdash^c \Gamma, x : B}{x[y].(P_1 \mid P_2) \vdash^c \Delta, \Gamma, x : A \otimes B} \otimes$$

From the standpoint of the “propositions-as-sessions” interpretation, by Theorem 2.5 there exists a process $(P)^{\bullet}$ and fresh names z and w such that $\Delta^{\bullet}, z : (A \otimes B)^{\bullet} \vdash^i (P)^{\bullet} :: w : \mathbf{1}$. As Table 1 (second column) shows, $(A \otimes B)^{\bullet} = ((A^{\bullet} \multimap \mathbf{1}) \otimes (B^{\bullet} \multimap \mathbf{1})) \multimap \mathbf{1}$. To determine the shape of $(P)^{\bullet}$, we can reason inductively and apply Theorem 2.5 to the judgments $P_1 \vdash^c \Delta, y : A$ and $P_2 \vdash^c \Gamma, x : B$. This gives us $\Delta^{\bullet}, y : A^{\bullet} \vdash^i (P_1)^{\bullet} :: z_1 : \mathbf{1}$ and $\Gamma^{\bullet}, x : B^{\bullet} \vdash^i (P_2)^{\bullet} :: z_2 : \mathbf{1}$, respectively. We can then obtain the following typing derivation (and shape) for $(P)^{\bullet}$:

$$\frac{\frac{\Delta^{\bullet}, y : A^{\bullet} \vdash^i (P_1)^{\bullet} :: z_1 : \mathbf{1} \quad \Gamma^{\bullet}, x : B^{\bullet} \vdash^i (P_2)^{\bullet} :: z_2 : \mathbf{1}}{\Delta^{\bullet} \vdash^i z_1(y).(P_1)^{\bullet} :: z_1 : A^{\bullet} \multimap \mathbf{1} \quad \Gamma^{\bullet} \vdash^i z_2(x).(P_2)^{\bullet} :: z_2 : B^{\bullet} \multimap \mathbf{1}}}{\Delta^{\bullet}, \Gamma^{\bullet} \vdash^i z_2[z_1].(z_1(y).(P_1)^{\bullet} \mid z_2(x).(P_2)^{\bullet}) :: z_2 : (A^{\bullet} \multimap \mathbf{1}) \otimes (B^{\bullet} \multimap \mathbf{1})} \quad (\star)}{\Delta^{\bullet}, \Gamma^{\bullet}, x' : (A \otimes B)^{\bullet} \vdash^i \underbrace{x'[z_2].(z_2[z_1].(z_1(y).(P_1)^{\bullet} \mid z_2(x).(P_2)^{\bullet}) \mid [x' \leftrightarrow w])}_{(P)^{\bullet}} :: w : \mathbf{1}}$$

where (\star) stands for $x' : \mathbf{1} \vdash^i [x' \leftrightarrow w] :: w : \mathbf{1}$. Above, we see how each nested “ $\multimap \mathbf{1}$ ” induced by Laurent’s translation entails extra actions on the level of processes, due to the interpretation of \multimap as input (when introduced on the right, as in this case): moving $y : A^{\bullet}$ and $x : B^{\bullet}$ to the right-hand side induces the inputs along z_1 and z_2 , respectively. Subsequently, we use the \otimes rule on the right, which produces the output on z_2 ; we then move the resulting assignment for z_2 back to the left, finally obtaining $x' : (A \otimes B)^{\bullet}$. This last movement adds the final “ $\multimap \mathbf{1}$ ”: because it is introduced on the left, we obtain an output along x' . At this point, we can return to the classical setting by applying the translation $(-)^{\perp}$ to the derivation above, which leads to the following typing derivation for $(P)^{\bullet}$ in CP_0 :

$$\frac{\frac{(P_1)^{\bullet} \vdash^c \Delta^{\bullet\perp}, y : A^{\bullet\perp}, z_1 : \mathbf{1} \quad (P_2)^{\bullet} \vdash^c \Gamma^{\bullet\perp}, x : B^{\bullet\perp}, z_2 : \mathbf{1}}{z_1(y).(P_1)^{\bullet} \vdash^c \Delta^{\bullet\perp}, z_1 : A^{\bullet\perp} \wp \mathbf{1} \quad z_2(x).(P_2)^{\bullet} \vdash^c \Gamma^{\bullet\perp}, z_2 : B^{\bullet\perp} \wp \mathbf{1}}}{z_2[z_1].(z_1(y).(P_1)^{\bullet} \mid z_2(x).(P_2)^{\bullet}) \vdash^c \Delta^{\bullet\perp}, \Gamma^{\bullet\perp}, z_2 : (A^{\bullet\perp} \wp \mathbf{1}) \otimes (B^{\bullet\perp} \wp \mathbf{1})} \quad (\star\star)}{\underbrace{x'[z_2].(z_2[z_1].(z_1(y).(P_1)^{\bullet} \mid z_2(x).(P_2)^{\bullet}) \mid [x' \leftrightarrow w])}_{(P)^{\bullet}} \vdash^c \Delta^{\bullet\perp}, \Gamma^{\bullet\perp}, x' : (A \otimes B)^{\bullet\perp}, w : \mathbf{1}}$$

where $(\star\star)$ stands for $[x' \leftrightarrow w] \vdash^c x' : \perp, w : \mathbf{1}$. Importantly, while the translation $(-)^{\perp}$ modifies the types for $(P)^{\bullet}$, it does not change its shape. Also, it is worth noticing how the output on x in P is mimicked by $(P)^{\bullet}$ through the output on z_2 , not by the output on x' .

Now consider the case when $Q = x(y).Q_1$, which is typed as:

$$\frac{Q_1 \vdash^c \Delta, y : A, x : B}{x(y).Q_1 \vdash^c \Delta, x : A \wp B}$$

In this case, we expect to obtain a process $(Q)^\bullet$ such that $\Delta^\bullet, x' : (A \wp B)^\bullet \vdash^i (Q)^\bullet :: w : \mathbf{1}$, where $(A \wp B)^\bullet = A^\bullet \otimes B^\bullet$ (cf. Table 1, second column). By reasoning inductively on $Q_1 \vdash^c \Delta, y : A, x : B$, we obtain $\Delta^\bullet, y : A^\bullet, x' : B^\bullet \vdash^i (Q_1)^\bullet :: w : \mathbf{1}$, which enables us to obtain the following derivation:

$$\frac{\Delta^\bullet, y : A^\bullet, x' : B^\bullet \vdash^i (Q_1)^\bullet :: w : \mathbf{1}}{\Delta^\bullet, x' : A^\bullet \otimes B^\bullet \vdash^i \underbrace{x'(y).(Q_1)^\bullet}_{(Q)^\bullet} :: w : \mathbf{1}}$$

Differently from the case of \otimes , here the transformation $(-)^{\bullet}$ does not add any “ $\rightarrow \mathbf{1}$ ”. This is relevant, because it ensures that the process $(Q)^\bullet$ does not have input/output actions in front of the input on x' , which mimics the input on x in Q . By applying the translation $(-)^{\perp}$ to the derivation above, we obtain the following typing derivation for $(Q)^\bullet$ in CP_0 :

$$\frac{(Q_1)^\bullet \vdash^c \Delta^{\bullet\perp}, y : A^{\bullet\perp}, x' : B^{\bullet\perp}, w : \mathbf{1}}{\underbrace{x'(y).(Q_1)^\bullet}_{(Q)^\bullet} \vdash^c \Delta^{\bullet\perp}, x' : (A \wp B)^{\bullet\perp}, w : \mathbf{1}}$$

Once again, notice that the translation $(-)^{\perp}$ does not modify the shape of $(Q)^\bullet$.

A key observation is that although $P = x[y].(P_1 \mid P_2)$ and $Q = x(y).Q_1$ are compatible (i.e., they have complementary actions on x), their translations $(P)^\bullet$ and $(Q)^\bullet$ are not. In general, given two composable processes $P \vdash^c \Delta, x : A$ and $Q \vdash^c \Gamma, x : A^\perp$, we have:

$$(P)^\bullet \vdash^c \Delta^{\bullet\perp}, x' : A^{\bullet\perp}, w : \mathbf{1} \quad (Q)^\bullet \vdash^c \Gamma^{\bullet\perp}, x' : A^{\perp\bullet\perp}, z : \mathbf{1}$$

and so $(P)^\bullet$ and $(Q)^\bullet$ cannot be composed directly: the types of x' are not dual ($(A^{\bullet\perp})^\perp \neq A^{\perp\bullet\perp}$). To circumvent this difficulty, we shall consider *synchronizer processes* $\mathcal{S}_{z,w}^A$ such that

$$\mathcal{S}_{z,w}^A \vdash^c w : A^\bullet \otimes \perp, z : A^{\perp\bullet} \otimes \perp, s : \mathbf{1}$$

Using synchronizers, a mediated composition between $(P)^\bullet$ and $(Q)^\bullet$ is then possible:

$$(\nu w)((\nu z)(z(y).(Q)^\bullet \mid \mathcal{S}_{z,w}^A) \mid w(x').(P)^\bullet)$$

Synchronizer processes have a purely logical origin: Laurent [16] shows that for any A the sequent $\vdash^c A^\bullet \otimes \perp, A^{\perp\bullet} \otimes \perp, \mathbf{1}$ is provable; using this result, the definition of synchronizers (given next) arises by reading off the process associated with this proof.

► **Definition 3.2** (Synchronizer). *Given $F \in CP_0$ and names z, w , and s , we define the synchronizer process $\mathcal{S}_{z,w}^A$, satisfying $\mathcal{S}_{z,w}^A \vdash^c z : A^\bullet \otimes \perp, w : A^{\perp\bullet} \otimes \perp, s : \mathbf{1}$, by recursion on A .*

Armed with the notion of synchronizer processes, we can finally define:

► **Definition 3.3** (Laurent's translation on processes). *Let $\mathcal{S}_{z,w}^A$ be a synchronizer as in Definition 3.2. Given a typed process $P \vdash^c \Delta$, we define $(P)^\bullet$ inductively in Figure 6.*

The next lemma ensures that for a given CP_0 process P , $(P)^\bullet$ is well-typed.

$$\begin{aligned}
 ([x \leftrightarrow y])^\bullet &= x'[x].([x \leftrightarrow y] \mid [x' \leftrightarrow w]) \\
 (x().P)^\bullet &= x'().(P)^\bullet \\
 (x[])^\bullet &= x'[x].(x[] \mid [x' \leftrightarrow w]) \\
 (x(y).P)^\bullet &= x'(y).(P)^\bullet \\
 (x[y].(P \mid Q))^\bullet &= x'[z_2].(z_2[z_1].(z_1(y).(P)^\bullet \mid z_2(x).(Q)^\bullet) \mid [x' \leftrightarrow w]) \\
 (x.\text{case}(P_1, P_2))^\bullet &= x'.\text{case}((P_1)^\bullet, (P_2)^\bullet) \\
 (x[i].P)^\bullet &= x'[z].(z[i].z(y).(P)^\bullet \mid [x' \leftrightarrow w]) \\
 (!x(y).P)^\bullet &= x'[x].(!x(v).v(y).(P)^\bullet \mid [x' \leftrightarrow w]) \\
 (?x[y].P)^\bullet &= ?x'[m].m[v].(v(y).(P)^\bullet \mid [m \leftrightarrow w]) \\
 ((\nu x)(P \mid Q))^\bullet &= (\nu w)((\nu z)(z(x').(Q)^\bullet \mid \mathcal{S}_{z,w}^A \mid w(x').(P)^\bullet)
 \end{aligned}$$

■ **Figure 6** Laurent’s translation on CP processes (Definition 3.3).

► **Lemma 3.4.** *Let $P \vdash^c \Delta$ be a process in CP_0 , then $(P)^\bullet \vdash^c \Delta^{\bullet\perp}, w : \mathbf{1}$.*

► **Example 3.5.** To illustrate mediated composition, consider the processes $x[] \vdash^c x : \mathbf{1}$ and $x().P \vdash^c \Delta, x : \perp$. By Lemma 3.4, we have $x'[x].(x[] \mid [x' \leftrightarrow w]) \vdash^c x' : \mathbf{1}^{\bullet\perp}, w : \mathbf{1}$ and $z(m).m().(P)^\bullet \vdash^c \Delta^{\bullet\perp}, m : \perp, z : \mathbf{1}$, respectively. These two processes can be composed with the synchronizer for $A = \mathbf{1}$:

$$\mathcal{S}_{w,z}^{\mathbf{1}} = w[x'].(x'(x).z[m_1].([m_1 \leftrightarrow x] \mid [x' \leftrightarrow z]) \mid [w \leftrightarrow s])$$

By expanding Definition 3.3, we obtain the following observational equivalence:

$$\begin{aligned}
 ((\nu x)(x[] \mid x().P))^\bullet &= (\nu z)((\nu w)(w(x').(x[])^\bullet \mid \mathcal{S}_{z,w}^{\mathbf{1}} \mid z(m).(x().P)^\bullet) \\
 &= (\nu z)((\nu w)(w(x').x'[x].(x[] \mid [x' \leftrightarrow w]) \\
 &\quad \mid w[x'].(x'(x).z[m].([m \leftrightarrow x] \mid [x' \leftrightarrow z]) \mid [w \leftrightarrow s])) \\
 &\quad \mid z(m).m().(P)^\bullet) \\
 &\simeq (P)^\bullet \{s/z\}
 \end{aligned}$$

Properties. The logical translations strongly suggest that P and $(P)^\bullet$ should be equivalent in some sense. How to state this relation? Our technical insight is to bring $(-)^{\bullet\perp}$ to the level of denotations: we define the function $\mathbb{L}_A(-) : \llbracket A \rrbracket \rightarrow \llbracket A^{\bullet\perp} \rrbracket$, which “saturates” $\llbracket A \rrbracket$ by adding as many “*” (the observation of $\mathbf{1}$ and \perp) as residual \perp s and $\mathbf{1}$ s are induced by $(-)^{\bullet\perp}$.

► **Definition 3.6** (Transformations on Denotations). *Given $A \in CP_0$, $\mathbb{L}_A(-) : \llbracket A \rrbracket \mapsto \llbracket A^{\bullet\perp} \rrbracket$ is defined in Figure 7. Given $\Delta = x_1 : A_1, \dots, x_n : A_n$, we define $\mathbb{L}_\Delta^*(-) : \llbracket \Delta \rrbracket \mapsto \llbracket \Delta^{\bullet\perp}, \mathbf{1} \rrbracket$ as*

$$\mathbb{L}_\Delta^*((a_1, \dots, a_n)) = (\mathbb{L}_{A_1}(a_1), \dots, \mathbb{L}_{A_n}(a_n), *)$$

Our goal is to show that $\mathbb{L}_\Delta^*(\llbracket P \vdash^c \Delta \rrbracket) = \llbracket (P)^\bullet \vdash^c \Delta^{\bullet\perp}, w : \mathbf{1} \rrbracket$ (Theorem 3.11). In particular we use synchronizers (and their observations) to prove the property for processes with cut. Also, the following two properties will be useful:

► **Lemma 3.7.** *For any A and Δ in CP_0 , both $\mathbb{L}_A(-)$ and $\mathbb{L}_\Delta^*(-)$ are injective.*

$$\begin{aligned}
\mathbb{L}_\perp(*) &= * & \mathbb{L}_{!A}(\wr a_1, a_2 \S) &= (\wr(\mathbb{L}_A(a_1), *), (\mathbb{L}_A(a_2), *) \S, *) \\
\mathbb{L}_1(*) &= (*, *) & \mathbb{L}_{?A}(\wr a_1, a_2 \S) &= \wr((\mathbb{L}_A(a_1), *), *), ((\mathbb{L}_A(a_2), *), *) \S) \\
\mathbb{L}_{A \wp B}((a, b)) &= (\mathbb{L}_A(a), \mathbb{L}_B(b)) & \mathbb{L}_{A \otimes B}((a, b)) &= ((\mathbb{L}_A(a), *), (\mathbb{L}_B(b), *), *) \\
\mathbb{L}_{A_1 \& A_2}((i, a)) &= (i, \mathbb{L}_{A_i}(a)) & \mathbb{L}_{A_1 \oplus A_2}((i, a)) &= ((i, (\mathbb{L}_{A_i}(a), *)), *)
\end{aligned}$$

■ **Figure 7** Transformation on denotations induced by Laurent’s translation (Definition 3.6). The generalized definitions $\mathbb{L}_{!A}(\wr a_1, \dots, a_n \S)$ and $\mathbb{L}_{?A}(\wr a_1, \dots, a_n \S)$ arise as expected.

► **Lemma 3.8.** *Let $?A \in CP_0$. Suppose $\alpha_j \in \llbracket ?A \rrbracket$ for all $j = 1, \dots, k$. Then $\mathbb{L}_{?A}(\uplus_{j=1}^k \alpha_j) = \uplus_{j=1}^k \mathbb{L}_{?A}(\alpha_j)$.*

Proof sketch. It follows by Definition 3.6 and definition of \uplus . ◀

As explained above, synchronizers mediate between the translation of two processes. The following lemma ensures that a synchronizer $\mathcal{S}_{w,z}^A$ acts as a forwarder:

► **Lemma 3.9.** *Let $\Delta = z : A^\bullet \otimes \perp, w : A^\perp \bullet \otimes \perp, s : \mathbf{1}$, for some $A \in CP_0$. Then $\llbracket \mathcal{S}_{z,w}^A \vdash^c \Delta \rrbracket = \{((\mathbb{L}_A(a), *), (\mathbb{L}_{A^\perp}(a), *), *) \mid a \in \llbracket A \rrbracket\}$.*

Proof sketch. The proof follows by induction on A . ◀

The next lemma is crucial to ensure that the denotations of a composed process correspond (in the sense of Definition 3.6) with those of its translation:

► **Lemma 3.10** (Synchronizers are well-behaved). *Let $P, P', Q, Q' \in CP_{02}$, such that*

$$\begin{aligned}
\llbracket P' \vdash^c \Delta', x' : A^{\bullet \perp}, w : \mathbf{1} \rrbracket &= \mathbb{L}_{\Delta, x:A}^* (\llbracket P \vdash^c \Delta, x : A \rrbracket) \\
\llbracket Q' \vdash^c \Gamma', x' : A^\perp \bullet, z : \mathbf{1} \rrbracket &= \mathbb{L}_{\Gamma, y:A^\perp}^* (\llbracket Q \vdash^c \Gamma, x : A^\perp \rrbracket)
\end{aligned}$$

Then:

$$(\delta, \gamma) \in \llbracket (\nu x)(P \mid Q) \rrbracket \Leftrightarrow (\mathbb{L}_\Delta(\delta), \mathbb{L}_\Gamma(\gamma), *) \in \llbracket (\nu w)(w(x').P' \mid (\nu z)(z(x').Q' \mid \mathcal{S}_{z,w}^A)) \rrbracket.$$

Proof.

$$\begin{aligned}
(\delta, \gamma) \in \llbracket (\nu x)(P \mid Q) \rrbracket & \\
\Leftrightarrow (\delta, a) \in \llbracket P \rrbracket \wedge (\gamma, a) \in \llbracket Q \rrbracket & \quad \text{(by Figure 5)} \\
\Leftrightarrow (\mathbb{L}_\Delta(\delta), \mathbb{L}_A(a), *) \in \llbracket P' \rrbracket \wedge (\mathbb{L}_\Gamma(\gamma), \mathbb{L}_{A^\perp}(a), *) \in \llbracket Q' \rrbracket & \quad \text{(by assumption)} \\
\Leftrightarrow (\mathbb{L}_\Delta(\delta), (\mathbb{L}_A(a), *)) \in \llbracket w(x').P' \rrbracket \wedge (\mathbb{L}_\Gamma(\gamma), (\mathbb{L}_{A^\perp}(a), *)) \in \llbracket z(x').Q' \rrbracket & \quad \text{(by Figure 5)} \\
\Leftrightarrow (\mathbb{L}_\Delta(\delta), (\mathbb{L}_{A^\perp}(a), *), *) \in \llbracket (\nu w)(w(x').P' \mid \mathcal{S}_{z,w}^A) \rrbracket & \\
\quad \wedge (\mathbb{L}_\Gamma(\gamma), (\mathbb{L}_{A^\perp}(a), *)) \in \llbracket z(x').Q' \rrbracket & \quad \text{(by Lemma 3.9)} \\
\Leftrightarrow (\mathbb{L}_\Delta(\delta), \mathbb{L}_\Gamma(\gamma), *) \in \llbracket (\nu w)(w(x').P' \mid (\nu z)(z(x').Q' \mid \mathcal{S}_{z,w}^A)) \rrbracket & \quad \text{(by Figure 5)}
\end{aligned}$$

The next result, Theorem 3.11, states that the lifting of Laurent’s transformation $(-)^{\bullet \perp}$ to the level of denotations is correct.

► **Theorem 3.11.** *Let $P \vdash^c \Gamma$ be a CP_0 process. Then $\mathbb{L}_\Gamma^*(\llbracket P \vdash^c \Gamma \rrbracket) = \llbracket (P)^{\bullet \perp} \vdash^c \Gamma^{\bullet \perp}, w : \mathbf{1} \rrbracket$.*

$$\frac{C \Downarrow \theta \quad C' \Downarrow \gamma}{C \mid C' \Downarrow (\sigma, \gamma)} \text{OBSMIX} \quad \frac{C_1 \vdash_{\text{cfg}} \Gamma_1 \mid \Sigma_1 \quad C_2 \vdash_{\text{cfg}} \Gamma_2 \mid \Sigma_2}{C_1 \mid C_2 \vdash_{\text{cfg}} \Gamma_1, \Gamma_2 \mid \Sigma_1, \Sigma_2} \text{CFG MIX}$$

$$\llbracket P \mid Q \vdash^c \Gamma, \Delta \rrbracket = \{(\gamma, \delta) \mid \gamma \in \llbracket P \vdash^c \Gamma \rrbracket, \delta \in \llbracket Q \vdash^c \Delta \rrbracket\}$$

■ **Figure 8** Extensions concerning MIX₂.

Proof sketch. By induction on the structure of $P \vdash^c \Gamma$, with a case analysis in the last rule applied. We give a representative case. Consider $!x(y).P \vdash^c ?\Delta, x : !A$, with $\Delta = x_1 : A_1, \dots, x_n : A_n$. In one direction, we apply Lemma 3.8 and Definition 3.6 to show

$$\begin{aligned} & \mathbb{L}_{? \Delta, x : !A}^* (\llbracket !x(y).P \vdash^c ?\Delta, x : !A \rrbracket) \\ &= \{ \mathbb{L}_{? \Delta, x : !A}^* (\uplus_{j=1}^k \alpha_j^1, \dots, \uplus_{j=1}^k \alpha_j^n, \langle a_1, \dots, a_k \rangle) \mid \\ & \quad \forall i \in \{1, \dots, k\}. (\alpha_i^1, \dots, \alpha_i^n, a_i) \in \llbracket P \vdash^c ?\Delta, y : A \rrbracket \} \\ &= \{ (\uplus_{j=1}^k \beta_j^1, \dots, \uplus_{j=1}^k \beta_j^n, \langle \langle b_1, * \rangle, \dots, \langle b_k, * \rangle \rangle, *, *) \mid \\ & \quad \forall i \in \{1, \dots, k\}. (\alpha_i^1, \dots, \alpha_i^n, a_i, *) \in \llbracket P \vdash^c ?\Delta, y : A \rrbracket \} \end{aligned}$$

with $\mathbb{L}_{A_i}(\alpha^i) = \beta^i$,
(and $\mathbb{L}_A(a) = b$)

In the other direction, by the I.H., we obtain:

$$\begin{aligned} & \llbracket (!x(y).P)^\bullet \vdash^c (?\Delta)^{\bullet\perp}, m : (!A)^{\bullet\perp}, w : \mathbf{1} \rrbracket \\ &= \{ (\uplus_{j=1}^k \beta_j^1, \dots, \uplus_{j=1}^k \beta_j^n, \langle \langle b_1, * \rangle, \dots, \langle b_k, * \rangle \rangle, *, *) \mid \\ & \quad \forall i \in \{1, \dots, k\}. (\alpha_i^1, \dots, \alpha_i^n, a_i, *) \in \llbracket P \vdash^c ?\Delta, y : A \rrbracket \} \end{aligned}$$

when the last rule is cut, we rely on I.H. and Lemma 3.10. ◀

By combining Theorems 2.2 and 3.11 and Lemma 3.7, we obtain our first full abstraction result:

► **Corollary 3.12** (Full Abstraction (I)). *Suppose $P, Q \vdash^c \Delta$. Then $P \simeq Q$ iff $(P)^\bullet \simeq (Q)^\bullet$.*

Proof. We have the following equivalences:

$$\begin{aligned} P \simeq Q & \Leftrightarrow \llbracket P \vdash^c \Delta \rrbracket = \llbracket Q \vdash^c \Delta \rrbracket && \text{(by Theorem 2.2)} \\ & \Leftrightarrow \mathbb{L}_\Delta^* (\llbracket P \vdash^c \Delta \rrbracket) = \mathbb{L}_\Delta^* (\llbracket Q \vdash^c \Delta \rrbracket) && \text{(by Lemma 3.7)} \\ & \Leftrightarrow \llbracket (P)^\bullet \vdash^c \Delta^{\bullet\perp}, w : \mathbf{1} \rrbracket = \llbracket (Q)^\bullet \vdash^c \Delta^{\bullet\perp}, w : \mathbf{1} \rrbracket && \text{(by Theorem 3.11)} \\ & \Leftrightarrow (P)^\bullet \simeq (Q)^\bullet && \text{(by Theorem 2.2)} \end{aligned}$$

4 An Operational Characterization of Laurent's Translation

In this section, we show that the translation $(-)^{\bullet}$ can be *internalized* as an evaluation context. That is, given $P \vdash^c \Delta$, we can define a corresponding *transformer context*, denoted $\widehat{\mathbb{T}}_\Delta[-]$. Using this context, we obtain a process denoted $\widehat{\mathbb{T}}_\Delta[P]$, in which the behavior of P is adapted following Δ , so as to produce $\Delta^{\bullet\perp}, w : \mathbf{1}$. This is clearly different from translating P into $(P)^\bullet$ by examining its structure. We shall show that $\widehat{\mathbb{T}}_\Delta[P]$ is equivalent to $(P)^\bullet$ (Corollary 4.10). As in § 3, we will also show a full-abstraction result for $\widehat{\mathbb{T}}_\Delta[-]$ (Corollary 4.15).

$$\begin{aligned}
\mathbb{T}_{x,x'}^\perp &= [x \leftrightarrow y] \vdash^c x : \mathbf{1}, x' : \perp \\
\mathbb{T}_{x,x'}^{\mathbf{1}} &= x'[y].([y \leftrightarrow x] \mid x'().\mathbf{0}) \vdash^c x : \perp, x' : \mathbf{1} \otimes \perp \\
\mathbb{T}_{x,z}^{A \otimes B} &= x(y).z[z_2].(z_1[z_2].(z_1(y).(\mathbb{T}_{y,y'}^A \mid z_1[]) \mid z_2(x').(\mathbb{T}_{x,x'}^B \mid z_2[])) \mid z().\mathbf{0}) \vdash^c \Delta \\
&\quad (\text{with } \Delta = x : A^\perp \wp B^\perp, z : (A \otimes B)^{\bullet\perp}) \\
\mathbb{T}_{x,x'}^{A \wp B} &= x'(y').x[y].(\mathbb{T}_{y,y'}^A \mid \mathbb{T}_{x,x'}^B) \vdash^c x : A^\perp \otimes B^\perp, x' : A \wp B^{\bullet\perp} \\
\mathbb{T}_{x,x'}^{!A} &= x'[w].(!w'(w).?x[y].w(y').(\mathbb{T}_{y,y'}^A \mid w[])) \mid x'().\mathbf{0} \vdash^c x : ?(A^\perp), x' : (!A)^{\bullet\perp} \\
\mathbb{T}_{x,x'}^{?A} &= !x(y).?x'[m].m[z].(z(y').(\mathbb{T}_{y,y'}^A \mid z[])) \mid m().\mathbf{0} \vdash^c x : !(A^\perp), x' : (?A)^{\bullet\perp} \\
\mathbb{T}_{x,x'}^{A_1 \& A_2} &= x'.\text{case}(x[1].\mathbb{T}_{x,x'}^{A_1}, x[2].\mathbb{T}_{x,x'}^{A_2}) \vdash^c x : A_1^\perp \oplus A_2^\perp, x' : (A_1 \& A_2)^{\bullet\perp} \\
\mathbb{T}_{x,x'}^{A_1 \oplus A_2} &= y.\text{case}(P_1, P_2) \vdash^c x : A_1^\perp \& A_2^\perp, x' : (A_1 \oplus A_2)^{\bullet\perp} \\
&\quad (\text{with } P_i = m[w].(w[i].w(y').(\mathbb{T}_{x,x'}^{A_i} \mid w[])) \mid m().\mathbf{0})
\end{aligned}$$

■ **Figure 9** Transformer processes (Definition 4.3).

This strategy works in presence of Rule MIX_2 (cf. Figure 2). Hence, in this section we work with typed processes in CP_{02} . Accordingly, we extend the denotational semantics (cf. § 2) as given in Figure 8. It is easy to check that soundness and completeness (Theorem 2.2 and Corollary 2.3) still hold for CP_{02} . In CP_{02} , additional observational equivalences arise from permutation of Rule MIX_2 with other rules:

► **Lemma 4.1.** *Given $P, Q, R \in \text{CP}_{02}$, we have: $x.\text{case}(P, Q) \mid R \simeq x.\text{case}(P \mid R, Q \mid R)$, $(\nu x)(P \mid Q) \mid R \simeq (\nu x)(P \mid (Q \mid R)) \simeq (\nu x)((P \mid R) \mid Q)$ and $x[y].(P \mid (Q \mid R)) \simeq x[y].(P \mid Q) \mid R$.*

We also need to extend $(-)^{\bullet}$ (Definition 3.3). Given processes $P \vdash^c \Delta$ and $Q \vdash^c \Gamma$ (with their translations $(P)^{\bullet} \vdash^c \Delta^{\bullet\perp}, y : \mathbf{1}$ and $(Q)^{\bullet} \vdash^c \Gamma^{\bullet\perp}, x : \mathbf{1}$, respectively), we define:

$$(P \mid Q)^{\bullet} = (\nu x)(x[y].((P)^{\bullet} \mid (Q)^{\bullet}) \mid M_x)$$

where $M_x = x(y).y().[x \leftrightarrow m]$. It is easy to check that $M_x \vdash^c x : \perp \wp \perp, m : \mathbf{1}$.

► **Remark 4.2.** The results about $\mathbb{L}^*(-)$ in § 3 can be adapted to CP_{02} .

Transformers. We define *transformer processes*, which adapt the behavior of one session on a given name.

► **Definition 4.3 (Transformers).** *Given a type A in CP_{02} , we define the transformer process $\mathbb{T}_{x,x'}^A \vdash^c x : A^\perp, x' : A^{\bullet\perp}$ by induction on the type A as in Figure 9. With a slight abuse of notation, in the figure we write $\mathbb{T}_{x,x'}^A = P \vdash^c \Delta$ to express that $\mathbb{T}_{x,x'}^A = P$ with $P \vdash^c \Delta$.*

We now define *transformer contexts*, which adapt an entire context Δ using transformer processes. We first define *typed contexts*.

Typed Process Contexts. A typed context is a typed process with a typed hole. We write $K[\square] \vdash_k^c \Delta \parallel \Gamma$ for a typed context which contains a typed hole \square , and which produces a process of type Γ . That is, given a process $P \vdash^c \Delta$, we can fill $K[\square]$ as $K[P] \vdash^c \Gamma$, by

$$\frac{K[\Box] \vdash_k^c \Sigma \parallel \Gamma, x : A \quad Q \vdash^c \Delta, x : A^\perp}{(\nu x)(K[\Box] \mid Q) \vdash_k^c \Sigma \parallel \Gamma, \Delta} \text{KCUT}_1$$

$$\frac{K[\Box] \vdash_k^c \Sigma \parallel \Gamma \quad P \vdash^c \Delta}{K[\Box] \mid P \vdash_k^c \Sigma \parallel \Gamma, \Delta} \text{KMIX} \quad \frac{}{\Box \vdash_k^c \Delta \parallel \Delta} \text{KHOLE}$$

■ **Figure 10** Typed Contexts.

replacing the unique occurrence of \Box with P . Figure 10 gives the rules for forming typed contexts. As an example, consider the derivation for a parallel context $(\nu x)(\Box \mid \mathbb{T}_{x,y}^A)$:

$$\frac{\Box \vdash_k^c x : A \parallel x : A \quad \mathbb{T}_{x,y}^A \vdash^c x : A^\perp, y : A^{\bullet\perp}}{(\nu x)(\Box \mid \mathbb{T}_{x,y}^A) \vdash_k^c x : A \parallel y : A^{\bullet\perp}}$$

Above, we can replace the use of Rule KHole with a typing derivation for $P \vdash^c x : A$, thus obtaining $(\nu x)(P \mid \mathbb{T}_{x,y}^A) \vdash^c y : A^{\bullet\perp}$. Such “filling” of contexts can be done in general:

► **Lemma 4.4.** *Given $K[\Box] \vdash_k^c \Delta \parallel \Gamma$ and $P \vdash^c \Delta$, we have that $K[P] \vdash^c \Gamma$ is derivable.*

Transformer contexts. As we have seen, contexts in our setting are hardly arbitrary: only type-compatible processes are inserted into holes. Based on this observation, and following the typing rules, we define *transformer contexts* and *transformer contexts with closing name*:

- **Definition 4.5.** *Let $\Delta = x_1 : A_1, \dots, x_n : A_n$ be a typing context. We define:*
- *transformer contexts:* $\mathbb{T}_\Delta[\Box] = (\nu x_n)(\dots(\nu x_1)(\Box \mid \mathbb{T}_{x_1, y_1}^{A_1}) \mid \dots) \mid \mathbb{T}_{x_n, y_n}^{A_n}$
 - *transformer contexts with closing name (z is fresh wrt Δ):* $\widehat{\mathbb{T}}_\Delta[\Box] = \mathbb{T}_\Delta[\Box] \mid z[]$.

Note that by Remark 2.4 the order of the cuts in $\mathbb{T}_\Delta[-]$ does not matter.

We will show that transformers are correct: the transformed process $\widehat{\mathbb{T}}_\Delta[P]$ is equivalent to the translated process $(P)^\bullet$ (Lemma 4.9). We need auxiliary results about transformers.

► **Lemma 4.6.** *The following observational equivalences hold:*

- $x'(y').\widehat{\mathbb{T}}_{\Delta, y: A, x: B}[P] \simeq \widehat{\mathbb{T}}_{\Delta, x: A \otimes B}[x(y).P]$
- $x'[w].(!w'(w).w(y').\widehat{\mathbb{T}}_{\Delta, y: A}[P] \mid x'().\mathbf{0}) \mid n[] \simeq \widehat{\mathbb{T}}_{\Delta, x: !A}[!x(y).P]$
- $?x[m].m[z].(z(y).\widehat{\mathbb{T}}_{\Delta, y: A}[P] \mid m().\mathbf{0}) \simeq \widehat{\mathbb{T}}_{\Delta, x: ?A}[?x[y].P]$
- $m[w].(w[i].w(y).\widehat{\mathbb{T}}_{\Delta, y: A}[P] \mid m().\mathbf{0}) \mid n[] \simeq \widehat{\mathbb{T}}_{y: A_1 \oplus A_2}[y[i].P]$
- $y'.\text{case}(\widehat{\mathbb{T}}_{\Delta, y: A_1}[P_1], \widehat{\mathbb{T}}_{\Delta, y: A_2}[P_2]) \simeq \widehat{\mathbb{T}}_{\Delta, y: A_1 \& A_2}[y.\text{case}(P_1, P_2)]$
- $z[z_2].(z_2[z_1].(z_1(y').\widehat{\mathbb{T}}_{\Delta, y: A}[P_1] \mid z_2(x').\widehat{\mathbb{T}}_{\Gamma, x: B}[P_2]) \mid z().\mathbf{0}) \mid w[] \simeq \widehat{\mathbb{T}}_{\Delta, \Gamma, x: A \otimes B}[x[y].(P_1 \mid P_2)]$

Proof sketch. The proof follows from Remark 2.4 and Lemma 4.1. ◀

► **Lemma 4.7.** $\llbracket [x \leftrightarrow y] \vdash^c x : \mathbf{1}, y : \perp \rrbracket = \llbracket [x[] \mid y().\mathbf{0} \vdash^c x : \mathbf{1}, y : \perp] \rrbracket$

► **Lemma 4.8.** *Let $\Delta = x_1 : !A^\perp, x'_1 : ?((A \otimes \mathbf{1}) \otimes \perp)$. We have:*

$$\llbracket \mathbb{T}_{x_1, x'_1}^{?A} \vdash^c \Delta \rrbracket = \left\{ \begin{array}{l} (\uparrow a_1, \dots, a_k, \uplus_{j=1}^k \uparrow((a', *), *)) \\ \mid \forall i \in \{1, \dots, k\}. (a_i, a'_i) \in \llbracket \mathbb{T}_{y, y'}^A \vdash^c y : A^\perp, y' : A^{\bullet\perp} \rrbracket \end{array} \right\}$$

We may now establish the correctness of transformers:

► **Lemma 4.9.** *Suppose $P \vdash^c \Gamma$. Then $\llbracket (P)^\bullet \vdash^c \Gamma^{\bullet\perp}, w : \mathbf{1} \rrbracket = \llbracket \widehat{T}_\Gamma[P] \vdash^c \Gamma^{\bullet\perp}, w : \mathbf{1} \rrbracket$.*

Proof sketch. By induction on the structure of P . We consider a number of illustrative cases. If $P = !x(y).P'$, then:

$$\begin{aligned} (!x(y).P')^\bullet &= m[x].(!x(w).w(y).(P')^\bullet \mid [m \leftrightarrow n]) \\ &\simeq m[x].(!x(w).w(y).\widehat{T}_{\Delta,y:A}[P'] \mid m().\mathbf{0}) \mid n[] && \text{(by I.H. and Lemma 4.7)} \\ &\simeq \widehat{T}_{\Delta,x:!(A)}[P'] && \text{(by Lemma 4.6)} \end{aligned}$$

If $P = (\nu x)(R \mid Q)$, then we need to show:

$$(P)^\bullet \simeq (\nu z)(z(y).\widehat{T}_{\Gamma,x:A^\perp}[Q] \mid (\nu w)(w(x').\widehat{T}_{\Delta,x:A}[R] \mid \mathcal{S}_{z,w}^A)) \simeq \widehat{T}_{\Gamma,\Delta}[(\nu x)(R \mid Q)]$$

By I.H. and Theorem 3.11 we know that:

$$\begin{aligned} (\mathbb{L}_\Delta(\delta), \mathbb{L}_\Gamma(\gamma), *) &\in \llbracket (\nu z)(z(y).\widehat{T}_{\Gamma,x:A^\perp}[Q] \mid (\nu w)(w(x').\widehat{T}_{\Delta,x:A}[R] \mid \mathcal{S}_{z,w}^A)) \rrbracket \\ &\Leftrightarrow (\delta, \gamma) \in \llbracket (\nu x)(R \mid Q) \rrbracket \end{aligned}$$

Thus, the observations on the left-hand side are exactly those from $(\nu x)(R \mid Q)$ under some transformation; that transformation being the one induced by the transformers, having thus the same observation as $\widehat{T}_{\Gamma,\Delta}[(\nu x)(R \mid Q)]$. When the last rule applied is either W or C, we rely on the I.H. and Lemma 4.8. ◀

► **Corollary 4.10.** *Given $P \vdash^c \Delta$, then $(P)^\bullet \simeq \widehat{T}_\Delta[P]$.*

Proof. The proof follows from Corollary 2.3 and Lemma 4.9. ◀

Transformer contexts and $\mathbb{L}_\Delta^*(-)$. Transformer contexts induce a function on denotations, similar to $\mathbb{L}_\Delta^*(-)$ (Definition 3.6). In general, we have the following result, for any context.

► **Definition 4.11.** *For $K[\square] \vdash_k^c \Delta \parallel \Gamma$, we define $\llbracket K[\square] \rrbracket : \mathcal{P}(\llbracket \Delta \rrbracket) \mapsto \mathcal{P}(\llbracket \Gamma \rrbracket)$ inductively:*

$$\begin{aligned} \llbracket \square \vdash_k^c \Delta \parallel \Delta \rrbracket(X) &= X \\ \llbracket (\nu x)(K[\square] \mid Q) \vdash_k^c \Delta \parallel \Sigma, \Gamma \rrbracket(X) &= \left\{ (\sigma, \gamma) \mid \begin{array}{l} (\sigma, a) \in \llbracket K[\square] \vdash_k^c \Delta \parallel \Sigma, x : A \rrbracket(X), \\ (\gamma, a) \in \llbracket Q \vdash^c \Gamma, x : A^\perp \rrbracket \end{array} \right\} \\ \llbracket K[\square] \mid Q \vdash_k^c \Delta \parallel \Sigma, \Gamma \rrbracket(X) &= \{(\sigma, \gamma) \mid \sigma \in \llbracket K[\square] \vdash_k^c \Delta \parallel \Sigma \rrbracket(X), \gamma \in \llbracket Q \vdash^c \Gamma \rrbracket\} \end{aligned}$$

► **Lemma 4.12.** *Let $K[\square] \vdash_k^c \Delta \parallel \Gamma$ and $P \vdash^c \Delta$ be a typed context and process, respectively. Then $\llbracket K[\square] \vdash_k^c \Delta \parallel \Gamma \rrbracket(\llbracket P \vdash^c \Delta \rrbracket) = \llbracket K[P] \vdash^c \Gamma \rrbracket$.*

Definition 4.11 can be specialized to transformer contexts, so as to obtain the following function: $\llbracket \widehat{T}_\Delta[\square] \rrbracket : \mathcal{P}(\llbracket \Delta \rrbracket) \rightarrow \mathcal{P}(\llbracket \Delta^{\bullet\perp}, w : \mathbf{1} \rrbracket)$. Putting all these elements together, we can show that transformers also internalize Laurent's translation on the level of denotations.

► **Lemma 4.13.** *For any type $A \in CP_{02}$, and for any $a \in \llbracket A \rrbracket$, $b \in \llbracket A^{\bullet\perp} \rrbracket$, we have*

$$(a, b) \in \llbracket T_{x,y}^A \rrbracket \iff \mathbb{L}_A(a) = b$$

Proof. By induction on the type A . ◀

► **Theorem 4.14.** *For any typing context Δ , and for any set $X \subseteq \llbracket \Delta \rrbracket$,*

$$\llbracket \widehat{T}_\Delta[\square] \vdash_k^c \Delta \parallel \Delta^{\bullet\perp}, w : \mathbf{1} \rrbracket(X) = \mathbb{L}_\Delta^*(X)$$

Proof. Let $\Delta = x_1 : A_1, \dots, x_n : A_n$. Then, $\llbracket \Delta^{\bullet\perp}, w : \mathbf{1} \rrbracket = \llbracket A_1^{\bullet\perp} \rrbracket \times \dots \times \llbracket A_n^{\bullet\perp} \rrbracket \times \{*\}$. Then, we reason as follows:

$$\begin{aligned}
& (\delta_1, \dots, \delta_n, *) \in \llbracket \widehat{\mathsf{T}}_{\Delta}[\square] \rrbracket(X) \\
\iff & (\delta_1, \dots, \delta_n) \in \llbracket \mathsf{T}_{\Delta}[\square] \rrbracket(X) = \llbracket (\nu x_n)(\dots(\nu x_1)(\square \mid \mathsf{T}_{x_1, x'_1}^{A_1} \mid \dots) \mid \mathsf{T}_{x_n, x'_n}^{A_n}) \rrbracket(X) \\
\iff & \exists d_n \in \llbracket A_n \rrbracket. (d_n, \delta_n) \in \llbracket \mathsf{T}_{x_n, x'_n}^{A_n} \rrbracket \wedge \\
& (\delta_1, \dots, \delta_{n-1}, d_n) \in \llbracket (\nu x_{n-1})(\dots(\nu x_1)(\square \mid \mathsf{T}_{x_1, x'_1}^{A_1} \mid \dots)) \rrbracket(X) \\
\iff & \exists d_n \in \llbracket A_n \rrbracket, \dots, d_1 \in \llbracket A_1 \rrbracket. (d_n, \delta_n) \in \llbracket \mathsf{T}_{x_n, x'_n}^{A_n} \rrbracket \wedge \dots \wedge (d_1, \delta_1) \in \llbracket \mathsf{T}_{x_1, x'_1}^{A_1} \rrbracket \wedge \\
& (d_1, \dots, d_n) \in \llbracket \square \rrbracket(X) \\
\iff & \exists (d_1, \dots, d_n) \in X. \mathbb{L}_{A_1}(d_1) = \delta_1 \wedge \dots \wedge \mathbb{L}_{A_n}(d_n) = \delta_n \\
\iff & (\delta_1, \dots, \delta_n, *) \in \mathbb{L}_{\Delta}^*(X) \quad \blacktriangleleft
\end{aligned}$$

Thus, Theorem 4.14 shows that $\widehat{\mathsf{T}}_{\Delta}[\square]$ is the proper internalization of Laurent’s translation as a typed context in CP_{02} . In § 3 we have shown that Laurent’s translation preserves and reflects equivalence of processes. Theorem 4.14 allows us to lift that result to processes with transformers, thus also obtaining a full abstraction result:

► **Corollary 4.15** (Full Abstraction (II)). *For all $P \in \mathsf{CP}_{02}$,*

$$P \simeq Q \vdash^c \Delta \iff \widehat{\mathsf{T}}_{\Delta}[P] \simeq \widehat{\mathsf{T}}_{\Delta}[Q] \vdash^c \Delta^{\bullet\perp}, w : \mathbf{1}$$

Proof. Follows from the soundness of denotational semantics, Lemmata 3.7 and 4.12 and Theorem 4.14. ◀

5 Concluding Remarks

This paper has brought the translation $(-)^{\bullet} : \mathsf{CLL} \rightarrow \mathsf{ILL}$ (due to Laurent [16]), into the realm of concurrent interpretations of linear logic (“propositions-as-sessions”). As we have seen, under the “propositions-as-sessions” interpretation, the translation converts a classical process P into an intuitionistic process $(P)^{\bullet}$ (cf. Definition 3.3); then, exploiting the fact that $(P)^{\bullet}$ can be analyzed without changes in the classical setting, we contrast the behavior of P and $(P)^{\bullet}$ using Atkey’s observational semantics for CP [1]. Our two full abstraction results (Corollary 3.12 and Corollary 4.15) give denotational and operational characterizations that extend the scope of Laurent’s translation, and connect purely logical results with their corresponding computational interpretations. To our knowledge, ours is the first formal relationship of its kind.

Differences between classical and intuitionistic variants of “propositions-as-sessions” have already been observed by Caires and Pfenning [8] and by Wadler [24]. There are superficial differences, such as the nature/reading of typing judgments (already discussed) and the number of typing rules – classical interpretations have one rule per connective, whereas intuitionistic ones have two: one for expressing the reliance on a behavior, another for expressing an offer. But there are also more subtle differences, in particular the *locality* principle, which, informally speaking, ensures that received names can only be used for sending. Intuitionistic interpretations enforce locality for shared names. Consider, e.g., the process $P = x(y).!y(z).Q$, which uses the name y received on x to define a server behavior. Because P does not respect locality, it is not typable in the intuitionistic system of [8].

Prior work by Van den Heuvel and Pérez [22, 23] studies this specific difference: they study the sets of processes typable under classical and intuitionistic interpretations, and use non-local processes such as P to prove that the intuitionistic set is strictly included

in the classical one. Crucially, this prior work focuses on typing, and does not formally relate the behavioral equivalences in the two classes, as we achieve here by coupling $(-)^{\bullet}$ with typed observational equivalences on processes. In fact, our results go beyond [22, 23] in that Definition 3.3 stipulates how to translate a process P with non-local servers into a corresponding process $(P)^{\bullet}$ with localized servers. This translation not only follows directly the logical translation by Laurent, but is also correct in a strong sense under the two different perspectives (denotational and operational) given by our full abstraction results.

The issue of translating non-local processes into local processes was studied, albeit in a different setting, by Boreale [5], who considers a calculus with locality as an intermediate language between the asynchronous π -calculus and the internal π -calculus. His work makes heavy use of link processes, which are closely related to the forwarding process $[x \leftrightarrow y]$ of CP. More fundamentally, because Boreale’s translations and results are framed in the untyped, asynchronous setting, comparisons with our work in the typed setting are difficult to draw.

We find it remarkable that our results leverage two separate, well-established technical ingredients, namely Laurent’s translation and Atkey’s observational equivalence and denotational semantics [1]. In particular, Atkey’s denotational semantics, based on the relational semantics of CLL, is simple and effective for our purposes, and also amenable to extensions (like incorporating support for Mix_2). Indeed, our denotational characterization $\mathbb{L}_A(-)$ of Laurent’s translation (Definition 3.6) benefits from this simplicity.

Our technical results make use of the mix principles – we use Rule Mix_0 in § 3 and also and Rule Mix_2 in § 4. The use of mix principles in the context of “propositions-as-sessions” has been analyzed by Atkey et al. [2]. Already, one difference between Wadler’s presentation in [24] and Atkey’s observational semantics in [1] is the use of Rule Mix_0 . As we have briefly mentioned, our results in § 3 hold also for CP_{02} , the extension with both Mix_0 and Mix_2 .

Finally, we note that Caires and Pfenning based their interpretation on Barber’s Dual Intuitionistic Linear Logic (DILL) [4], which is based on sequents of the form $\Gamma; \Delta \vdash^d A$, where Γ and Δ specify unrestricted and linear assignments, respectively. This is a bit different from ILL as considered by Laurent. However, the two systems are equivalent (as logics), and so this difference does not jeopardize our results. Barber [4] provides translations between DILL and ILL and shows that ILL is isomorphic to the sub-system of DILL with sequents of the form $\cdot; \Delta \vdash^d A$. From the point of view of $(-)^{\bullet}$, this means that $\vdash^c \Delta$ is provable in CLL iff $\cdot; \Delta^{\bullet} \vdash^d \mathbf{1}$ is provable in DILL. Hence we can regard $(P)^{\bullet}$ as a DILL process. This observation, together with the fact that $(P)^{\bullet}$ is typable with both $\Delta^{\bullet} \vdash^i \mathbf{1}$ and $\vdash^c \Delta^{\bullet\perp}, \mathbf{1}$, provides us with a solid groundwork for the computational interpretation of the translation.

Future Work. We intend to adapt our approach to other denotational semantics for typed languages under “propositions-as-sessions”, such as the one by Kokke et al. [15], whose definition is inspired by Brzozowski derivatives and includes the polarity of names/channels. Also, we plan to study the potential of our full abstraction results as a tool for a better understanding of the locality principle for shared names in the session-typed setting. Moreover, it would be worthwhile exploring the consequences of varying the parameter R in Laurent’s translation, which we currently instantiate with the simplest possible proposition/type.

From a more applied perspective, we believe that our work can shed light on connections between different existing implementation strategies for process calculi with session types based on linear logic. On the intuitionistic side, the work by Pfenning and Griffith develops SILL, a language based on ILL [17]; on the classical side, recent work by Caires and Toninho develops a Session Abstract Machine based on CLL [10]. It would be interesting to establish to what extent our work can be applied to connect such language implementations.

References

- 1 Robert Atkey. Observed communication semantics for classical processes. In Hongseok Yang, editor, *Programming Languages and Systems*, pages 56–82, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg. doi:10.1007/978-3-662-54434-1_3.
- 2 Robert Atkey, Sam Lindley, and J. Garrett Morris. *Conflation Confers Concurrency*, pages 32–55. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-30936-1_2.
- 3 Stephanie Balzer and Frank Pfenning. Manifest sharing with session types. *Proc. ACM Program. Lang.*, 1(ICFP):37:1–37:29, 2017. doi:10.1145/3110281.
- 4 Andrew Barber. Dual intuitionistic linear logic, 1996. Technical report, available at <https://www.lfcs.inf.ed.ac.uk/reports/96/ECS-LFCS-96-347/>.
- 5 Michele Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195(2):205–226, 1998. Concurrency Theory. doi:10.1016/S0304-3975(97)00220-X.
- 6 Luís Caires and Jorge A. Pérez. Linearity, control effects, and behavioral types. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 229–259. Springer, 2017. doi:10.1007/978-3-662-54434-1_9.
- 7 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Domain-aware session types. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 39:1–39:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.39.
- 8 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010. doi:10.1007/978-3-642-15375-4_16.
- 9 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Math. Struct. Comput. Sci.*, 26(3):367–423, 2016. doi:10.1017/S0960129514000218.
- 10 Luís Caires and Bernardo Toninho. The session abstract machine. In Stephanie Weirich, editor, *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, volume 14576 of *Lecture Notes in Computer Science*, pages 206–235. Springer, 2024. doi:10.1007/978-3-031-57262-3_9.
- 11 Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Department of Computer Science, Carnegie Mellon University, November 2003. doi:10.1184/R1/6587498.v1.
- 12 Ornela Dardha and Simon J. Gay. A new linear logic for deadlock-free session-typed processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 91–109. Springer, 2018. doi:10.1007/978-3-319-89366-2_5.
- 13 Farzaneh Derakhshan, Stephanie Balzer, and Limin Jia. Session logical relations for noninterference. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470654.

- 14 Simon Fowler, Wen Kokke, Ornela Dardha, Sam Lindley, and J. Garrett Morris. Separating sessions smoothly. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 36:1–36:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.36.
- 15 Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Better late than never: a fully-abstract semantics for classical processes. *Proc. ACM Program. Lang.*, 3(POPL):24:1–24:29, 2019. doi:10.1145/3290337.
- 16 Olivier Laurent. Around classical and intuitionistic linear logics. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '18*, pages 629–638, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209132.
- 17 Frank Pfenning and Dennis Griffith. Polarized substructural session types. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015. doi:10.1007/978-3-662-46678-0_1.
- 18 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014. doi:10.1016/j.ic.2014.08.001.
- 19 Zesen Qian, G. A. Kavvos, and Lars Birkedal. Client-server sessions in linear logic. *Proc. ACM Program. Lang.*, 5(ICFP):1–31, 2021. doi:10.1145/3473567.
- 20 Harold Schellinx. Some Syntactical Observations on Linear Logic. *Journal of Logic and Computation*, 1(4):537–559, September 1991. doi:10.1093/logcom/1.4.537.
- 21 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In Peter Schneider-Kamp and Michael Hanus, editors, *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 161–172. ACM, 2011. doi:10.1145/2003476.2003499.
- 22 Bas van den Heuvel and Jorge A. Pérez. Session type systems based on linear logic: Classical versus intuitionistic. In Stephanie Balzer and Luca Padovani, editors, *Proceedings of the 12th International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES@ETAPS 2020, Dublin, Ireland, 26th April 2020*, volume 314 of *EPTCS*, pages 1–11, 2020. doi:10.4204/EPTCS.314.1.
- 23 Bas van den Heuvel and Jorge A. Pérez. Comparing session type systems derived from linear logic. *CoRR*, abs/2401.14763, 2024. doi:10.48550/arXiv.2401.14763.
- 24 Philip Wadler. Propositions as sessions. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming, ICFP '12*, pages 273–286, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2364527.2364568.
- 25 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi:10.1017/S095679681400001X.

Bi-Reachability in Petri Nets with Data

Łukasz Kamiński 

University of Warsaw, Poland

Sławomir Lasota 

University of Warsaw, Poland

Abstract

We investigate Petri nets with data, an extension of plain Petri nets where tokens carry values from an infinite data domain, and executability of transitions is conditioned by equalities between data values. We provide a decision procedure for the bi-reachability problem: given a Petri net and its two configurations, we ask if each of the configurations is reachable from the other. This pushes forward the decidability borderline, as the bi-reachability problem subsumes the coverability problem (which is known to be decidable) and is subsumed by the reachability problem (whose decidability status is unknown).

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Petri nets, Petri nets with data, reachability, bi-reachability, reversible reachability, mutual reachability, orbit-finite sets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.31

Funding *Łukasz Kamiński*: Partially supported by NCN grant 2021/41/B/ST6/00535.

Sławomir Lasota: Partially supported by the ERC grant INFYSYS, agreement no. 950398.

Acknowledgements We are grateful to Piotrek Hofman for inspiring discussions.

1 Introduction

We investigate the model of Petri nets with data, where tokens carry values from some fixed data domain, and executability of transitions is conditioned by relations between data values involved. We study Petri nets with *equality* data [20, 22, 28], i.e., a countable infinite data domain with equality as the only relation. Other data domains have been also studied, for instance Petri nets with *ordered* data [22], i.e., a countable infinite, densely and totally ordered data domain (the model subsumes Petri nets with equality data). One can also consider an abstract setting of Petri nets with an arbitrary fixed data domain [20].

As an illustrating example, consider a Petri net with equality data which has two places p_1, p_2 and two transitions t_1, t_2 , as depicted in Fig. 1. Transition t_1 outputs two tokens with arbitrary but distinct data values onto place p_1 . Transition t_2 inputs two tokens with the same data value, say a , one from p_1 and one from p_2 , and outputs three tokens: two tokens with arbitrary but equal data values b , where $b \neq a$, one onto p_1 and the other onto p_2 , plus one token with a data value $c \neq a$ onto p_1 . Note that transition t_2 does not specify whether $b = c$ or not, and therefore both options are allowed.

The most fundamental decision problem for Petri nets, the *reachability* problem, asks, given a net together with source and target configurations, if there is a run from source to target. It is well known that the reachability problem is undecidable for Petri nets with ordered data [22], while the decidability status of this problem for equality data still remains an intriguing open question. The same applies to two other major extensions of plain Petri nets, namely pushdown Petri nets [23] and branching Petri nets [9, 29]. On the other hand, the *coverability* problem (where we ask if there is a run from source to a configuration that possibly extends target by some extra tokens) is decidable for both equality and ordered



© Łukasz Kamiński and Sławomir Lasota;

licensed under Creative Commons License CC-BY 4.0

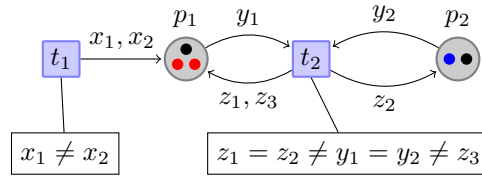
35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 31; pp. 31:1–31:20

Leibniz International Proceedings in Informatics



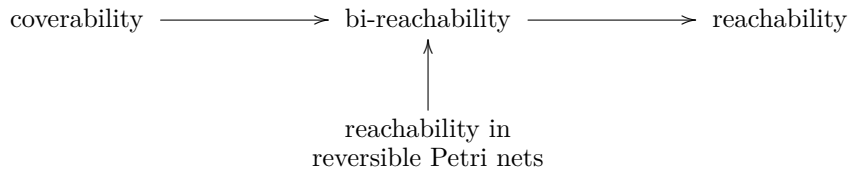
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A Petri net with equality data, with places $\{p_1, p_2\}$ and transitions $\{t_1, t_2\}$. The shown configuration engages 5 tokens, carrying 3 different data values, depicted through different colors.

data [20]. As widely known, coverability easily reduces to reachability. Furthermore, the reachability problem is decidable, also for equality and ordered data, in the special case of *reversible* Petri nets (where transitions are closed under reverse), as recently shown in [15].

In this paper we do a step towards decidability of reachability in Petri nets with equality data, and study a relevant decision problem sandwiched between reachability and the two latter decidable problems: the *bi-reachability* problem (also called *mutual reachability problem* or *reversible reachability problem* [24]). It asks, for a net and two its configurations, if each of the configurations is reachable from the other one. In other words, the problem asks if two given configurations are in the same bi-reachability equivalence class. Here are all known reductions, valid for Petri nets with either equality or ordered data, as well as for plain Petri nets (without data):



As our main result we prove decidability of this problem for equality data domain. This result pushes further the decidability border, subsuming decidability of coverability, and of reachability in reversible Petri nets with equality data. Our approach is specific to equality data, and thus we leave unresolved the status of bi-reachability in case of ordered data.

The decision procedure for bi-reachability is inspired by the classical decomposition approach used to decide reachability in plain Petri nets [18, 19, 27]. There, it is often more convenient to work with *vector addition systems with states* (VASS) instead of Petri nets [18, 27]. Following this line, for technical convenience we prefer to work with the model of *data VASS* (DVASS) [16] rather than with Petri nets. In short, our approach consists of two ingredients. First, we provide a sufficient condition for a DVASS to admit bi-reachability (resembling Θ_1 and Θ_2 conditions of [18, 27]), which is effectively testable. Second, in case the condition fails, we provide an effective way of reducing a DVASS to an equivalent one, with respect to bi-reachability, which has smaller *rank*. As ranks are well founded, the reduction step guarantees correctness and termination. Importantly, the decision procedure manipulates DVASS, and does not need to resort to manipulation of more general structures (like generalised VASS of [18, 27], or graph-transition sequences of [19], or witness graph sequences of [25], or KLM sequences of [26]). This allows us to avoid similar generalisations in the data setting, and allows to keep the algorithm relatively simple.

Our work leaves two exciting open questions: can one extend our approach to bi-reachability in case of ordered data, or to reachability in case of equality data. Clearly, if attempting to solve the latter problem, one unavoidably will be faced with some generalisation of the above-mentioned structures to the data setting.

Related research. Petri nets with equality data are a well established and widely studied model of concurrent systems, as data allow to model important aspects of such systems not captured by plain Petri nets, e.g. process identity [1, 5]. The model can be also seen as a reinterpretation of the classical definition of Petri nets with a relaxed notion of finiteness, namely orbit-finiteness, where one allows for orbit-finite sets of places and transitions instead of just finite ones; this is along the lines of [3, 4]. Similar net models have been proposed already in the early 80ies: high-level Petri nets [13] and colored Petri nets [17]. Since then, similar formalisms seem to have been rediscovered, for instance constraint multiset rewriting [6, 7, 8].

In plain Petri nets, bi-reachability is decidable as a consequence of decidability of reachability [18, 19, 27]. Later, exact EXPSPACE complexity was established in [24]. In our setting, the problem is ACKERMANN-hard due to [23]. In pushdown Petri nets, decidability of reachability in the reversible subclass has been shown only recently [12], while decidability status of bi-reachability is still open. Indeed, it is known that reachability in pushdown Petri nets with d places reduces to coverability in pushdown Petri nets with $d + 1$ places, and the latter problem reduces to bi-reachability in pushdown Petri nets. Hence, decidability of bi-reachability would imply decidability of reachability in case of pushdown Petri nets.

2 Preliminaries: orbit-finite sets and vectors

In the sequel, let \mathbb{A} denote a fixed countable infinite set of data values (called also *atoms*). By $\text{Aut}(\mathbb{A})$ we denote the set of all permutations of \mathbb{A} (called also *automorphisms*). For a subset $S \subseteq \mathbb{A}$ we define the subgroup $\text{Aut}_S(\mathbb{A}) = \{\sigma \in \text{Aut}(\mathbb{A}) \mid \sigma(s) = s \text{ for all } s \in S\}$. Permutations in $\text{Aut}_S(\mathbb{A})$ we call *S-automorphisms*.

Orbit-finite sets. In the following we study actions of the group $\text{Aut}(\mathbb{A})$ on different sets. An action of $\text{Aut}(\mathbb{A})$ on a set Z is a group homomorphism ι from $\text{Aut}(\mathbb{A})$ to functions $Z \rightarrow Z$. We write $\sigma(z)$ instead of $\iota(\sigma)(z)$ for $\sigma \in \text{Aut}(\mathbb{A})$ and $z \in Z$. In the sequel we always use the natural action of $\text{Aut}(\mathbb{A})$ that, regardless of Z , renames atoms $a \in \mathbb{A}$ but leaves other elements intact. Here are two specific examples of such action that will serve later as building blocks in the definition of our model:

► **Example 1.** Let $\perp \notin \mathbb{A}$. For any finite sets L and \mathcal{R} of *locations* and *register names*, respectively, the group $\text{Aut}(\mathbb{A})$ acts naturally on the set of *states* $Q = L \times (\mathcal{R} \rightarrow (\mathbb{A} \cup \{\perp\}))$, namely given $\sigma \in \text{Aut}(\mathbb{A})$ and $q = (\ell, \nu) \in Q$, we put

$$\sigma(q) := (\ell, \sigma(\nu)) \quad \text{where } \sigma(\nu)(r) = \begin{cases} \sigma(\nu(r)), & \text{if } \nu(r) \in \mathbb{A} \\ \perp & \text{if } \nu(r) = \perp. \end{cases}$$

Furthermore, for any finite sets H, P of plain *places* and atom *places*, respectively, $\text{Aut}(\mathbb{A})$ acts naturally on functions $H \cup P \times \mathbb{A} \rightarrow \mathbb{Z}$, namely given $\sigma \in \text{Aut}(\mathbb{A})$ and $\mathbf{v} : H \cup P \times \mathbb{A} \rightarrow \mathbb{Z}$ we put $\sigma(\mathbf{v})(h) := \mathbf{v}(h)$ for $h \in H$, and $\sigma(\mathbf{v})(p, \sigma(a)) := \mathbf{v}(p, a)$ for $p \in P$. ◻

Roughly speaking, a set is *orbit-finite* if it has a finite number of elements up to automorphisms of atoms. We define the *orbit* of an element $z \in Z$:

$$\text{ORBIT}(z) := \{\sigma(z) \mid \sigma \in \text{Aut}(\mathbb{A})\}.$$

31:4 Bi-Reachability in Petri Nets with Data

As different orbits are necessarily disjoint, Z partitions uniquely into orbits. A subset $X \subseteq Z$ is *orbit-finite* if it is a finite union of orbits. Clearly all orbits, and hence also all finite unions thereof, are closed under the action of $\text{Aut}(\mathbb{A})$. Orbit-finite sets are closed under finite unions and products [2, Lem. 3.24].

► **Example 2.** We continue Example 1. The whole set $Q = L \times (\mathcal{R} \rightarrow (\mathbb{A} \cup \{\perp\}))$ is orbit-finite, since the orbit of a state $q = (\ell, \nu) \in Q$ is determined by its location ℓ , the inverse image of \perp , namely $\{r \in \mathcal{R} \mid \nu(r) = \perp\}$, and the *equality type* of ν , namely the set $\{(r, r') \in \mathcal{R}^2 \mid \nu(r) = \nu(r') \neq \perp\}$. Indeed, for every two states $q = (\ell, \nu)$ and $q' = (\ell', \nu')$ such that $\ell = \ell'$, and ν and ν' have the same inverse image of \perp and the same equality type, there is an automorphism $\sigma \in \text{Aut}(\mathbb{A})$ such that $\sigma(q) = q'$.

On the other hand, the function space $(H \cup P \times \mathbb{A}) \rightarrow \mathbb{Z}$ is not orbit-infinite. \lrcorner

Vectors. Given a set X , by $X \rightarrow_{\text{fin}} \mathbb{Z}$ we denote the commutative group freely generated by X , and the group operation we denote by \oplus . We write $\mathbf{v} \ominus \mathbf{w}$ instead of $\mathbf{v} \oplus \mathbf{w}^{-1}$. Equivalently, $X \rightarrow_{\text{fin}} \mathbb{Z}$ can be identified with the set of all functions $\mathbf{v} : X \rightarrow \mathbb{Z}$ which map almost all elements of X to 0, i.e., those functions where the set $\{x \in X \mid \mathbf{v}(x) \neq 0\}$ is finite. Elements of $X \rightarrow_{\text{fin}} \mathbb{Z}$ we call *X-vectors*, or simply *vectors* if the generating set X is clear from the context. The zero vector we denote by $\mathbf{0}$, irrespectively of X , and a single-generator vector $x \in X$ we denote by $\mathbf{1}_x$. Seen as a function $X \rightarrow \mathbb{Z}$, the vector $\mathbf{1}_x$ maps x to 1 and all other elements of X to 0. When X is finite, we call *X-vectors* finite as well. Nonnegative vectors, denoted $X \rightarrow_{\text{fin}} \mathbb{N}$, are elements of the commutative monoid freely generated by X , or functions $\mathbf{v} : X \rightarrow \mathbb{N}$ which map almost all elements of X to 0 or, equivalently, finite multisets of elements of X . We write $\oplus W$ to denote the sum of a finite set W of vectors.

In the sequel the generating set X is most often of the form $H \cup P \times \mathbb{A}$ for some finite sets P, H . Clearly, $(H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}$ is isomorphic to $(H \rightarrow_{\text{fin}} \mathbb{Z}) \times (P \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{Z})$, as every vector $\mathbf{v} : (H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}$ decomposes uniquely as the sum $\mathbf{v} = \mathbf{u} \oplus \mathbf{w}$, where $\mathbf{u} : H \rightarrow_{\text{fin}} \mathbb{Z}$, $\mathbf{w} : P \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{Z}$. Given $\mathbf{v} : (H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}$, we define its support $\text{SUPP}(\mathbf{v}) \subseteq \mathbb{A}$, as the (necessarily finite) set of those atoms which are sent by \mathbf{v} to a nonzero value:

$$\text{SUPP}(\mathbf{v}) := \{a \in \mathbb{A} \mid \exists p \in P : \mathbf{v}(p, a) \neq 0\}.$$

Intuitively, $\text{SUPP}(\mathbf{v})$ contains those atoms that 'appear' in \mathbf{v} . We observe that $\sigma(\mathbf{v}) = \mathbf{v}$ as long as $\sigma(a) = a$ for all $a \in \text{SUPP}(\mathbf{v})$. The natural action of $\text{Aut}(\mathbb{A})$ given in Example 1 restricts to the set of vectors $(H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}$, which is still not orbit-infinite.

► **Example 3.** Transitions t_1, t_2 of Petri net in Figure 1 are semantically orbit-finite sets of $(P \times \mathbb{A})$ -vectors, where $P = \{p_1, p_2\}$ (i.e., $H = \emptyset$). Indeed, the effect of firing t_1 amounts to adding two arbitrary but different atoms $a \neq b$ to place p_1 , i.e., is described by an *X*-vector

$$\mathbf{v}_1 = (p_1, a) \oplus (p_1, b) \quad (a \neq b).$$

As the choice of atoms $a \neq b$ is arbitrary, all possible effects of firing the transition span one orbit of vectors: $V_1 = \text{ORBIT}(\mathbf{v}_1)$. The effect of firing t_2 amounts to removing some arbitrary atom a from both p_1 and p_2 , and adding two further atoms b, c not equal to a : one of them is added to both p_1 and p_2 , while the other one only to p_1 . As it is not specified whether $b = c$ or not, we describe t_2 by two *X*-vectors:

$$\mathbf{v}_2 = (p_1, c) \oplus (p_1, b) \oplus (p_2, b) \ominus (p_1, a) \ominus (p_2, a) \quad (a \neq b \neq c \neq a) \quad (1)$$

$$\mathbf{v}'_2 = (p_1, b) \oplus (p_1, b) \oplus (p_2, b) \ominus (p_1, a) \ominus (p_2, a) \quad (a \neq b). \quad (2)$$

As before, the choice of atoms is arbitrary, and hence all possible effects of firing t_2 span the union of two orbits of vectors: $V_2 = \text{ORBIT}(\mathbf{v}_2) \cup \text{ORBIT}(\mathbf{v}'_2)$. Intuitively, different orbits in T_2 correspond to different *equality types* of a tuple of atoms (a, b, c) : one defined by inequalities $a \neq b \neq c \neq a$, and another defined by $a \neq b = c$. This example illustrates a transformation of Petri nets to data VASS, the model we work with in this paper.¹ \lrcorner

Multiset sum problem. The following core decision problem, parametrised by an orbit-finite set X , will be useful later:

MULTISET SUM

Input: an orbit-finite set M of X -vectors, and an X -vector \mathbf{b} .

Question: is \mathbf{b} equal to the sum of a finite multiset of vectors from M ?

In other words, we ask if \mathbf{b} is a nonnegative integer linear combination of vectors from M . We assume that M is represented by a finite set of representatives, one per orbit.

► **Lemma 4** ([14, Thm. 17]). *MULTISET SUM is decidable.*

3 Data vector addition systems with states

Classical Petri nets are equivalent, with respect to most decision problems, to vector addition systems (VASS). Likewise, we introduce here a formalism equivalent to Petri nets with data, called data vector addition systems with states (DVASS). It is an extension of (stateless) data vector addition systems (DVAS) studied in [16].

Data VASS. A data VASS (DVASS) $\mathcal{V} = (L, \mathcal{R}, H, P, T)$ consists of pairwise disjoint finite sets of locations L , register names \mathcal{R} , plain places H , atom places P , and an *orbit-finite* set

$$T \subseteq Q \times ((H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}) \times Q$$

of *transitions*, where $Q = L \times (\mathcal{R} \rightarrow (\mathbb{A} \cup \{\perp\}))$ is the set of *states*. The set T is thus assumed to be a finite union of orbits, under the natural action of $\text{Aut}(\mathbb{A})$ on transitions that extends the action on vectors and states given in Example 1: for $t = (q, \mathbf{v}, q') \in T$ we put $\sigma(t) := (\sigma(q), \sigma(\mathbf{v}), \sigma(q'))$. In particular, T is closed under the action of $\text{Aut}(\mathbb{A})$. Given a state $q = (\ell, \nu) \in Q$, the function ν is called *register valuation*. Intuitively, $\nu(r) = a$ means that register r contains atom a , while $\nu(r) = \perp$ means that r is empty. The vector \mathbf{v} is called the *effect* of transition (q, \mathbf{v}, q') .

The model of (plain) VASS corresponds to the special case where $\mathcal{R} = \emptyset$ and $P = \emptyset$, i.e., DVASS without registers and atom places. In this case the set $T \subseteq Q \times (H \rightarrow_{\text{fin}} \mathbb{Z}) \times Q$, being orbit-finite, is necessarily finite. The model of DVAS corresponds to the special case when $L = \{*\}$ is a singleton and $\mathcal{R} = \emptyset$ and $H = \emptyset$, i.e., DVASS without locations, registers and plain places.

A *pseudo-configuration* of \mathcal{V} is a pair $(q, \mathbf{v}) \in Q \times ((H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z})$, written $q(\mathbf{v})$. A *pseudo-run* from $q_1(\mathbf{v}_0)$ to $q_k(\mathbf{v}_k)$ is a sequence of pseudo-configurations $\pi = q_0(\mathbf{v}_0) q_1(\mathbf{v}_1) \dots q_k(\mathbf{v}_k)$ such that $t_i = (q_{i-1}, \mathbf{v}_i - \mathbf{v}_{i-1}, q_i) \in T$ for every $i = 1, \dots, k$. We say that the pseudo-run π *uses* the transitions $t_1, \dots, t_k \in T$. The support of a state $q = (\ell, \nu)$ is the set of all atoms used in registers, i.e. $\text{SUPP}(q) = \nu(\mathcal{R}) \cap \mathbb{A}$. The support of a transition $t = (q, \mathbf{v}, q')$ is $\text{SUPP}(t) = \text{SUPP}(q) \cup \text{SUPP}(\mathbf{v}) \cup \text{SUPP}(q')$. We also

¹ As in a transformation from plain Petri nets to VASS, in case of Petri net with *tight loops*, i.e., transitions that simultaneously input and output the same atom from/to the same place, we would have to split every such transition into input and output part.

31:6 Bi-Reachability in Petri Nets with Data

define the support of a pseudo-run as the union of supports of all its pseudo-configurations: $\text{SUPP}(\pi) = \text{SUPP}(q_0) \cup \text{SUPP}(\mathbf{v}_0) \cup \text{SUPP}(q_1) \cup \dots \cup \text{SUPP}(q_k) \cup \text{SUPP}(\mathbf{v}_k)$. We again extend the action of $\text{Aut}(\mathbb{A})$, this time to pseudo-runs, in an expected way:

$$\sigma(q_0(\mathbf{v}_0) q_1(\mathbf{v}_1) \dots q_k(\mathbf{v}_k)) := \sigma(q_0)(\sigma(\mathbf{v}_0)) \sigma(q_1)(\sigma(\mathbf{v}_1)) \dots \sigma(q_k)(\sigma(\mathbf{v}_k)).$$

The set of pseudo-runs is closed under the action of $\text{Aut}(\mathbb{A})$.

Configurations are those pseudo-configurations $q(\mathbf{v})$ where the vector \mathbf{v} is nonnegative, i.e., $\mathbf{v} : (H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{N}$. Let $\text{CONF} = Q \times ((H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{N})$ denote the set of all configurations. A *run* is a pseudo-run where every pseudo-configuration $q_i(\mathbf{v}_i)$ is actually a configuration. We write $q(\mathbf{v}) \dashrightarrow q'(\mathbf{v}')$ (resp. $q(\mathbf{v}) \rightarrow q'(\mathbf{v}')$) if there is a pseudo-run (resp. a run) from $q(\mathbf{v})$ to $q'(\mathbf{v}')$.

► **Example 5.** Continuing Example 3, Petri net in Figure 1 is equivalent to a DVAS $\mathcal{V} = (\{*\}, \emptyset, \emptyset, \{p_1, p_2\}, T)$, whose transitions are (as $\mathcal{R} = \emptyset$, we omit register valuations)

$$T = \{*\} \times (V_1 \cup V_2) \times \{*\}.$$

The initial configuration shown in Figure 1 is $*(\mathbf{v})$, where $\mathbf{v} = (p_1, a) \oplus (p_1, c) \oplus (p_1, c) \oplus (p_2, a) \oplus (p_2, b)$ for some distinct atoms $a, b, c \in \mathbb{A}$. In order to illustrate DVASS, we drop the first inequality in the constraint on t_2 , and consider the relaxed constraint $z_1 = z_2 \wedge y_1 = y_2 \neq z_3$ instead. This adds a third orbit of possible effects of firing t_2 , when the atom b added to places p_1 and p_2 is the same as the atom a removed (c.f. (1) in Example 3):

$$\mathbf{v}_2'' = (p_1, c) \oplus (p_1, a) \oplus (p_2, a) \ominus (p_1, a) \ominus (p_2, a) \quad (a \neq c).$$

The modified Petri net is equivalent to a DVASS $\mathcal{V}' = (L, \emptyset, \emptyset, P, T')$ with two locations $L = \{\ell, \ell'\}$, still no registers, a larger set of atom places $P = \{p_1, p_2, \bar{p}\}$, and transitions:

$$T' = \{\ell\} \times (V_1 \cup V_2) \times \{\ell\} \cup \{\ell\} \times \text{ORBIT}(\mathbf{w}) \times \{\ell'\} \cup \{\ell'\} \times \text{ORBIT}(\mathbf{w}') \times \{\ell\},$$

where vectors \mathbf{w}, \mathbf{w}' are splitting \mathbf{v}_2'' into input and output part, using an auxiliary place \bar{p} to temporarily store atom a :

$$\mathbf{w} = (\bar{p}, a) \ominus (p_1, a) \ominus (p_2, a) \quad \mathbf{w}' = (p_1, c) \oplus (p_1, a) \oplus (p_2, a) \ominus (\bar{p}, a) \quad (a \neq c). \quad (3)$$

Transitions in T' corresponding to $V_1 \cup V_2$ go from ℓ to ℓ , while the other transitions go from ℓ to ℓ' , or from ℓ' to ℓ . The initial configuration of \mathcal{V}' is $\ell(\mathbf{v})$. Instead of place \bar{p} one could also use a single register $\mathcal{R} = \{r\}$, and transitions of the form

$$((\ell, \perp), \ominus(p_1, a) \ominus (p_2, a), (\ell', a)) \quad ((\ell', a), (p_1, c) \oplus (p_1, a) \oplus (p_2, a), (\ell, \perp)) \quad (a \neq c),$$

to the same effect as in (3). The initial configuration would be then $q(\mathbf{v})$, where $q = (\ell, \perp)$. ◻

► **Remark 6.** Our model of DVASS syntactically extends DVAS by locations, registers and plain places. The extended model is convenient for our decidability argument, while being equivalent to DVAS with respect to most of decision problems. Indeed, a DVASS may be transformed into an essentially equivalent DVAS in three steps (as in the proof of Lemma 12):

$$\begin{array}{ccc} \text{locations} & \xrightarrow{2} & \text{plain places} \\ \uparrow 1 & & \downarrow 3 \\ \text{registers} & \xrightarrow{1} & \text{atom places} \end{array} \quad (4)$$

First, we eliminate registers using locations and atom places, then we encode locations into plain places, and finally we encode plain places into atom ones.

State graph. We define the *state graph* $\text{GRAPH}(T) = (Q, E)$ of a DVASS \mathcal{V} . Its nodes are states Q , and its edges $E \subseteq Q \times Q$ are pairs of states related by some transition of \mathcal{V} :

$$E = \{(q, q') \in Q^2 \mid (q, \mathbf{v}, q') \in T \text{ for some vector } \mathbf{v} : (H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}\}.$$

When \mathcal{R} is non-empty, the sets of nodes and edges of $\text{GRAPH}(T)$ are infinite but orbit-finite.

Bi-reachability problem. We say that a configuration $q'(\mathbf{v}')$ is *reachable* from $q(\mathbf{v})$ if there is a run $q(\mathbf{v}) \rightarrow q'(\mathbf{v}')$. Two configurations $q(\mathbf{v}), q'(\mathbf{v}')$ are *bi-reachable* if each of them is reachable from the other: $q(\mathbf{v}) \rightarrow q'(\mathbf{v}')$ and $q(\mathbf{v}) \leftarrow q'(\mathbf{v}')$.

DVASS BI-REACHABILITY

Input: a DVASS $(L, \mathcal{R}, H, P, T)$ and two configurations, $q(\mathbf{v})$ and $q'(\mathbf{v}')$.

Question: are $q(\mathbf{v}), q'(\mathbf{v}')$ bi-reachable?

As before, we assume that the orbit-finite set T of transitions is represented by a finite set of representatives, one per orbit. As our main result we prove:

► **Theorem 7.** DVASS *bi-reachability problem is decidable.*

Since our model of DVASS includes plain places, we can assume w.l.o.g. a convenient form of source and target configuration that consists, essentially, of just a location. Let \perp denote the empty register valuation: $\perp(r) = \perp$ for every $r \in \mathcal{R}$.

► **Lemma 8.** *In DVASS BI-REACHABILITY problem we may assume, w.l.o.g., that $q = (\ell, \perp)$, $q' = (\ell', \perp)$, and $\mathbf{v} = \mathbf{v}' = \mathbf{0}$.*

4 Toolset

Our decision procedure relies on a number of existing tools. One of them is solvability of MULTISSET SUM (Lemma 4). Here we introduce two further tools: a sufficient condition for VASS reachability, and computability of coverability sets in DVAS.

Sufficient condition for VASS reachability. We recall a condition that guarantees existence of a run in a VASS. It is a simplification of the classical condition of [18, 19, 27] which guarantees existence of a run in a *generalised* VASS. Consider a VASS \mathcal{V} with plain places H . For $\mathbf{v} : H \rightarrow_{\text{fin}} \mathbb{N}$ we write $\mathbf{v} \gg \mathbf{0}$ to mean that for every $h \in H$ we have $\mathbf{v}(h) > 0$.

- Θ_1 : For every $m \in \mathbb{N}$, there is a pseudo-run $q(\mathbf{0}) \dashrightarrow q'(\mathbf{0})$ using every transition at least m times.
- Θ_2 : For some vectors $\Delta, \Delta' \gg \mathbf{0}$, there are runs: $q(\mathbf{0}) \rightarrow q(\Delta)$ and $q'(\Delta') \rightarrow q'(\mathbf{0})$.

► **Lemma 9** (Thm. 2 in [18], Prop. 1 in [21]). *For every VASS, $\Theta_1 \wedge \Theta_2$ implies $q(\mathbf{0}) \rightarrow q'(\mathbf{0})$.*

Coverability sets in DVASS. Let $\mathcal{V} = (L, \mathcal{R}, H, P, T)$ be a DVASS, and let $X = H \cup (P \times \mathbb{A})$. We define the pointwise order on nonnegative vectors $X \rightarrow_{\text{fin}} \mathbb{N}$: $\mathbf{v} \leq \mathbf{v}'$ if and only if for every $x \in X$ we have $\mathbf{v}(x) \leq \mathbf{v}'(x)$. We define a quasi-order by relaxation of \leq , up to automorphisms: $\mathbf{v} \sqsubseteq \mathbf{v}'$ if $\sigma(\mathbf{v}) \leq \mathbf{v}'$ for some $\sigma \in \text{Aut}(\mathbb{A})$. We extend the relation \sqsubseteq to configurations: for states q, q' and vectors $\mathbf{v}, \mathbf{v}' \in X \rightarrow_{\text{fin}} \mathbb{N}$ we put $q(\mathbf{v}) \sqsubseteq q'(\mathbf{v}')$ if $\sigma(q) = q'$ and $\sigma(\mathbf{v}) \leq \mathbf{v}'$ for some $\sigma \in \text{Aut}(\mathbb{A})$.

► **Lemma 10.** \sqsubseteq *is a WQO on configurations.*

The *coverability set* of a configuration $q(\mathbf{v})$ is defined as the downward closure, with respect to \sqsubseteq , of the reachability set:

$$\text{COVER}(q(\mathbf{v})) = \{\bar{s}(\bar{\mathbf{w}}) \in \text{CONF} \mid \exists s(\mathbf{w}) \in \text{CONF} : \bar{s}(\bar{\mathbf{w}}) \sqsubseteq s(\mathbf{w}) \wedge q(\mathbf{v}) \longrightarrow s(\mathbf{w})\}.$$

It is known that the coverability set is representable by a finite union of *ideals* (downward closed directed sets) [10, 11]. Let's complete \mathbb{N} with a top element, $\mathbb{N}_\omega \stackrel{\text{def}}{=} \mathbb{N} \cup \{\omega\}$, which is larger than all numbers: $n < \omega$ for all $n \in \mathbb{N}$. We consider pairs $(q, f) \in Q \times (X \rightarrow \mathbb{N}_\omega)$, written $q(f)$, and called ω -configurations. Each such pair determines a set of configurations (we extend \sqsubseteq to all ω -configurations in the expected way):

$$q(f)\downarrow := \{s(\mathbf{v}) \in \text{CONF} \mid s(\mathbf{v}) \sqsubseteq q(f)\},$$

which is downward closed (whenever $s(\mathbf{v}) \in q(f)\downarrow$ and $s'(\mathbf{v}') \sqsubseteq s(\mathbf{v})$ then $s'(\mathbf{v}') \in q(f)\downarrow$) and directed (for every two $s(\mathbf{v}), s'(\mathbf{v}') \in q(f)\downarrow$ there is $\bar{s}(\bar{\mathbf{v}}) \in q(f)\downarrow$ such that $s(\mathbf{v}) \sqsubseteq \bar{s}(\bar{\mathbf{v}})$ and $s'(\mathbf{v}') \sqsubseteq \bar{s}(\bar{\mathbf{v}})$). The set $q(f)\downarrow$ is thus an *ideal*. We call an ω -configuration $q(f)$ *simple* if for every $p \in P$, either $f(p, a) = 0$ for *almost all* $a \in \mathbb{A}$ (i.e., for all $a \in \mathbb{A}$ except finitely many), or $f(p, a) = \omega$ for almost all $a \in \mathbb{A}$. Simple ω -configurations are thus finitely representable. Ideals determined by simple ω -configurations we call simple too.

► **Example 11.** In the DVASS \mathcal{V}' in Example 5, $\text{COVER}(\ell(\mathbf{v})) = \ell(f)\downarrow \cup \ell(g)\downarrow \cup \ell'(f')\downarrow \cup \ell'(g')\downarrow$, where $f(p_1, c) = g(p_1, c) = f'(p_1, c) = g'(p_1, c) = \omega$ for every $c \in \mathbb{A}$,

$$\begin{aligned} f(p_2, a) = f(p_2, b) &= 1 & g(p_2, a) &= 2 \\ f'(p_2, a) = f'(\bar{p}, b) &= 1 & g'(p_2, a) = g'(\bar{p}, a) &= 1 \end{aligned}$$

for some $a \neq b \in \mathbb{A}$, and all other arguments are mapped by f, g, f' and g' to 0. Indeed, due to transition t_1 , place p_1 can be filled up with arbitrary many tokens with any atoms. On the other hand place p_2 has two tokens in the initial configuration $\ell(\mathbf{v})$, and hence will invariantly have, in location ℓ , two tokens whose atoms may be equal or not. Furthermore, in location ℓ' , places p_2 and \bar{p} have always one token each, with atoms equal or not. \lrcorner

Simple ω -configurations provide finite representations of simple ideals. Relying on the result of [16], the coverability set in a DVAS is a union of a finite set of simple ideals, which is computable. We lift this result to the model of DVASS:

► **Lemma 12.** *Given a DVASS and its configuration $q(\mathbf{v})$, one can compute a finite set of simple ω -configurations $\{s_1(f_1), \dots, s_n(f_n)\}$ such that $\text{COVER}(q(\mathbf{v})) = s_1(f_1)\downarrow \cup \dots \cup s_n(f_n)\downarrow$.*

Proof. Let $\mathcal{V} = (L, \mathcal{R}, H, P, T)$ be a DVASS, let $q(\mathbf{v})$ be its configuration, where $q = (\ell, \eta)$. Theorem 3.5 in [16] proves the claim in the special case of DVAS. We reduce DVASS to DVAS in three steps, as shown in the diagram (4) in Remark 6.

As the first step we get rid of registers by considering them as additional atom places that store at most one token, while keeping track, in locations, of the set of currently empty registers. We set $P_1 := P \cup \mathcal{R} \cup \bar{\mathcal{R}}$, where $\bar{\mathcal{R}} = \{\bar{r} \mid r \in \mathcal{R}\}$ is distinct a copy of \mathcal{R} , and $L_1 = (L \cup \bar{L}) \times \mathcal{P}(\mathcal{R})$, where $\bar{L} = \{\bar{\ell} \mid \ell \in L\}$, and define the new set of transitions T_1 by transforming transitions from T as follows. In the construction we identify a register valuation μ with a vector $\mu = \oplus \{(r, a) \mid \mu(r) = a \neq \perp\}$, or with a vector $\bar{\mu} = \oplus \{(\bar{r}, a) \mid \mu(r) = a \neq \perp\}$. Every transition $t = ((\ell, \mu), \mathbf{v}, (\ell', \mu')) \in T$ gives rise to a transition in T_1

$$\left((\ell, \mu^{-1}(\perp)), \mathbf{v} \oplus \mu \oplus \bar{\mu}', (\bar{\ell}', (\mu')^{-1}(\perp)) \right)$$

that starts in location $(\ell, \mu^{-1}(\perp))$, ends in a location $(\bar{\ell}', (\mu')^{-1}(\perp))$ and whose effect is \mathbf{v} plus, intuitively speaking, removing μ from places \mathcal{R} and putting μ' to places $\bar{\mathcal{R}}$. In addition, all transitions of the form

$$\left((\bar{\ell}', \nu^{-1}(\perp)), \nu \ominus \bar{\nu}, (\ell', \nu^{-1}(\perp)) \right)$$

are added to T_1 , where $\nu : \mathcal{R} \rightarrow (\mathbb{A} \cup \{\perp\})$ is any register valuation. Intuitively, these transitions flash back all tokens from places $\bar{\mathcal{R}}$ to the corresponding places \mathcal{R} . This yields a DVASS $\mathcal{V}_1 := (L_1, \emptyset, H, P_1, T_1)$ computable from \mathcal{V} , and its location $\ell_1 = (\ell, \eta^{-1}(\perp))$ corresponding to state q such that the coverability sets in \mathcal{V} (on the left) is computable from the one in \mathcal{V}_1 (on the right):

▷ **Claim 13.** $\text{COVER}(q(\mathbf{v})) = \{(\ell', \mu')(\mathbf{v}') \mid (\ell', (\mu')^{-1}(\perp))(\mathbf{v}' \oplus \mu') \in \text{COVER}(\ell_1(\mathbf{v} \oplus \eta))\}$.

As the second step, we dispose of locations L_1 by moving them to plain places. We set $H_2 := H \cup L_1 \cup \widetilde{L}_1$, where $\widetilde{L}_1 = \{\widetilde{\ell} \mid \ell \in L_1\}$, and transform each transition $t = (\ell, \mathbf{v}, \ell') \in T_1$ into a transition in T_2 :

$$(*, \mathbf{v} \ominus \ell \oplus \widetilde{\ell}', *)$$

We also add a new transition $(*, \ell \ominus \widetilde{\ell}, *)$ for every $\ell \in L_1$. This yields a DVASS $\mathcal{V}_2 := (\{*\}, \emptyset, H_2, P_1, T_2)$ computable from \mathcal{V}_1 , and the corresponding configuration $*(\mathbf{v} \oplus \ell)$ such that the coverability sets in \mathcal{V}_1 (on the left) is computable from the one in \mathcal{V}_2 (on the right):

▷ **Claim 14.** $\text{COVER}(\ell(\mathbf{v})) = \{\ell'(\mathbf{v}') \mid \mathbf{v}' \oplus \ell' \in \text{COVER}(*(\mathbf{v} \oplus \ell))\}$.

Eventually, as the last step we get rid of plain places H_2 by moving them to atom ones, and considering atoms residing on these atom places irrelevant. Let $P_3 := H_2 \cup P_1$. In order to transfer transitions T_2 from plain places to the new atom places, we introduce the projection mapping $\pi : (H_2 \cup P_1) \times \mathbb{A} \rightarrow H_2 \cup (P_1 \times \mathbb{A})$,

$$(h, a) \mapsto h \quad (p, a) \mapsto (p, a) \quad (h \in H_2, p \in P_1, a \in \mathbb{A}),$$

that forgets, intuitively speaking, about atoms on the new atom places. It extends uniquely to a commutative group homomorphism π from $(H_2 \cup P_1) \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{Z}$ to $H_2 \cup (P_1 \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}$. We define transitions as the inverse image of T_2 along π :

$$T_3 := \pi^{-1}(T_2).$$

This yields a DVAS $\mathcal{V}_3 := (\{*\}, \emptyset, P_3, T_3)$. We observe that $\pi^{-1}(\mathbf{v})$ is orbit-finite for every vector \mathbf{v} , and therefore $\pi^{-1}(T_2)$, being orbit-finite union of orbit-finite sets, is itself orbit-finite [2, Ex. 62]. Therefore \mathcal{V}_3 is computable from \mathcal{V}_2 . The coverability set in \mathcal{V}_2 is computable from the one in \mathcal{V}_3 , since coverability sets commute the projection (the coverability set on the left is in \mathcal{V}_2 , while the one on the right is in \mathcal{V}_3):

▷ **Claim 15.** $\text{COVER}(*(\pi(\mathbf{w}))) = \pi(\text{COVER}(*(\mathbf{w})))$.

Indeed, in order to compute a representation $\text{COVER}(*(\mathbf{v})) = g_1 \downarrow \cup \dots \cup g_n \downarrow$ in \mathcal{V}_2 , we take any \mathbf{w} with $\pi(\mathbf{w}) = \mathbf{v}$, compute a representation $\text{COVER}(*(\mathbf{w})) = f_1 \downarrow \cup \dots \cup f_n \downarrow$ in \mathcal{V}_3 using [16, Thm. 3.5], and modify the functions f_i by summing up, for every $h \in H_2$, namely (under the proviso that $\omega + n = \omega + \omega = \omega$):

$$g_i(h) := \sum_{a \in \mathbb{A}} f_i(h, a) \quad g_i(p, a) := f_i(p, a) \quad (h \in H_2, p \in P_1, a \in \mathbb{A}).$$

This concludes the proof. ◀

5 Sufficient condition for DVASS bi-reachability

In this and in the next section we prove Theorem 7. Throughout the rest of the paper let $\mathcal{V} = (L, \mathcal{R}, H, P, T)$ be an input DVASS. Relying on Lemma 8 we investigate bi-reachability of $q(\mathbf{0})$ and $q'(\mathbf{0})$, for states $q = (\ell, \perp) \in Q$ and $q' = (\ell', \perp') \in Q$ with empty register valuations. The states q, q' are invariant under the action of $\text{Aut}(\mathbb{A})$, which is crucial in the sequel:

▷ **Claim 16.** For every $\sigma \in \text{Aut}(\mathbb{A})$, we have $\sigma(q) = q$ and $\sigma(q') = q'$.

We now formulate a sufficient condition for bi-reachability of $q(\mathbf{0})$ and $q'(\mathbf{0})$, as an adaptation of the classical Θ_1 and Θ_2 conditions. In the context of bi-reachability, it is enough to rely on a simplified version of these conditions given by Lemma 9. We write below $\mathbf{v} \gg \mathbf{0}$ to mean that $\mathbf{v}(h) > 0$ for every $h \in H$, and for every $p \in P$ there is some $a \in \mathbb{A}$ such that $\mathbf{v}(p, a) > 0$.

■ Φ_1 : There are pseudo-runs, each of them using some transition from every orbit in T :

$$q(\mathbf{0}) \dashrightarrow q'(\mathbf{0}) \quad q(\mathbf{0}) \dashleftarrow q'(\mathbf{0}). \quad (5)$$

■ Φ_2 : For some vectors $\Delta, \Delta', \Gamma, \Gamma' \gg \mathbf{0}$, there are runs:

$$\begin{aligned} q(\mathbf{0}) &\longrightarrow q(\Delta) & q'(\Delta') &\longrightarrow q'(\mathbf{0}) \\ q(\mathbf{0}) &\longleftarrow q(\Gamma) & q'(\Gamma') &\longleftarrow q'(\mathbf{0}). \end{aligned} \quad (6)$$

► **Lemma 17.** $\Phi_1 \wedge \Phi_2$ implies $q(\mathbf{0}) \longrightarrow q'(\mathbf{0})$ and $q(\mathbf{0}) \longleftarrow q'(\mathbf{0})$.

Proof. Assume \mathcal{V} satisfies $\Phi_1 \wedge \Phi_2$. Let S be the union of supports of the two pseudo-runs (5) and the four runs (6). Recall that $\text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A}) \subseteq \text{Aut}(\mathbb{A})$ denotes the subset of those automorphisms σ that are identity outside S : $\sigma(a) = a$ for every $a \notin S$. When restricted to S , each such automorphism is a permutation, i.e., $\sigma(S) = S$. In the sequel we will apply permutations $\sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})$ to atoms from S only, and therefore the value $\sigma(a) = a$, for $a \notin S$, will be irrelevant. The set $\text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})$ is finite, $|\text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})| = |S|!$.

We define a (plain) VASS \mathcal{V}_S by, intuitively speaking, restricting the set of atoms to the finite set S . The set of locations of \mathcal{V}_S is $L_S := L \times (\mathcal{R} \rightarrow S \cup \{\perp\})$, its places are $H_S := H \cup P \times S$, and its transitions $T_S \subseteq T$ are all transitions of \mathcal{V} that use only atoms from S . Formally, $\mathcal{V}_S = (L_S, \emptyset, H_S, \emptyset, T_S)$, where $T_S := \{t \in T \mid \text{SUPP}(t) \subseteq S\}$. We claim that the VASS satisfies the conditions Θ_1 and Θ_2 of Lemma 9.

We consider Θ_1 first. Let $\pi : q(\mathbf{0}) \dashrightarrow q'(\mathbf{0})$ and $\pi' : q'(\mathbf{0}) \dashrightarrow q(\mathbf{0})$ be the pseudo-runs in (5). By applying all permutations $\sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})$ to their concatenation $\pi; \pi' : q(\mathbf{0}) \dashrightarrow q'(\mathbf{0}) \dashrightarrow q(\mathbf{0})$, and concatenating all the $|S|!$ resulting cyclic pseudo-runs, we get a cyclic pseudo-run $\delta : q(\mathbf{0}) \dashrightarrow q(\mathbf{0})$. This pseudo-run uses every transition from T_S at least once, since π uses a representative of every orbit of T , and the following fact holds:

▷ **Claim 18.** Let $t, t' \in T$ be two transitions in the same orbit such that $\text{SUPP}(t), \text{SUPP}(t') \subseteq S$. Then $t' = \sigma(t)$ for some $\sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})$.

Likewise we get a cyclic pseudo-run $\delta' : q'(\mathbf{0}) \dashrightarrow q'(\mathbf{0})$ that uses every transition from T_S at least once. Furthermore, for every $m \in \mathbb{N}$, the m -fold concatenation of δ or δ' yields a cyclic pseudo-run that uses every transition from T_S at least m times. We thus have two pseudo-runs

$$\delta^m; \pi : q(\mathbf{0}) \dashrightarrow q'(\mathbf{0}) \quad (\delta')^m; \pi' : q'(\mathbf{0}) \dashleftarrow q'(\mathbf{0})$$

each of them using every transition from T_S at least m times. Thus the VASS \mathcal{V}_S satisfies two instances of Θ_1 , one towards a run $q(\mathbf{0}) \rightarrow q'(\mathbf{0})$ and the other one towards a run $q'(\mathbf{0}) \rightarrow q(\mathbf{0})$.

Now we concentrate on Θ_2 . We proceed similarly as before, namely apply all automorphisms $\sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})$ to Δ , and sum up all the resulting vectors:

$$\Delta_S := \oplus \{ \sigma(\Delta) \mid \sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A}) \}.$$

Let $\overline{\Delta}_S : H \cup P \times S \rightarrow_{\text{fin}} \mathbb{N}$ be the restriction of Δ_S to $H \cup P \times S$. Knowing that $\Delta \gg \mathbf{0}$, we deduce that $\Delta_S(h) > 0$ for every $h \in H$, and $\Delta_S(p, a) > 0$ for every $(p, a) \in P \times S$. In other words, $\overline{\Delta}_S \gg \mathbf{0}$. By applying all automorphisms $\sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})$ to the run $q(\mathbf{0}) \rightarrow q(\Delta)$ in Φ_2 , and concatenating all the resulting runs, we get a run $q(\mathbf{0}) \rightarrow q(\Delta_S)$ in \mathcal{V} . Clearly, only atoms from S appear in this run, and therefore it is also a run $q(\mathbf{0}) \rightarrow q(\overline{\Delta}_S)$ in \mathcal{V}_S . In a similar way we define vectors $\overline{\Delta}'_S$, $\overline{\Gamma}_S$ and $\overline{\Gamma}'_S$, and the corresponding runs in \mathcal{V}_S :

$$q(\mathbf{0}) \rightarrow q(\overline{\Delta}_S) \quad q'(\overline{\Delta}'_S) \rightarrow q'(\mathbf{0}) \quad (7)$$

$$q(\mathbf{0}) \leftarrow q(\overline{\Gamma}_S) \quad q'(\overline{\Gamma}'_S) \leftarrow q'(\mathbf{0}). \quad (8)$$

Therefore, the VASS \mathcal{V}_S satisfies two instances of Θ_2 , one towards a run $q(\mathbf{0}) \rightarrow q'(\mathbf{0})$ and the other one towards a run $q'(\mathbf{0}) \rightarrow q(\mathbf{0})$.

Finally, using Lemma 9 we deduce two runs in \mathcal{V}_S , which are automatically also runs in \mathcal{V} . This completes the proof. \blacktriangleleft

6 Reduction algorithm

As the *rank* of a DVASS $\mathcal{V} = (L, \mathcal{R}, H, P, T)$ we take the triple $\text{RANK}(\mathcal{V}) = (|P|, |H|, ||T||)$, consisting of the number of atom places, the number of plain places, and the number of orbits $||T||$ the set T partitions into. Ranks are compared lexicographically.

Given a DVASS \mathcal{V} , the algorithm verifies the conditions Φ_1 and Φ_2 . If they are all satisfied, it answers positively, relying on Lemma 17. Otherwise, depending on which of the conditions is violated, the algorithm either immediately answers negatively, or applies a reduction step, as outlined below in Sections 6.1 and 6.2. Each of the steps produces a new DVASS $\widehat{\mathcal{V}}$ of strictly smaller rank, which guarantees termination. Finally, when both P and H are empty, the problem reduces to reachability from q to q' in state graph $\text{GRAPH}(\mathcal{V})$, which is decidable due to:

► **Lemma 19.** *For a set $E \subseteq Q \times Q$ of edges between states, given as a finite union of orbits, and a pair $(s, s') \in Q \times Q$, it is decidable if there is a path from s to s' in the graph (Q, E) .*

Proof. The orbit of an edge $((\ell, \nu), (\ell', \nu')) \in E$ is determined by the following data: locations ℓ, ℓ' ; the inverse images $\nu^{-1}(\perp)$, $(\nu')^{-1}(\perp)$; and the equality type of the remaining entries in ν and ν' , that is:

$$\{(r, r') \mid \nu(r) = \nu(r') \neq \perp\} \quad \{(r, r') \mid \nu'(r) = \nu'(r') \neq \perp\} \quad \{(r, r') \mid \nu(r) = \nu'(r') \neq \perp\}.$$

Using equational reasoning, one computes the transitive closure E^* of E , by consecutively adding to E^* every new orbit which is forced to be included in E^* by some two orbits already included in E^* , until saturation. Termination is guaranteed as $Q \times Q$ is orbit-finite. The transitive closure is thus forcedly a finite union of orbits. Finally, one tests if the orbit of (s, s') is included in E^* . \blacktriangleleft

31:12 Bi-Reachability in Petri Nets with Data

For future use we note an immediate consequence of the above proof: for every pair of states, if there is a path from one to the other, then there is also a path of bounded length. This implies a bound on the number of atoms involved:

► **Corollary 20.** *There is an effective bound $b(Q) \in \mathbb{N}$ such that whenever there is a path from $s \in Q$ to $s' \in Q$ in the graph (Q, E) , then there is such a path π with $|\text{SUPP}(\pi)| \leq b(Q)$.*

Below we describe the two reduction steps, proving their progress property (decreasing rank), correctness and effectiveness.

6.1 Violation of Φ_1

Suppose \mathcal{V} violates Φ_1 . If states q, q' are not in the same strongly connected component of $\text{GRAPH}(T)$, which is testable using Lemma 19, the configurations $q(\mathbf{0})$, $q'(\mathbf{0})$ are clearly not bi-reachable and the algorithm answers negatively. Otherwise, the algorithm constructs a DVASS $\widehat{\mathcal{V}}$ of smaller rank, as defined below, such that bi-reachability of $q(\mathbf{0})$ and $q'(\mathbf{0})$ in \mathcal{V} is equivalent to their bi-reachability in $\widehat{\mathcal{V}}$.

Let $\text{GRAPH}(T) = (Q, E)$. Transitions witnessing bi-reachability of $q(\mathbf{0})$ and $q'(\mathbf{0})$, namely used in some cyclic run $q(\mathbf{0}) \rightarrow q'(\mathbf{0}) \rightarrow q(\mathbf{0})$, form a cycle in $\text{GRAPH}(T)$. As a consequence, a transition $(s, \mathbf{v}, s') \in T$ may be useful for bi-reachability only if the edge (s, s') belongs to the strongly connected component of $\text{GRAPH}(T)$ containing q and q' . Therefore, bi-reachability of $q(\mathbf{0})$, $q'(\mathbf{0})$ in \mathcal{V} reduces to bi-reachability of $q(\mathbf{0})$, $q'(\mathbf{0})$ in $\widehat{\mathcal{V}}$ obtained by restriction to the strongly connected component of q and q' . This component is computed by enumerating all orbits included in E . For every orbit $o \subseteq E$ one chooses a representative $(s, s') \in o$, and uses Lemma 19 to test reachability, in $\text{GRAPH}(T)$, for the four pairs: (q, s) , (s', q) , (q', s) or (s', q') . Then one removes from T all orbits of transitions (s, \mathbf{v}, s') such that reachability test fails for any of the four pairs above. The resulting set of transitions is still a finite union of orbits. Consequently, from now on we may assume, w.l.o.g., that $\text{GRAPH}(T)$ is strongly connected (we ignore isolated vertices).

Useful transitions. By a finite multiset of transitions we mean a nonnegative vector $\mathbf{f} : T \rightarrow_{\text{fin}} \mathbb{N}$. Given such a finite multiset, let $\text{STATE-EQ}(q, q')$ denote conjunction of the following conditions:

- (a) the sum of effects of all transitions in the multiset is $\mathbf{0}$,
- (b) for every state $s \notin \{q, q'\}$, the number of transitions incoming to s equals the number of ones outgoing from s ,
- (c) the number of transitions outgoing from q exceeds by one the number of incoming ones,
- (d) the number of transitions incoming to q' exceeds by one the number of outgoing ones;

Symmetrically, let $\text{STATE-EQ}(q', q)$ denote the conjunction of (a), (b) and the symmetric versions of (c) and (d) with q and q' swapped. Let $O = \{\text{ORBIT}(t) \mid t \in T\}$ be the set of all orbits in T . We call an orbit $o \in O$ *useful* if there are two finite multisets of transitions \mathbf{f}, \mathbf{f}' satisfying $\text{STATE-EQ}(q, q')$ and $\text{STATE-EQ}(q', q)$, respectively, each of them containing some transition from o .

► **Lemma 21.** Φ_1 holds if and only if all orbits of transitions are useful.

Proof. The only if direction of the characterisation is immediate, as the multiset of transitions used in a pseudo-run $q(\mathbf{0}) \dashrightarrow q'(\mathbf{0})$ necessarily contains some transition from every orbit and satisfies all the conditions (a)–(d), and likewise for a pseudo-run $q'(\mathbf{0}) \dashrightarrow q(\mathbf{0})$. For

the opposite direction, suppose that for every orbit $o \in O$ there are finite multisets $\mathbf{f}_o, \mathbf{f}'_o$ satisfying $\text{STATE-EQ}(q, q')$ and $\text{STATE-EQ}(q', q)$, respectively, each of them containing some transition from o . Let

$$\mathbf{f} := \oplus \{\mathbf{f}_o \mid o \in O\} \quad \mathbf{f}' := \oplus \{\mathbf{f}'_o \mid o \in O\} \quad S := \text{SUPP}(\mathbf{f}) \cup \text{SUPP}(\mathbf{f}'),$$

where \oplus denotes the multiset sum operator. Thus S is the (finite) set of atoms used in all the transitions appearing in \mathbf{f} or \mathbf{f}' . Similarly as in the proof of Lemma 17 we use the subgroup $\text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A}) \subseteq \text{Aut}(\mathbb{A})$ of automorphisms σ of \mathbb{A} that are identity outside S , and define a plain VASS $\mathcal{V}_S = (L_S, \emptyset, H_S, \emptyset, T_S)$ by restricting the set of atoms to S . Locations of \mathcal{V}_S are $L_S := L \times (\mathcal{R} \rightarrow S \cup \{\perp\})$, its places are $H_S := H \cup P \times S$, and its transitions $T_S \subseteq T$ are all transitions of \mathcal{V} that use only atoms from S : $T_S = \{t \in T \mid \text{SUPP}(t) \subseteq S\}$. The state graph $\text{GRAPH}(T_S)$ is a subgraph of $\text{GRAPH}(T)$. As $\text{GRAPH}(T)$ is strongly connected, we use Corollary 20 to deduce that, for sufficiently large S , the state graph $\text{GRAPH}(T_S)$ is also strongly connected. Therefore we enlarge S , if necessary, to assure that $\text{GRAPH}(T_S)$ is strongly connected.

As finite multisets \mathbf{f}, \mathbf{f}' are just nonnegative vectors $T \rightarrow_{\text{fin}} \mathbb{N}$, they inherit the natural (pointwise) action of $\text{Aut}(\mathbb{A})$. Basing on \mathbf{f}, \mathbf{f}' we define two larger multisets of transitions by applying all automorphisms from $\text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})$ to \mathbf{f} and \mathbf{f}' , respectively, and summing up all the resulting multisets:

$$\mathbf{g} := \oplus \{\sigma(\mathbf{f}) \mid \sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})\} \quad \mathbf{g}' := \oplus \{\sigma(\mathbf{f}') \mid \sigma \in \text{Aut}_{\mathbb{A} \setminus S}(\mathbb{A})\}.$$

By Claim 18, each of \mathbf{g}, \mathbf{g}' contains all transitions from T_S . Furthermore, the multiset $\mathbf{h} = \mathbf{f} \oplus \mathbf{g} \oplus \mathbf{g}'$ satisfies $\text{STATE-EQ}(\hat{q}, \hat{q}')$, where \hat{q}, \hat{q}' are locations (=states) of \mathcal{V}_S corresponding to q and q' respectively. Likewise, the multiset $\mathbf{h}' = \mathbf{f}' \oplus \mathbf{g} \oplus \mathbf{g}'$ satisfies $\text{STATE-EQ}(\hat{q}', \hat{q})$. Using the standard Euler argument in the (strongly) connected graph $\text{GRAPH}(T_S)$, and relying on conditions (b)–(d), we deduce existence of a pseudo-run in \mathcal{V}_S that uses exactly transitions \mathbf{h} . Due to condition (a), this is a pseudo-run $\hat{q}(\mathbf{0}) \dashrightarrow \hat{q}'(\mathbf{0})$ in \mathcal{V}_S . Likewise we deduce a pseudo-run $\hat{q}'(\mathbf{0}) \dashrightarrow \hat{q}(\mathbf{0})$ in \mathcal{V}_S . The pseudo-runs are essentially also pseudo-runs in \mathcal{V} , both supported by S , and both using some transition from every orbit in T . This completes the proof of the characterisation. \blacktriangleleft

Reduction step. We define a DVASS $\hat{\mathcal{V}}$ by removing some useless orbit of transitions, $\hat{\mathcal{V}} = (L, \mathcal{R}, H, P, \hat{T})$. It has the same locations, registers and places as \mathcal{V} .

► **Lemma 22 (Progress).** $\|\hat{T}\| < \|T\|$, and hence $\text{RANK}(\hat{\mathcal{V}}) < \text{RANK}(\mathcal{V})$.

Proof. As some useless orbit of transitions is removed from \hat{T} , we have $\|\hat{T}\| < \|T\|$, and therefore $\text{RANK}(\hat{\mathcal{V}}) = (|P|, |H|, \|\hat{T}\|) < (|P|, |H|, \|T\|) = \text{RANK}(\mathcal{V})$. \blacktriangleleft

► **Lemma 23 (Correctness).** *The configurations $q(\mathbf{0}), q'(\mathbf{0})$ are bi-reachable in \mathcal{V} if and only if they are bi-reachable in $\hat{\mathcal{V}}$.*

Proof. Indeed, useless transitions can not be used in runs between $q(\mathbf{0})$ and $q'(\mathbf{0})$. \blacktriangleleft

► **Lemma 24 (Effectiveness).** *The condition Φ_1 is decidable. When it fails, some useless orbit of transitions is computable.*

Proof. It is sufficient to prove that it is decidable if a given orbit $o \in O$ is useful. We show decidability by reduction to MULTISSET SUM (recall Lemma 4), i.e., we use the algorithm for MULTISSET SUM to check existence of a finite multiset \mathbf{f} that satisfies conditions (a)–(d) and

31:14 Bi-Reachability in Petri Nets with Data

contains a transition from o (and likewise to check existence of \mathbf{f}'). Let $X = H \cup P \times \mathbb{A}$ and $Y = X \cup Q \cup \{*\}$. The set Y is orbit-finite. Let every transition $t = (s, \mathbf{v}, s') \in T$ determine a vector $\mathbf{y}_t : Y \rightarrow_{\text{fin}} \mathbb{Z}$, and let every transition $t = (s, \mathbf{v}, s') \in o$ determine additionally a vector $\mathbf{x}_t : Y \rightarrow_{\text{fin}} \mathbb{Z}$, each of them extending the vector $\mathbf{v} : X \rightarrow_{\text{fin}} \mathbb{Z}$ as follows:

$$\begin{array}{lll} \mathbf{x}_t(s) = -1 & \mathbf{x}_t(s') = 1 & \mathbf{x}_t(*) = 1, \\ \mathbf{y}_t(s) = -1 & \mathbf{y}_t(s') = 1 & \mathbf{y}_t(*) = 0, \end{array} \quad (9)$$

and $\mathbf{x}_t(r) = \mathbf{y}_t(r) = 0$ for other states $r \in Q \setminus \{s, s'\}$. Intuitively, the first two columns track contribution of t to the number of transitions outgoing from s , or incoming to s' , while the latter column tracks the number of usages of transitions from o . Likewise, let the target vector $\mathbf{b} : Y \rightarrow_{\text{fin}} \mathbb{Z}$ extend $\mathbf{0} : X \rightarrow_{\text{fin}} \mathbb{Z}$ by

$$\mathbf{b}(q) = -1 \quad \mathbf{b}(q') = 1 \quad \mathbf{b}(*) = 1, \quad (10)$$

and $\mathbf{b}(s) = 0$ for other states $s \in Q \setminus \{q, q'\}$. Let $M = \{\mathbf{y}_t \mid t \in T\} \cup \{\mathbf{x}_t \mid t \in o\}$, and consider the instance let (M, \mathbf{b}) of MULTISUM. Observe that solutions of (M, \mathbf{b}) necessarily use exactly one vector \mathbf{x}_t exactly once, as in (11) below. It remains to prove that some finite multiset $\mathbf{f} = \{t, u_1, \dots, u_m\}$ of transitions from T , where $t \in o$, satisfies the conditions (a)–(d) exactly when

$$\mathbf{b} = \mathbf{x}_t + \mathbf{y}_{u_1} + \dots + \mathbf{y}_{u_m}. \quad (11)$$

As $\mathbf{b}(c) = 0$ for all $c \in H \cup P \times \mathbb{A}$, condition (a) is equivalent to the equality (11) restricted to $H \cup P \times \mathbb{A}$. By the first two columns in (9) and (10), and since $\mathbf{b}(s) = 0$ for $s \in Q \setminus \{q, q'\}$, condition (b) is equivalent to the equality (11) restricted to $Q \setminus \{q, q'\}$, and the conditions (c) and (d) are equivalent to the equality (11) restricted to $\{q, q'\}$. ◀

6.2 Violation of Φ_2

Suppose \mathcal{V} violates Φ_2 . We define a DVASS $\widehat{\mathcal{V}}$ of smaller rank and two its states $\widehat{q}, \widehat{q}'$ such that bi-reachability of $q(\mathbf{0})$ and $q'(\mathbf{0})$ in \mathcal{V} is equivalent to bi-reachability of $\widehat{q}(\mathbf{0})$ and $\widehat{q}'(\mathbf{0})$ in $\widehat{\mathcal{V}}$.

Pumpability and boundedness. Any run of the form $q(\mathbf{0}) \longrightarrow q(\Delta)$ (resp. $q(\Delta) \longrightarrow q(\mathbf{0})$) we call *forward* (resp. *backward*) *pump* from q . Likewise we define forward (resp. backward) pumps from q' .

A plain place $h \in H$ (resp. an atom place $p \in P$) we call *forward pumpable* from q if there is a pump $q(\mathbf{0}) \longrightarrow q(\Delta)$ such that $\Delta(h) > 0$ (resp. $\Delta(p, a) > 0$ for some atom $a \in \mathbb{A}$). Symmetrically, a place $h \in H$ (resp. $p \in P$) we call *backward pumpable* from q if there is a pump $q(\Delta) \longrightarrow q(\mathbf{0})$ such that $\Delta(h) > 0$ (resp. $\Delta(p, a) > 0$ for some atom $a \in \mathbb{A}$). Likewise we define places forward (resp. backward) pumpable from q' . Finally, a (plain or atom) place is called *pumpable* if it is forward and backward pumpable both from q and from q' . Otherwise, the place is called *unpumpable*.

► **Lemma 25.** Φ_2 holds if and only if all places are pumpable.

Proof. In one direction, Φ_2 amounts to *simultaneous* (= using one pump) forward and backward pumpability of all places, from both q and q' . In the converse direction, suppose all places are forward and backward pumpable from both q and q' . We observe that forward (resp. backward) pumps from q compose, and hence all places are *simultaneously* forward (resp. backward) pumpable from q . Likewise for q' . ◀

We now introduce a suitable version of boundedness. In case of plain places $h \in H$ boundedness applies, as expected, to the value $\mathbf{v}(h)$ in a configuration $s(\mathbf{v})$. On the other hand in case of atom places $p \in P$, boundedness applies to the number of tokens on a place. For uniformity we define the size of a place $p \in P$ in a vector \mathbf{v} as

$$\mathbf{v}(p) = \sum \{\mathbf{v}(p, a) \mid a \in \mathbb{A}, \mathbf{v}(p, a) > 0\} \in \mathbb{N}. \quad (12)$$

For a family \mathcal{F} of runs, we say that a place $c \in H \cup P$ is *bounded on \mathcal{F} by $B \in \mathbb{N}$* if for every configuration $s(\mathbf{v})$ appearing in every run in the family \mathcal{F} , $\mathbf{v}(c) \leq B$. A place is *bounded on \mathcal{F}* , if it is bounded on \mathcal{F} by some B . Otherwise, the place is called *unbounded on \mathcal{F}* .

As forward (resp. backward) pumps compose, every place which is forward (resp. backward) pumpable from q (resp. q') is necessarily unbounded on respective pumps. We show the opposite implication, namely unpumpable places are bounded on respective pumps:

► **Lemma 26.** *A place which is not forward (resp. backward) pumpable from q is bounded on forward (resp. backward) pumps from q . Likewise for q' .*

Proof. W.l.o.g. we focus on forward pumps from q only. (Backward pumps are tackled similarly as forward ones, but in the *reversed* DVASS, whose transitions $\{(s', -\mathbf{v}, s) \mid (s, \mathbf{v}, s') \in T\}$ are inverses of transitions of \mathcal{V} .) Assuming that a place $c \in H \cup P$ is unbounded on forward pumps from q , we show that c is forward pumpable from q . Unboundedness of c means that for every $i \in \mathbb{N}$ there is a run

$$q(\mathbf{0}) \longrightarrow s_i(\mathbf{v}_i) \longrightarrow q(\mathbf{\Delta}_i) \quad (13)$$

such that $n_i := \mathbf{v}_i(c) > i$. By choosing a subsequence of $(n_i)_i$ we may assume w.l.o.g. that this sequence is strictly increasing. As \sqsubseteq is a WQO, for some $j < i$ we have $s_j(\mathbf{v}_j) \sqsubseteq s_i(\mathbf{v}_i)$, i.e., there is some $\sigma \in \text{Aut}(\mathbb{A})$ such that $\sigma(s_j) = s_i$ and $\sigma(\mathbf{v}_j) \leq \mathbf{v}_i$. Equivalently, $\sigma(\mathbf{v}_j) + \mathbf{\Delta} = \mathbf{v}_i$ for some nonnegative vector $\mathbf{\Delta}$. Recall that $q = (\ell, \perp)$ and hence $\sigma(q) = q$. Therefore, we can construct a new run, by first using the first half of (13) to go from $p(\mathbf{0})$ to $s_i(\mathbf{v}_i)$, and then applying σ to the second half $s_j(\mathbf{v}_j) \longrightarrow q(\mathbf{\Delta}_j)$ to return from $\sigma(s_j(\mathbf{v}_j)) = \sigma(s_j)(\sigma(\mathbf{v}_j))$:

$$q(\mathbf{0}) \longrightarrow s_i(\mathbf{v}_i) = \sigma(s_j)(\sigma(\mathbf{v}_j) + \mathbf{\Delta}) \longrightarrow q(\sigma(\mathbf{\Delta}_j) + \mathbf{\Delta}).$$

By strict monotonicity of $(n_i)_i$ we have $n_j < n_i$, i.e., $\mathbf{v}_j(c) < \mathbf{v}_i(c)$, which implies $\mathbf{\Delta}(c) > 0$. The place c is thus forward pumpable from q , as required. ◀

Reduction step. Let $B \in \mathbb{N}$ be the universal bound for all unpumpable places on all respective pumps. Formally, let's assume that B bounds every place which is not forward (resp. backward) pumpable from q , on all forward (resp. backward) pumps from q , and B also bounds every place which is not forward (resp. backward) pumpable from q' , on all forward (resp. backward) pumps from q' . As \mathcal{V} violates Φ_2 , we know by Lemma 25 that there are some unpumpable places. We define a DVASS $\widehat{\mathcal{V}} = (\widehat{L}, \widehat{\mathcal{R}}, \widehat{H}, \widehat{P}, \widehat{T})$ by, intuitively speaking, removing some such place. Let $X = H \cup P \times \mathbb{A}$.

Case 1: some plain place is unpumpable. Let $\widehat{P} := P$ and $\widehat{\mathcal{R}} = \mathcal{R}$. We choose an arbitrary unpumpable $h \in H$, and let $\widehat{H} := H \setminus \{h\}$. We keep track of values on h by extending the locations $\widehat{L} := L \times \{0 \dots B\}$. Finally, transitions \widehat{T} are obtained from transitions T by replacing each $t = ((\ell, \mu), \mathbf{v}, (\ell', \mu')) \in T$ by all the transitions of the form

$$\widehat{t} = (((\ell, n), \mu), \mathbf{w}, ((\ell', n + \mathbf{v}(h)), \mu')),$$

where \mathbf{w} is the restriction of $\mathbf{v} : (H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}$ to $\widehat{H} \cup P \times \mathbb{A}$. Let $\widehat{q} = ((\ell, 0), \perp)$ and $\widehat{q}' = ((\ell', 0), \perp)$.

Case 2: some atom place is unpumpable. Let $\widehat{L} := L$ and $\widehat{H} := H$. We choose an arbitrary unpumpable $p \in P$, remove place p , namely $\widehat{P} := P \setminus \{p\}$, and add B new registers, namely $\widehat{\mathcal{R}} = \mathcal{R} \cup \{r_1, \dots, r_B\}$, which store, intuitively speaking, every possible content of the place p . Using the new registers the transitions \widehat{T} track, intuitively speaking, the effect of transitions from T on the removed place p . Every register valuation μ naturally induces a vector $\widehat{\mu} : \{p\} \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{N}$, namely $\widehat{\mu} := \oplus \{(p, a) \mid \mu(r_i) = a \neq \perp, i \in \{1 \dots B\}\}$. Using this notation we define the transitions \widehat{T} by replacing every transition $((\ell, \nu), \mathbf{v}, (\ell', \nu')) \in T$ with all the transitions of the form

$$((\ell, \nu \oplus \mu), \mathbf{w}, (\ell', \nu' \oplus \mu')),$$

where \mathbf{w} is the restriction of $\mathbf{v} : (H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{Z}$ to $H \cup \widehat{P} \times \mathbb{A}$, and $\mathbf{v} = \mathbf{w} \oplus \widehat{\mu}' \ominus \widehat{\mu}$. Let $\widehat{q}, \widehat{q}'$ be extensions of the states q, q' , respectively, by empty valuation of all the new registers $\{r_1, \dots, r_B\}$.

► **Lemma 27 (Progress).** $\text{RANK}(\widehat{\mathcal{V}}) < \text{RANK}(\mathcal{V})$.

Proof. In the former case $\widehat{P} = P$ and $|\widehat{H}| < |H|$, while in the latter case $|\widehat{P}| < |P|$. In each case, $\text{RANK}(\widehat{\mathcal{V}}) = (|\widehat{P}|, |\widehat{H}|, |\widehat{T}|) < (|P|, |H|, |T|) = \text{RANK}(\mathcal{V})$. ◀

► **Lemma 28 (Correctness).** *The configurations $q(\mathbf{0})$, $q'(\mathbf{0})$ are bi-reachable in \mathcal{V} if and only if $\widehat{q}(\mathbf{0})$, $\widehat{q}'(\mathbf{0})$ are bi-reachable in $\widehat{\mathcal{V}}$.*

Proof. As $\widehat{\mathcal{V}}$ is obtained from \mathcal{V} by restricting values in configurations on some places, each run in $\widehat{\mathcal{V}}$ is automatically a run in \mathcal{V} , and bi-reachability in $\widehat{\mathcal{V}}$ implies bi-reachability in \mathcal{V} .

For the converse implication, suppose $q(\mathbf{0})$, $q'(\mathbf{0})$ are bi-reachable in \mathcal{V} , and fix two arbitrary runs $\pi : q(\mathbf{0}) \rightarrow q'(\mathbf{0})$ and $\pi' : q'(\mathbf{0}) \rightarrow q(\mathbf{0})$. Consider any unpumpable place $c \in H \cup P$ and suppose, w.l.o.g., that the place is not forward pumpable from q . By Lemma 26 the place is bounded by B on all forward pumps from q . In particular, it is bounded by B on the composed run $\pi; \pi' : q(\mathbf{0}) \rightarrow q(\mathbf{0})$, i.e., on both π and π' . Therefore we know that in each configuration (s, \mathbf{v}) in both runs, $\mathbf{v}(c) \leq B$. If $c \in H$ we keep track of $\mathbf{v}(c)$ in locations; and if $c \in P$, we keep track of all tokens on c by storing their atoms in the new registers $\{r_1, \dots, r_B\}$. Therefore, both runs are realisable in $\widehat{\mathcal{V}}$. ◀

► **Lemma 29 (Effectiveness).** *The set of pumpable places and the universal bound B are computable (and therefore the condition Φ_2 is decidable).*

Proof. For a simple ω -configuration $s(f)$ and an atom place $p \in P$ we define $f(p) = \omega$ if $f(p, a) = \omega$ for some atom $a \in \mathbb{A}$, and otherwise $f(p) := \sum \{f(p, a) \mid f(p, a) > 0, a \in \mathbb{A}\}$. Relying on Lemma 25, we test forward (resp. backward) pumpability of every place from q (and likewise from q'). Specifically, to test if a place is forward pumpable from q , we apply Lemma 12 and compute a simple-ideal representation of the coverability set $\text{COVER}(q(\mathbf{0}))$, given by simple ω -configurations $G = \{s_1(g_1), \dots, s_n(g_n)\}$. A plain or atom place $c \in H \cup P$ is forward pumpable from q if for some $i \in \{1 \dots n\}$ we have $s_i = q$ and $g_i(c) > 0$. Likewise we test if a place is backward pumpable from q (using the reverse DVASS), or forward (resp. backward) pumpable from q' . The set of pumpable places is thus computable.

Suppose there is some unpumpable place c , w.l.o.g. say not forward pumpable from q . By Lemma 26, the place c is bounded on all forward pumps from q , but may be a priori unbounded on some other run, and therefore it may happen that $g_i(c) = \omega$ for some $i \in \{1 \dots n\}$. Nevertheless we claim the following bound on forward pumps:

$$B := \max \{g_i(c) \mid g_i(c) < \omega, i \in \{1 \dots n\}\}. \quad (14)$$

▷ Claim 30. The place c is bounded on all forward pumps from q by B .

Proof. Consider an atom place $p \in P$ (the argument for plain places is similar but simpler). Towards contradiction, suppose that some configuration $s(\mathbf{v})$ on some forward pump from q satisfies $\mathbf{v}(p) > B$. We have thus two runs:

$$\pi : q(\mathbf{0}) \longrightarrow s(\mathbf{v}) \quad \rho : s(\mathbf{v}) \longrightarrow q(\mathbf{w})$$

for some $s \in Q$ and nonnegative vector \mathbf{w} . Therefore $s(\mathbf{v}) \in \text{COVER}(q(\mathbf{0}))$, and hence $s(\mathbf{v}) \in I = s_j(g_j) \downarrow$ for some $j \in \{1 \dots n\}$. As $\mathbf{v}(p)$ is larger than all $g_i(p) < \omega$ for $i \in \{1 \dots n\}$, we deduce that $g_j(p) = \omega$. Therefore the ideal I contains configurations $s'(\mathbf{v}')$ with arbitrary large values of $\mathbf{v}'(p)$. In particular, I contains some configuration $s'(\mathbf{v}')$ with $\mathbf{v}(p) < \mathbf{v}'(p)$. As I is directed, it contains a configuration $s''(\mathbf{v}'')$ such that

$$s(\mathbf{v}) \sqsubseteq s''(\mathbf{v}'') \quad s'(\mathbf{v}') \sqsubseteq s''(\mathbf{v}''),$$

which implies $\mathbf{v}(p) < \mathbf{v}''(p)$. Finally, as $I \subseteq \text{COVER}(q(\mathbf{0}))$, it must contain a configuration $\bar{s}(\bar{\mathbf{v}})$ reachable from $q(\mathbf{0})$, such that $s''(\mathbf{v}'') \sqsubseteq \bar{s}(\bar{\mathbf{v}})$. We thus have $s(\mathbf{v}) \sqsubseteq \bar{s}(\bar{\mathbf{v}})$ and $\mathbf{v}(p) < \bar{\mathbf{v}}(p)$, which means that for some automorphism $\sigma \in \text{Aut}(A)$ and vector $\bar{\mathbf{w}}$ we have

$$\sigma(s) = \bar{s} \quad \sigma(\mathbf{v}) + \bar{\mathbf{w}} = \bar{\mathbf{v}} \quad \bar{\mathbf{w}}(p) > 0.$$

By composing runs $\bar{\pi} : q(\mathbf{0}) \rightarrow \bar{s}(\bar{\mathbf{v}})$ and $\sigma(\rho) : \bar{s}(\sigma(\mathbf{v})) \rightarrow q(\sigma(\mathbf{w}))$ we get a run

$$\bar{\pi}; \sigma(\rho) : q(\mathbf{0}) \longrightarrow q(\sigma(\mathbf{w}) + \bar{\mathbf{w}}),$$

i.e., we deduce that p is forward pumpable from q , which is a contradiction. ◁

The universal bound is computed as the maximum of (14) for forward and backward pumps from q and q' , for all unbounded places c . ◀

7 Final remarks

We show decidability of the bi-reachability problem for Petri nets with equality data. The problem subsumes coverability, and reachability in the reversible subclass, and therefore the result pushes further the decidability border towards the reachability problem. The latter problem (which we believe to be decidable) is still beyond our reach, and development of this paper is not sufficient. For instance, the approach of proving of Lemma 17 would fail for reachability, as we rely on the fact that bi-reachability implies a cycle. Moreover, Φ_1 reduction step would fail as well, as it assumes that a transition (orbit) is either unusable, or usable unboundedly, while in case of reachability a transition can be usable only *boundedly*.

Our approach is specific to equality data, and thus we leave unresolved the status of bi-reachability in case of ordered data. In case of ordered data domain the approach of proving Lemma 17 would fail again, as the trick of applying all permutations of S would be impossible. Moreover, it is not clear how to implement Φ_2 reduction step, as no procedure computing coverability sets is known.

References

- 1 Michael Blondin and Francois Ladouceur. Population protocols with unordered data. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *Proc. ICALP 2023*, volume 261 of *LIPICs*, pages 115:1–115:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICALP.2023.115.

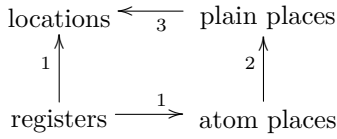
- 2 Mikołaj Bojańczyk. Slightly infinite sets, 2019. URL: <https://www.mimuw.edu.pl/~bojan/paper/atom-book>.
- 3 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3:4):paper 4, 2014.
- 4 Mikołaj Bojańczyk, Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk. Turing machines with atoms. In *LICS*, pages 183–192, 2013.
- 5 Iliano Cervesato, Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *Proc. CSFW 1999*, pages 55–69. IEEE Computer Society, 1999. doi:10.1109/CSFW.1999.779762.
- 6 Iliano Cervesato, Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *Proc. CSFW 1999*, pages 55–69, 1999.
- 7 Giorgio Delzanno. An overview of MSR(C): A CLP-based framework for the symbolic verification of parameterized concurrent systems. *Electr. Notes Theor. Comput. Sci.*, 76:65–82, 2002.
- 8 Giorgio Delzanno. Constraint multiset rewriting. Technical Report DISI-TR-05-08, DISI, Università di Genova, 2005.
- 9 Diego Figueira, Ranko Lazic, Jérôme Leroux, Filip Mazowiecki, and Grégoire Sutre. Polynomial-space completeness of reachability for succinct branching VASS in dimension one. In *Proc. ICALP 2017*, volume 80 of *LIPICs*, pages 119:1–119:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.119.
- 10 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: completions. In Susanne Albers and Jean-Yves Marion, editors, *Proc. STACS 2009*, volume 3 of *LIPICs*, pages 433–444. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2009.
- 11 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part II: complete WSTS. *Log. Methods Comput. Sci.*, 8(3), 2012.
- 12 Moses Ganardi, Rupak Majumdar, Andreas Pavlogiannis, Lia Schütze, and Georg Zetsche. Reachability in bidirected pushdown VASS. In *Proc. ICALP 2022*, volume 229 of *LIPICs*, pages 124:1–124:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 13 Hartmann J. Genrich and Kurt Lautenbach. System modelling with high-level Petri nets. *Theor. Comput. Sci.*, 13:109–136, 1981.
- 14 A. Ghosh, P. Hofman, and S. Lasota. Orbit-finite linear programming. In *Proc. LICS 2023*, pages 1–14, 2023.
- 15 Arka Ghosh and Sławomir Lasota. Equivariant ideals of polynomials. In *Proc. LICS 2024*, pages 38:1–38:14. ACM, 2024. doi:10.1145/3661814.3662074.
- 16 Piotr Hofman, Sławomir Lasota, Ranko Lazic, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability trees for petri nets with unordered data. In Bart Jacobs and Christof Löding, editors, *Proc. FOSSACS 2016*, volume 9634 of *Lecture Notes in Computer Science*, pages 445–461. Springer, 2016.
- 17 Kurt Jensen. Coloured Petri nets and the invariant-method. *Theor. Comput. Sci.*, 14:317–336, 1981.
- 18 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proc. STOC 1982*, pages 267–281, 1982.
- 19 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992.
- 20 Sławomir Lasota. Decidability border for Petri nets with data: WQO dichotomy conjecture. In *Proc. Petri Nets 2016*, volume 9698 of *Lecture Notes in Computer Science*, pages 20–36. Springer, 2016.
- 21 Sławomir Lasota. VASS reachability in three steps. *CoRR*, abs/1812.11966, 2018. arXiv:1812.11966.
- 22 Ranko Lazic, Thomas Christopher Newcomb, Joël Ouaknine, A. W. Roscoe, and James Worrell. Nets with tokens which carry data. In *Proc. ICATPN 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2007.

- 23 Ranko Lazic and Patrick Totzke. What makes Petri nets harder to verify: stack or data? In *Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*, volume 10160 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2017.
- 24 Jérôme Leroux. Vector addition system reversible reachability problem. *Log. Methods Comput. Sci.*, 9(1), 2013.
- 25 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proc. LICS 2015*, pages 56–67. IEEE Computer Society, 2015.
- 26 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proc. LICS 2019*, pages 1–13. IEEE, 2019.
- 27 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. STOC 1981*, pages 238–246, 1981.
- 28 Fernando Rosa-Velardo and David de Frutos-Escrig. Decidability and complexity of Petri nets with unordered data. *Theor. Comput. Sci.*, 412(34):4439–4451, 2011.
- 29 Kumar Neeraj Verma and Jean Goubault-Larrecq. Karp-Miller trees for a branching extension of VASS. *Discret. Math. Theor. Comput. Sci.*, 7(1):217–230, 2005. doi:10.46298/DMTCS.350.

A Proofs for Section 3 (Data vector addition systems with states)

► **Lemma 8.** *In DVASS BI-REACHABILITY problem we may assume, w.l.o.g., that $q = (\ell, \perp)$, $q' = (\ell', \perp)$, and $\mathbf{v} = \mathbf{v}' = \mathbf{0}$.*

Proof. Consider a DVASS $\mathcal{V} = (L, \mathcal{R}, P, H, T)$ and two configurations $q(\mathbf{v}), q'(\mathbf{v}')$, where $q = (\ell, \nu)$, $q' = (\ell', \nu')$. We proceed in three steps, as shown in the diagram (cf. diagram (4) in Remark 6):



As the first step we redo the first step of the proof of Lemma 12 which yields a DVASS $\mathcal{V}_1 = (L_1, \emptyset, H, P_1, T_1)$ without registers (which implies $q = (\ell, \perp)$ and $q' = (\ell', \perp)$). We choose initial and final location

$$\bar{\ell} := (\ell, \nu^{-1}(\perp)) \quad \bar{\ell}' := (\ell', (\nu')^{-1}(\perp)) \in L_1$$

and, identifying a register valuation ν with the vector $\oplus \{(r, a) \mid \nu(r) = a \neq \perp\}$, we choose initial and final vector $H \cup P_1 \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{N}$,

$$\bar{\mathbf{v}} := \mathbf{v} \oplus \nu \quad \bar{\mathbf{v}}' := \mathbf{v}' \oplus \nu',$$

and claim that the reachability is preserved (we omit registers, and write e.g. $\bar{\ell}(\bar{\mathbf{v}})$):

▷ **Claim 31.** The configurations $q(\mathbf{v}), q'(\mathbf{v}')$ are bi-reachable in \mathcal{V} if and only if $\bar{\ell}(\bar{\mathbf{v}}), \bar{\ell}'(\bar{\mathbf{v}}')$ are bi-reachable in \mathcal{V}_1 .

Second, consider a register-less DVASS $\mathcal{V}_1 = (L_1, \emptyset, H_1, P_1, T_1)$ and two configurations $\ell(\mathbf{v}) = q(\mathbf{u} \oplus \mathbf{w})$ and $\ell'(\mathbf{v}') = q'(\mathbf{u}' \oplus \mathbf{w}')$, where $\mathbf{u}, \mathbf{u}' : H_1 \rightarrow_{\text{fin}} \mathbb{N}$ and $\mathbf{w}, \mathbf{w}' : P_1 \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{N}$. We argue that w.l.o.g. we can assume $\mathbf{w} = \mathbf{w}' = \mathbf{0}$. Let $S = \text{SUPP}(\mathbf{v}) \cup \text{SUPP}(\mathbf{v}')$ be the set of those atoms which appear in \mathbf{w} or \mathbf{w}' . Intuitively, we move the set $P_1 \times S$ to plain places. We take $H_2 = H_1 \cup P_1 \times S$ as plain places, and consider atoms $\mathbb{A}' = \mathbb{A} \setminus S$ instead of \mathbb{A} . Clearly,

$$H_1 \cup (P_1 \times \mathbb{A}) = H_2 \cup (P_1 \times \mathbb{A}')$$

31:20 Bi-Reachability in Petri Nets with Data

and therefore we may take the same transitions T_1 as transitions of the new DVASS $\mathcal{V}_2 = (L_1, \emptyset, H_2, P_1, T_1)$. As S is finite, the set T_1 is still orbit-finite with respect to $\text{Aut}(\mathbb{A}')$.

▷ **Claim 32.** The configurations $\ell(\mathbf{v})$, $\ell'(\mathbf{v}')$ are bi-reachable in \mathcal{V}_1 if and only if $\ell(\mathbf{v})$, $\ell'(\mathbf{v}')$ are bi-reachable in \mathcal{V}_2 .

In the last third step, consider a DVASS $\mathcal{V} = (L_2, \emptyset, H_2, P_2, T_2)$ without registers and two configurations $\ell(\mathbf{u})$ and $\ell'(\mathbf{u}')$, where $\mathbf{u}, \mathbf{u}' : H_2 \rightarrow_{\text{fin}} \mathbb{N}$. We eliminate the initial and final values \mathbf{u}, \mathbf{u}' on plain places in a classical way, by introducing new initial and final locations $\bar{\ell}, \bar{\ell}'$ and adding to T_2 the following four transitions, and thus defining $\mathcal{V}_3 = (L_2 \cup \{\bar{\ell}, \bar{\ell}'\}, \emptyset, H_2, P_2, T_3)$:

$$(\bar{\ell}, \mathbf{u}, \ell) \quad (\ell', -\mathbf{u}', \bar{\ell}') \quad (\ell, -\mathbf{u}, \bar{\ell}) \quad (\bar{\ell}', \mathbf{u}', \ell'). \quad (15)$$

▷ **Claim 33.** The configurations $\ell(\mathbf{u})$, $\ell'(\mathbf{u}')$ are bi-reachable in \mathcal{V}_2 if and only if $\bar{\ell}(\mathbf{0})$, $\bar{\ell}'(\mathbf{0})$ are bi-reachable in \mathcal{V}_3 .

Indeed, a run $\ell(\mathbf{u}) \rightarrow \ell'(\mathbf{u}')$ in \mathcal{V}_2 extended with the first two transitions in (15) yields a run $\bar{\ell}(\mathbf{0}) \rightarrow \bar{\ell}'(\mathbf{0})$ in \mathcal{V}_3 , and likewise a run $\ell'(\mathbf{u}')$ in \mathcal{V}_2 extended with the last two transitions in (15) yields a run $\bar{\ell}'(\mathbf{0}) \rightarrow \bar{\ell}(\mathbf{0})$ in \mathcal{V}_3 . Conversely, consider a run $\pi : \bar{\ell}(\mathbf{0}) \rightarrow \bar{\ell}'(\mathbf{0})$ in \mathcal{V}_3 . It necessarily starts with the first transition in (15), and ends with the second one. If transitions (15) are used elsewhere in π , they are necessarily used in pairs, namely the second one followed immediately by the fourth one, or the third one is followed immediately by the first one. Effects of each such pair cancel out, and thus each pair can be safely removed from π . Finally, removing the first and the last transition makes π into a run $\ell(\mathbf{u}) \rightarrow \ell'(\mathbf{u}')$ in \mathcal{V}_2 , as required. Likewise we transform a run $\bar{\ell}'(\mathbf{0}) \rightarrow \bar{\ell}(\mathbf{0})$ in \mathcal{V}_3 . ◀

B Proofs for Section 4 (Toolset)

► **Lemma 10.** \sqsubseteq is a WQO on configurations.

Proof. Recall the sets of states $Q = L \times (\mathcal{R} \rightarrow (\mathbb{A} \cup \{\perp\}))$ and configurations $\text{CONF} = Q \times ((H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{N})$. The quasi-order \sqsubseteq is a WQO on the set of nonnegative vectors $P \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{N}$, as it is quasi-order-isomorphic to $M(P \rightarrow_{\text{fin}} \mathbb{Z})$, the set of finite multisets of finite vectors from $P \rightarrow_{\text{fin}} \mathbb{Z}$, ordered by multiset inclusion. Furthermore, \sqsubseteq is a WQO on $(H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{N}$, as it is quasi-order-isomorphic to the Cartesian product $(H \rightarrow_{\text{fin}} \mathbb{N}) \times (P \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{N})$ of two WQO's, and Cartesian product preserves WQO.

Every register valuation $\nu : \mathcal{R} \rightarrow (\mathbb{A} \cup \{\perp\})$ may be seen as an \mathcal{R}' -vector, where $\mathcal{R}' = \nu^{-1}(\mathbb{A})$ is the set of non-empty registers, namely $\hat{\nu} = \oplus \{(r, a) \mid \nu(r) = a \neq \perp\} : \mathcal{R}' \rightarrow_{\text{fin}} \mathbb{N}$. We use this fact to argue that \sqsubseteq is a WQO on $Y = (\mathcal{R} \rightarrow (\mathbb{A} \cup \{\perp\})) \times ((H \cup P \times \mathbb{A}) \rightarrow_{\text{fin}} \mathbb{N})$. Indeed, we split this set into $2^{|\mathcal{R}'|}$ subsets, determined by non-empty registers, i.e., for every subset $\mathcal{R}' \subseteq \mathcal{R}$ we consider a subset

$$C_{\mathcal{R}'} := \{(\nu, \mathbf{v}) \mid \nu^{-1}(\mathbb{A}) = \mathcal{R}'\} \subseteq Y.$$

For every fixed \mathcal{R}' , the set $C_{\mathcal{R}'}$ is essentially a subset of $H \cup (P \cup \mathcal{R}') \times \mathbb{A} \rightarrow_{\text{fin}} \mathbb{N}$, due to the bijection $(\nu, \mathbf{v}) \mapsto \hat{\nu} \oplus \mathbf{v}$, containing those vectors which use exactly one generator from $\mathcal{R}' \times \mathbb{A}$. Therefore $C_{\mathcal{R}'}$ is a WQO. In consequence, Y is a WQO too, as finite sums preserve WQO.

Finally, the set $\text{CONF} = L \times Y$ is a WQO, as Cartesian product of the finite set L and a WQO. ◀

Minimising the Probabilistic Bisimilarity Distance

Stefan Kiefer  

Department of Computer Science, University of Oxford, UK

Qiyi Tang  

Department of Computer Science, University of Liverpool, UK

Abstract

A labelled Markov decision process (MDP) is a labelled Markov chain with nondeterminism; i.e., together with a strategy a labelled MDP induces a labelled Markov chain. The model is related to interval Markov chains. Motivated by applications to the verification of probabilistic noninterference in security, we study problems of minimising probabilistic bisimilarity distances of labelled MDPs, in particular, whether there exist strategies such that the probabilistic bisimilarity distance between the induced labelled Markov chains is less than a given rational number, both for memoryless strategies and general strategies. We show that the distance minimisation problem is $\exists\mathbb{R}$ -complete for memoryless strategies and undecidable for general strategies. We also study the computational complexity of the qualitative problem about making the distance less than one. This problem is known to be NP-complete for memoryless strategies. We show that it is EXPTIME-complete for general strategies.

2012 ACM Subject Classification Theory of computation \rightarrow Program verification; Theory of computation \rightarrow Models of computation; Mathematics of computing \rightarrow Probability and statistics

Keywords and phrases Markov decision processes, Markov chains

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.32

Related Version *Full Version:* <https://arxiv.org/abs/2406.19830> [18]

Funding This work has been supported in part by the Engineering and Physical Sciences Research Council (EPSRC) through grant EP/X042596/1.

Acknowledgements We thank the referees for their constructive feedback.

1 Introduction

Given a model of computation (e.g., finite automata), and two instances of it, are they semantically equivalent (e.g., do the automata accept the same language)? Such *equivalence* problems can be viewed as a fundamental question for almost any model of computation. As such, they permeate computer science, in particular, theoretical computer science.

In *labelled Markov chains (LMCs)*, which are Markov chains whose states (or, equivalently, transitions) are labelled with an observable letter, there are two natural and very well-studied versions of equivalence, namely *trace (or language) equivalence* and *probabilistic bisimilarity*.

The *trace equivalence* problem has a long history, going back to Schützenberger [28] and Paz [21] who studied *weighted* and *probabilistic* automata, respectively. Those models generalise LMCs, but the respective equivalence problems are essentially the same. For LMCs, trace equivalence asks if the same label sequences have the same probabilities in the two LMCs. It can be extracted from [28] that equivalence is decidable in polynomial time, using a technique based on linear algebra; see also [32, 9].

Probabilistic bisimilarity is an equivalence that was introduced by Larsen and Skou [20]. It is finer than trace equivalence, i.e., probabilistic bisimilarity implies trace equivalence. A similar notion for Markov chains, called *lumpability*, can be traced back at least to the classical text by Kemeny and Snell [15]. Probabilistic bisimilarity can also be computed in



© Stefan Kiefer and Qiyi Tang;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 32; pp. 32:1–32:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

polynomial time [1, 7, 33]. Indeed, in practice, computing the bisimilarity quotient is fast and has become a backbone for highly efficient tools for probabilistic verification such as PRISM [19] and STORM [12].

Numerous quantitative generalisations of this behavioural equivalence have been proposed, the probabilistic bisimilarity distance due to Desharnais et al. [8] being the most notable one. This distance can be at most 1, and is 0 if and only if the LMCs are probabilistic bisimilar. It was shown in [5] that the distance can be computed in polynomial time.

In this paper, we study distance minimisation problems for (*labelled*) *Markov decision processes* (MDPs), which are LMCs plus nondeterminism; i.e., each state may have several *actions* (or “moves”) one of which is chosen by a controller, potentially randomly. An MDP and a controller *strategy* together induce an LMC (potentially with infinite state space, depending on the complexity of the strategy). We consider both general strategies and the more restricted memoryless ones. There are good reasons to consider memoryless strategies, particularly their naturalness and simplicity in implementations, and their connection to *interval Markov chains* (see, e.g., [14, 6]) and *parametric MDPs* (see, e.g., [11, 35]). There are also good reasons to consider general unrestricted strategies, primarily their naturalness (in their definition for MDPs) and their generality. The latter is important particularly for security applications, see below, where general strategies can make programs more secure, in a precise, quantitative sense.

Let us elaborate on the connection to security. *Noninterference* refers to an information-flow property of a program, stipulating that information about *high* data (i.e., data with high confidentiality) may not leak to *low* (i.e., observable) data, or, quoting [25], “that a program is secure whenever varying the initial values of high variables cannot change the low-observable (observable by the attacker) behaviour of the program”. It was proposed in [25] to reason about *probabilistic* noninterference in probabilistic multi-threaded programs by proving probabilistic bisimilarity; see also [29, 22]. More precisely, probabilistic noninterference is established if it can be shown that any two states that differ only in high data are probabilistic bisimilar, as then an attacker who only observes the low part of a state learns nothing about the high part. The observable behaviour of a multi-threaded program depends strongly on the *scheduler*, which in this context amounts to a strategy in the corresponding MDP.

Nevertheless, ensuring perfect (probabilistic) noninterference proves challenging, and a certain degree of information leakage may be acceptable [13, 24]. In such scenarios, where (probabilistic) bisimilarity might not hold under any scheduler, turning to bisimilarity distances allows us to estimate the security degree of a system under different schedulers. The smaller the distance, the more secure the system. Therefore, we would like to devise schedulers that minimise the probabilistic bisimilarity distances.

Some qualitative problems have already been studied in previous work. Concerning memoryless strategies, it was shown in [16] that the bisimilarity equivalence problem, i.e., whether strategies exist to make the distance 0, is NP-complete. Similarly, it was also shown in [16] that the problem whether memoryless strategies exist to make the distance less than one is NP-complete; cf. Table 1. The bisimilarity *inequivalence* problem, i.e., whether strategies exist to make the distance greater than 0, can be decided in polynomial time for memoryless strategies [16].

Concerning general strategies, the bisimilarity equivalence and inequivalence problems were studied in [17]. It was shown there that these problems are EXPTIME-complete and in P, respectively.

It remained open whether the existence of strategies to make the distance less than one is decidable for general strategies. We show that the distance less than one problem for general strategies is decidable. In fact, it is EXPTIME-complete, and therefore the problem has the

■ **Table 1** Summary of results on distance minimisation problems.

Problem	Memoryless Strategy	General Strategy
distance = 0	NP-complete [16]	EXPTIME-complete [17]
distance < 1	NP-complete [16]	EXPTIME-complete (Section 6)
distance < θ	$\exists\mathbb{R}$ -complete (Section 4)	undecidable (Section 5)

same complexity as the bisimilarity equivalence problem for general strategies. To obtain this result, we prove a tight connection between the distance less than one problem and the bisimilarity equivalence problem: loosely speaking, whenever there are general strategies for two states to have distance less than one, the two states can reach a pair of states whose distance can be made 0, thus probabilistic bisimilar. This connection is natural and known for *finite* labelled Markov chains, but nontrivial to establish in general.

We also study *quantitative* distance minimisation problems: do there exist memoryless (resp. general) strategies for two given MDPs such that the induced LMCs have distance less than a given threshold? We show that the distance minimisation problem is $\exists\mathbb{R}$ -complete for memoryless strategies and undecidable for general strategies. Here, $\exists\mathbb{R}$ refers to the class of problems that are many-one reducible to the existential theory of the reals; it is known that $\text{NP} \subseteq \exists\mathbb{R} \subseteq \text{PSPACE}$.

The rest of the paper is organised as follows. We give preliminaries in Section 2. In Section 3 we discuss probabilistic noninterference. In Sections 4 and 5 we prove our results on the quantitative distance minimisation problems for general strategies and memoryless strategies, respectively. We study the distance less than one problem for general strategies in Section 6. We conclude in Section 7. Missing proofs can be found in the full version of this paper [18].

2 Preliminaries

We write \mathbb{N} for the set of nonnegative integers. Let S be a finite set. We denote by $\text{Distr}(S)$ the set of probability distributions on S . For a distribution $\mu \in \text{Distr}(S)$ we write $\text{support}(\mu) = \{s \in S \mid \mu(s) > 0\}$ for its support. We denote the Dirac distribution concentrated on an element $s \in S$ by $\mathbf{1}_s$, that is, $\mathbf{1}_s(s) = 1$ and $\mathbf{1}_s(t) = 0$ for all $t \neq s$. We denote by $\rho(i)$ the i -th element of a sequence ρ . We denote the least fixed point of a function f by $\mu.f$.

A *labelled Markov chain* (LMC) is a quadruple $\langle S, L, \tau, \ell \rangle$ consisting of a nonempty countable set S of states, a nonempty finite set L of labels, a transition function $\tau : S \rightarrow \text{Distr}(S)$, and a labelling function $\ell : S \rightarrow L$. We denote by $\tau(s)(t)$ the transition probability from s to t . Similarly, we denote by $\tau(s)(E) = \sum_{t \in E} \tau(s)(t)$ the transition probability from s to $E \subseteq S$. We require the LMCs to be finitely branching, that is, $|\text{support}(\tau(s))|$ is finite for every $s \in S$.

An equivalence relation $R \subseteq S \times S$ is a *probabilistic bisimulation* if for all $(s, t) \in R$, $\ell(s) = \ell(t)$ and $\tau(s)(E) = \tau(t)(E)$ for each R -equivalence class E . *Probabilistic bisimilarity*, denoted by $\sim_{\mathcal{M}}$ (or \sim when \mathcal{M} is clear), is the largest probabilistic bisimulation.

The probabilistic bisimilarity distance, a pseudometric on LMCs, was first defined by Desharnais, Gupta, Jagadeesan and Panangaden in [8]. Their definition is based on a real-valued modal logic. This logic can be viewed as a function which maps a formula f of the logic and a state s of the LMC to a real number $f(s) \in [0, 1]$. The distance $d(s, t)$ between two states s, t is defined as $\sup_f |f(s) - f(t)|$. Later, Van Breugel and Worrell [34]

32:4 Minimising the Probabilistic Bisimilarity Distance

defined probabilistic bisimilarity distances on LMCs as a fixed point of a function. They showed that their pseudometric coincides with the one defined in [8]. In this paper, we use the definition from [34]. The *probabilistic bisimilarity distance*, denoted by $d_{\mathcal{M}}$ (or d when \mathcal{M} is clear), is a function from $S \times S$ to $[0, 1]$, that is, an element of $[0, 1]^{S \times S}$. It is the least fixed point of the following function:

$$\Delta(e)(s, t) = \begin{cases} 1 & \text{if } \ell(s) \neq \ell(t) \\ \min_{\omega \in \Omega(\tau(s), \tau(t))} \sum_{u, v \in S} \omega(u, v) e(u, v) & \text{otherwise} \end{cases}$$

where the set $\Omega(\mu, \nu)$ of *couplings* of $\mu, \nu \in \text{Distr}(S)$ is defined as $\Omega(\mu, \nu) = \{ \omega \in \text{Distr}(S \times S) \mid \sum_{t \in S} \omega(s, t) = \mu(s) \wedge \sum_{s \in S} \omega(s, t) = \nu(t) \}$. Note that a coupling $\omega \in \Omega(\mu, \nu)$ is a joint probability distribution with marginals μ and ν (see, e.g., [2, page 260-262]). For all $s, t \in S$, $s \sim t$ if and only if s and t has probabilistic bisimilarity distance zero [8, Theorem 1].

A (*labelled*) *Markov decision process* (MDP) is a tuple $\langle S, Act, L, \varphi, \ell \rangle$ consisting of a finite set S of states, a finite set Act of actions, a finite set L of labels, a partial function $\varphi : S \times Act \rightarrow \text{Distr}(S)$ denoting the probabilistic transition, and a labelling function $\ell : S \rightarrow L$. The set of available actions in a state s is $Act(s) = \{ m \in Act \mid \varphi(s, m) \text{ is defined} \}$.

A path is a sequence $\rho = s_0 m_1 s_1 \cdots m_n s_n$ such that $\varphi(s_i, m_{i+1})$ is defined and $\varphi(s_i, m_{i+1})(s_{i+1}) > 0$ for all $0 \leq i < n$. The last state of ρ is $\text{last}(\rho) = s_n$. Let $\text{Paths}(\mathcal{D})$ denote the set of paths in \mathcal{D} .

A (general) strategy for an MDP is a function $\alpha : \text{Paths}(\mathcal{D}) \rightarrow \text{Distr}(Act)$ that given a path ρ , returns a probability distribution on the available actions at the last state of ρ , $\text{last}(\rho)$. A memoryless strategy depends only on $\text{last}(\rho)$; so we can identify a memoryless strategy with a function $\alpha : S \rightarrow \text{Distr}(Act)$ that given a state s , returns a probability distribution on the available actions at that state.

A general strategy α for \mathcal{D} induces an LMC $\mathcal{D}(\alpha) = \langle \mathcal{P}, L, \tau, \ell' \rangle$, where $\mathcal{P} \subseteq \text{Paths}(\mathcal{D})$. For $\rho \in \mathcal{P}$, we have $\tau(\rho)(\rho m t) = \alpha(\rho)(m) \varphi(s, m)(t)$ and $\ell'(\rho) = \ell(s)$ where $s = \text{last}(\rho)$ and $m \in Act(s)$.

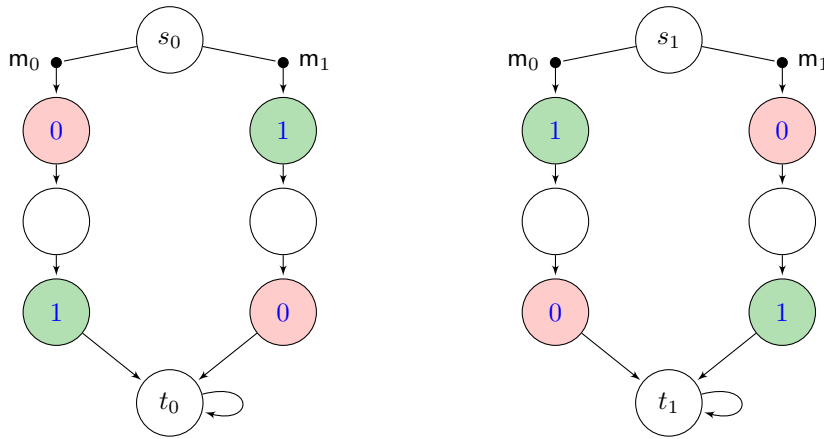
3 Probabilistic Noninterference

In this section we provide examples that show some challenges in distance minimisation and illustrate the relation between distance minimisation and probabilistic noninterference in security. As described in the introduction, we are interested in schedulers that minimise the information leakage.

► **Example 1.** We borrow an example from [25, Section 4] and [17, Section 3]. Consider the following simple program composed of two threads, involving a *high* boolean variable h (high confidentiality) and a *low* boolean variable l (observable):

$$l := h \quad | \quad l := \neg h$$

The vertical bar $|$ separates two threads. The order in which the threads are executed is determined by a scheduler. We assume that assignments to the value of variable l are visible. One may model the program as the following MDP in Figure 1. Here, s_0 and s_1 correspond to initial states with $h = 0$ and $h = 1$, respectively. The two actions in the MDP, m_0 and m_1 , correspond to the two possible orders of execution: action m_0 models the choice of executing $l := h$ first, followed by $l := \neg h$, while m_1 models the reverse order. The different colours



■ **Figure 1** The program from Example 1 as an MDP. The states s_0 and s_1 have two available actions, m_0 and m_1 . The default action m for the other states is omitted. Different colours (state labels) indicate the distinct values of the low data. Throughout the paper, transition probabilities out of each action are one unless explicitly specified.

represent the distinct values of the low, observable data. For instance, in state s_0 , if the scheduler selects m_0 (the left branch of s_0), then l becomes 0 after executing $l := h$ and 1 after executing $l := \neg h$. All transitions are with probability one. A memoryless strategy that chooses actions m_0, m_1 uniformly at random (i.e., with probability 0.5 each) makes s_0, s_1 probabilistic bisimilar; i.e., $d(s_0, s_1) = 0$ under this strategy. \lrcorner

► **Example 2.** Consider the following variant of Example 1.

```
repeat
   $l := h \mid l := \neg h$ 
until  $\text{coin}(p) \vee h$ 
```

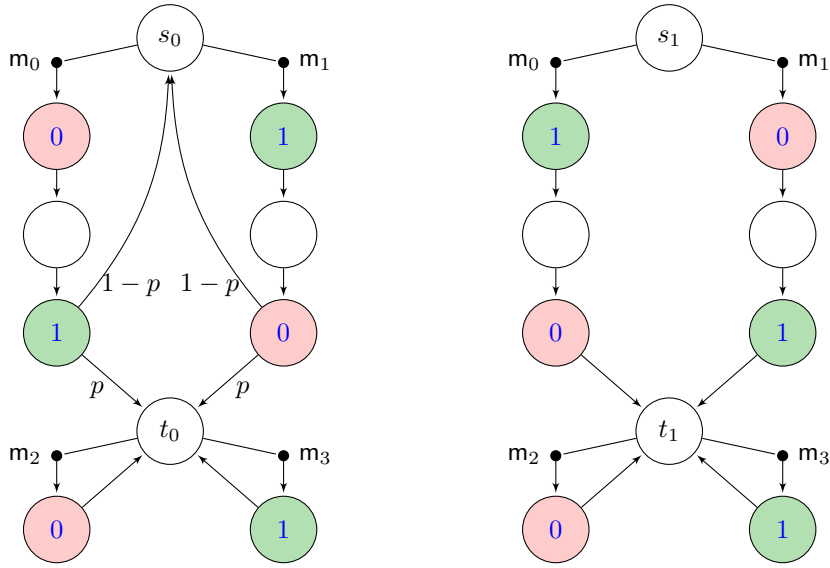
Here, $\text{coin}(p)$, for a fixed parameter $p \in [0, 1]$, models a biased coin that returns *true* with probability p and *false* with probability $1 - p$. One may model the program as the MDP in Figure 2, *except* that t_0, t_1 are sinks, as in Example 1. The value of h influences the termination condition of the loop and therefore “leaks” (with probability $1 - p$). As a result, under the optimal (in terms of minimising the distance) strategy, which is the same as in Example 1, we have now $d(s_0, s_1) = 1 - p$. The smaller p , the “worse” the leak. \lrcorner

The following example shows that general strategies may be needed for optimal security.

► **Example 3.** In order to mitigate the leak from Example 2, one might extend the program as follows, so that the scheduler is given an opportunity to disguise the fact that the program with $h = 1$ tends to terminate earlier than the program with $h = 0$:

```
repeat
   $l := h \mid l := \neg h$ 
until  $\text{coin}(p) \vee h$ 
repeat forever
   $l := 0 \oplus l := 1$ 
```

Here, \oplus stands for a nondeterministic choice, to be made by the scheduler, where exactly one of the instructions $l := 0$ and $l := 1$ is executed. In Figure 2, this corresponds to taking actions m_2 and m_3 , respectively. One can show that the optimal *memoryless* strategy



■ **Figure 2** The program from Example 3 as an MDP. The states s_0 and s_1 have two available actions, m_0 and m_1 . The states t_0 and t_1 also have two available actions, m_2 and m_3 . Different colours (state labels) indicate the distinct values of the low data.

chooses between m_0 and m_1 uniformly at random (as before), and also chooses between m_2 and m_3 uniformly at random. Under this strategy we have $d(s_0, t_1) = 0.5 + 0.5(1-p)d(s_0, t_1)$, implying $d(s_0, t_1) = \frac{1}{1+p}$, and thus $d(s_0, s_1) = (1-p)d(s_0, t_1) = \frac{1-p}{1+p}$, which is, for $p \in (0, 1)$, smaller (i.e., better) than the distance achievable in Example 2.

However, there is a general strategy α , not memoryless, that perfectly disguises when the first loop is exited. This strategy α chooses between m_0 and m_1 uniformly at random (as before). When the execution path visits t_0 or t_1 for the i th time, $i \geq 1$, then, if i is odd, α chooses between m_2 and m_3 uniformly at random, and if i is even, α chooses the action that was not taken upon the $(i-1)$ th visit of t_0 or t_1 . Under this strategy α we have $d(s_0, s_1) = 0$, i.e., s_0 and s_1 are probabilistic bisimilar. ┘

4 Memoryless Strategies: Distance Minimisation

In this section we consider the *memoryless distance minimisation problem* which, given an MDP, two states s_1, s_2 of the MDP, and a rational number θ , asks whether there is a memoryless strategy α such that $d(s_1, s_2) < \theta$ holds in the LMC induced by α .

We show that the memoryless distance minimisation problem is $\exists\mathbb{R}$ -complete. We prove the lower and upper bound in Theorems 7 and 8, respectively.

The *existential theory of the reals*, ETR, is the set of valid formulas of the form

$$\exists x_1 \dots \exists x_n R(x_1, \dots, x_n),$$

where R is a Boolean combination of comparisons of the form $p(x_1, \dots, x_n) \sim 0$, in which $p(x_1, \dots, x_n)$ is a multivariate polynomial (with rational coefficients) and $\sim \in \{<, >, \leq, \geq, =, \neq\}$. The complexity class $\exists\mathbb{R}$ [27] consists of those problems that are many-one reducible to ETR in polynomial time. Since ETR is NP-hard and in PSPACE [4, 23], we have $\text{NP} \subseteq \exists\mathbb{R} \subseteq \text{PSPACE}$.

To prove that the memoryless distance minimisation problem is $\exists\mathbb{R}$ -hard (Theorem 7), we proceed via a sequence of reductions, represented by the following lemmas, Lemmas 4–6.

► **Lemma 4.** *The following problem is $\exists\mathbb{R}$ -complete: given a multivariate polynomial $p : \mathbb{R}^n \rightarrow \mathbb{R}$ of (total) degree at most 6, does there exist $x \in \mathbb{R}^n$ with $p(x) < 0$? The problem remains $\exists\mathbb{R}$ -complete under the promise that if there is x with $p(x) < 0$ then there is x' with $p(x') < 0$ and $\|x'\| < 1$ (where $\|\cdot\|$ denotes the Euclidean norm).*

Proof. Membership in $\exists\mathbb{R}$ is clear. It remains to prove $\exists\mathbb{R}$ -hardness. It is shown in [26, Lemma 3.9] that the following problem is $\exists\mathbb{R}$ -complete: given multivariate polynomials $f_1, \dots, f_s : \mathbb{R}^n \rightarrow \mathbb{R}$, each of degree at most 2, does there exist $x \in \mathbb{R}^n$ with $\|x\| < 1$ such that $\bigwedge_{i=1}^s (f_i(x) = 0)$? It follows from the proof that the problem remains $\exists\mathbb{R}$ -complete under the promise that $\bigwedge_i f_i(x) = 0$ implies $\|x\| < 1$. We reduce from this promise problem. Let $f_1, \dots, f_s : \mathbb{R}^n \rightarrow \mathbb{R}$, each of degree at most 2, such that for all $x \in \mathbb{R}^n$ we have that $\bigwedge_i f_i(x) = 0$ implies $\|x\| < 1$. Define the polynomial $q : \mathbb{R}^n \rightarrow \mathbb{R}$ by $q(x) := \sum_{i=1}^s f_i(x)^2$. Clearly, $q(x) \geq 0$ always holds, and we have $q(x) = 0$ if and only if $\bigwedge_i f_i(x) = 0$. Consider the two sets $\{(q(x), x) \in \mathbb{R}^{n+1} \mid \|x\| \leq 1\}$ and $\{(0, x) \in \mathbb{R}^{n+1} \mid \|x\| \leq 1\}$. If q has a root x , then the two sets overlap in the point $(0, x)$; otherwise, by [27, Corollary 3.4], they have distance at least $2^{2^{-k}}$, where k is a natural number whose unary representation can be computed in polynomial time. It follows that if $\|x\| \leq 1$ and $q(x) < 2^{2^{-k}}$ then there exists x' such that $q(x') = 0$.

In the following let us use real-valued variables $x_1, \dots, x_n, y_1, \dots, y_k$ and write $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_k)$. Define the polynomial $r : \mathbb{R}^{n+k} \rightarrow \mathbb{R}$ (of degree at most 6) by

$$r(x, y) := (y_1 - 4)^2 + (y_2 - y_1^2)^2 + \dots + (y_k - y_{k-1}^2)^2 + y_k^2 q(x) + \|x\|^2 - 1.$$

Let us also use a real-valued variable z . Define the polynomial $p : \mathbb{R}^{n+k+1}$ (of degree at most 6) by

$$p(x, y, z) := z^6 r\left(\frac{x_1}{z}, \dots, \frac{x_n}{z}, \frac{y_1}{z}, \dots, \frac{y_k}{z}\right).$$

Suppose there is $x \in \mathbb{R}^n$ with $\bigwedge_i f_i(x) = 0$. Then $q(x) = 0$. For $1 \leq i \leq k$, set $y_i := 2^{2^i}$. Then $r(x, y) = \|x\|^2 - 1 < 0$. Set $z > 0$ small enough so that $z^2 (\|x\|^2 + \|y\|^2 + 1) < 1$. For $1 \leq i \leq n$, set $x'_i := x_i z$. For $1 \leq i \leq k$, set $y'_i := y_i z$. Then $p(x', y', z) = z^6 r(x, y) < 0$ and $\|x'\|^2 + \|y'\|^2 + z^2 = z^2 (\|x\|^2 + \|y\|^2 + 1) < 1$.

Towards the other direction, suppose there is $(x', y', z) \in \mathbb{R}^{n+k+1}$ with $p(x', y', z) < 0$. Since p is a polynomial, it is continuous. So we can assume without loss of generality that $z \neq 0$. For $1 \leq i \leq n$, set $x_i := x'_i / z$. For $1 \leq i \leq k$, set $y_i := y'_i / z$. Then $r(x, y) = p(x', y', z) / z^6 < 0$. This implies $y_k^2 q(x) < 1$ and $\|x\| < 1$. Using $r(x, y) < 0$, we show by induction that $y_i \geq 2^{2^{i-1}} + 1$ holds for all $i \in \{1, \dots, k\}$. For the induction base ($i = 1$) we have $(y_1 - 4)^2 \leq 1$. Thus, $y_1 - 4 \geq -1$, and so $y_1 \geq 3 = 2^{2^{1-1}} + 1$. For the step ($1 \leq i \leq k - 1$), suppose that $y_i \geq 2^{2^{i-1}} + 1$. Since $r(x, y) < 0$, we have $(y_{i+1} - y_i^2)^2 \leq 1$, and so

$$y_{i+1} \geq y_i^2 - 1 \geq (2^{2^{i-1}} + 1)^2 - 1 = 2^{2^i} + 2 \cdot 2^{2^{i-1}} \geq 2^{2^i} + 1.$$

Hence, we have shown that $y_k \geq 2^{2^{k-1}} + 1 > 2^{2^{k-1}}$. It follows that $q(x) < 1/y_k^2 < 2^{-2^k}$. Since $\|x\| < 1$, it follows from the argument at the beginning that there exists x' such that $q(x') = 0$ and so $\bigwedge_i f_i(x') = 0$.

32:8 Minimising the Probabilistic Bisimilarity Distance

This completes the hardness proof. Note that by combining the two directions, it follows that if there is $w \in \mathbb{R}^{n+k+1}$ with $p(w) < 0$, then there is $w' \in \mathbb{R}^{n+k+1}$ with $p(w') < 0$ and $\|w'\| < 1$, showing also $\exists\mathbb{R}$ -hardness of the promise version of the problem. \blacktriangleleft

► **Lemma 5.** *The following problem is $\exists\mathbb{R}$ -complete: given a multivariate polynomial $p : \mathbb{R}^n \rightarrow \mathbb{R}$ of degree at most 6, does there exist $x \in [0, 1]^n$ with $p(x) > 0$?*

Proof. Membership in $\exists\mathbb{R}$ is clear. For hardness we reduce from the promise problem from the previous lemma. Let $p : \mathbb{R}^n \rightarrow \mathbb{R}$ be a multivariate polynomial of degree at most 6 such that if there is $x \in \mathbb{R}^n$ with $p(x) < 0$ then there is $x' \in \mathbb{R}^n$ with $p(x') < 0$ and $\|x'\| < 1$. Define the polynomial $q : \mathbb{R}^{2n} \rightarrow \mathbb{R}$ by $q(y_1, \dots, y_n, z_1, \dots, z_n) := -p(y_1 - z_1, \dots, y_n - z_n)$. The degree of q is at most 6. We have to show that there is $x \in \mathbb{R}^n$ with $p(x) < 0$ if and only if there are $y_1, \dots, y_n, z_1, \dots, z_n \in [0, 1]$ with $q(y_1, \dots, y_n, z_1, \dots, z_n) > 0$.

Suppose there are $x_1, \dots, x_n \in \mathbb{R}$ with $p(x_1, \dots, x_n) < 0$. By the property of p we can assume that $x_1^2 + \dots + x_n^2 < 1$. It follows that $x_i \in [-1, 1]$ holds for all i . For all i with $x_i \geq 0$ define $y_i := x_i$ and $z_i := 0$. For all i with $x_i < 0$ define $y_i := 0$ and $z_i := -x_i$. Then we have $x_i = y_i - z_i$ and $y_i, z_i \in [0, 1]$ for all i . Further,

$$q(y_1, \dots, y_n, z_1, \dots, z_n) = -p(y_1 - z_1, \dots, y_n - z_n) = -p(x_1, \dots, x_n) > 0.$$

Towards the other direction, suppose that there are $y_1, \dots, y_n, z_1, \dots, z_n \in [0, 1]$ with $q(y_1, \dots, y_n, z_1, \dots, z_n) > 0$. For all i define $x_i := y_i - z_i$. Then we have

$$p(x_1, \dots, x_n) = p(y_1 - z_1, \dots, y_n - z_n) = -q(y_1, \dots, y_n, z_1, \dots, z_n) < 0,$$

as required. \blacktriangleleft

► **Lemma 6.** *The following problem is $\exists\mathbb{R}$ -complete: given a rational number $\theta \geq 0$ and a multivariate (degree-6) polynomial $p : \mathbb{R}^n \rightarrow \mathbb{R}$ of the form $p(x) = \sum_{j=1}^k f_j(x)$ where each $f_j(x_1, \dots, x_n)$ is a product of a nonnegative coefficient and 6 terms of the form x_i or $(1 - x_i)$, does there exist $x \in [0, 1]^n$ with $p(x) > \theta$?*

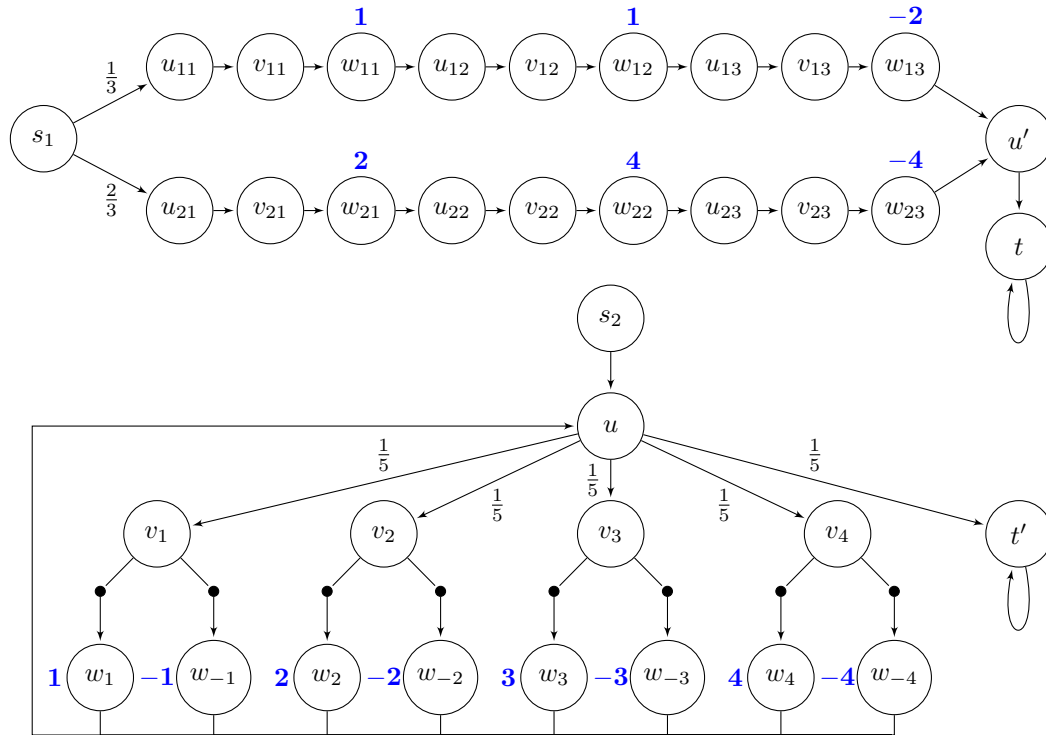
Proof. Membership in $\exists\mathbb{R}$ is clear. Towards hardness, suppose $m : \mathbb{R}^n \rightarrow \mathbb{R}$ is a monomial with a negative coefficient, i.e.,

$$m(x_1, \dots, x_n) = -c \prod_{j=1}^d x_{i_j} \quad \text{for some } c > 0 \text{ and } i_1, \dots, i_d \in \{1, \dots, n\}.$$

Then we have

$$\begin{aligned} m(x_1, \dots, x_n) &= -c \prod_{j=1}^d x_{i_j} = c(1 - x_{i_1}) \prod_{j=2}^d x_{i_j} - c \prod_{j=2}^d x_{i_j} = \dots \\ &= -c + \sum_{k=1}^d c(1 - x_{i_k}) \prod_{j=k+1}^d x_{i_j}. \end{aligned}$$

We reduce from the problem from Lemma 5. Let $p : \mathbb{R}^n \rightarrow \mathbb{R}$ be a multivariate polynomial of degree at most 6. By rewriting each monomial of p that has a negative coefficient using the pattern above, we can write $p(x) = -\theta + q(x)$ for some $\theta \geq 0$ and some $q : \mathbb{R}^n \rightarrow \mathbb{R}$ of the form $q(x) = \sum_{j=1}^k f_j(x)$ where each $f_j(x)$ is a product of a nonnegative coefficient and at most 6 terms of the form x_i or $(1 - x_i)$. As long as there is an $f_j(x_1, \dots, x_n)$ of degree less than 6, we can replace it by the two summands $x_1 f_j(x_1, \dots, x_n)$ and $(1 - x_1) f_j(x_1, \dots, x_n)$. So we can assume that every $f_j(x)$ has the required form. For all $x \in \mathbb{R}^n$ we have that $p(x) > 0$ if and only if $q(x) > \theta$, as required. \blacktriangleleft



■ **Figure 3** An illustration of the proof of Theorem 7. Consider the polynomial p with $p(x_1, x_2, x_3, x_4) = \frac{1}{3}x_1^2(1 - x_2) + \frac{2}{3}x_2x_4(1 - x_4)$. This example polynomial has degree 3 (instead of degree 6 in the proof) to allow for a more succinct picture. The analogous construction from the reduction yields the shown MDP. The labels are written next to the states in blue, unlike the other figures in this paper where we usually use different colours to indicate different state labels. We omit label 0. There is a one-to-one correspondence between an assignment $x \in [0, 1]^4$ and a memoryless strategy $\alpha(x)$ in the MDP. It is such that $d(s_1, s_2) = 1 - \frac{p(x)}{5^4}$, establishing a connection between an evaluation of p and the distance.

To show that the memoryless distance minimisation problem is $\exists\mathbb{R}$ -hard, we reduce from the problem in Lemma 6. We give a brief outline of the reduction. Given a multivariate polynomial $p : \mathbb{R}^n \rightarrow \mathbb{R}$ of the form as in Lemma 6, we construct an MDP with initial states s_1 and s_2 such that each assignment $x \in [0, 1]^n$ corresponds to a memoryless strategy $\alpha(x)$ of the MDP. The distance of s_1 and s_2 in the LMC induced by the memoryless strategy $\alpha(x)$ is $1 - c \cdot p(x)$ where c is a constant. Therefore, there exists $x \in [0, 1]^n$ with $p(x) > \theta$ if and only if there exists a memoryless strategy $\alpha(x)$ such that the distance of s_1 and s_2 is less than $1 - c \cdot \theta$.

► **Theorem 7.** *The memoryless distance minimisation problem is $\exists\mathbb{R}$ -hard.*

Proof. We reduce from the problem from Lemma 6. Let $\theta \geq 0$ and let $p : \mathbb{R}^n \rightarrow \mathbb{R}$ be a multivariate polynomial of the form $p(x) = \sum_{j=1}^m f_j(x)$ where each $f_j(x_1, \dots, x_n)$ is a product of a nonnegative coefficient and 6 terms of the form x_i or $(1 - x_i)$. Let us write $f_j(x_1, \dots, x_n) = c_j \prod_{k=1}^6 x_{\ell(j,k)}$ where each $c_j \geq 0$ and each $\ell(j, k) \in \{-n, \dots, -1, 1, \dots, n\}$ and we use the notation x_{-i} for $i > 0$ to mean $1 - x_i$. We can assume that $\sum_{j=1}^m c_j = 1$ (otherwise, divide θ and each c_j by $\sum_{j=1}^m c_j$).

Construct an MDP which consists of two disjoint parts as follows; see Figure 3 for an illustration. The first part is an LMC. Include states $u_{j,k}, v_{j,k}, w_{j,k}$ for each $j \in \{1, \dots, m\}$ and each $k \in \{1, \dots, 6\}$. Each $u_{j,k}, v_{j,k}$ has label 0, and each $w_{j,k}$ has label $\ell(j, k)$. Each

32:10 Minimising the Probabilistic Bisimilarity Distance

$u_{j,k}$ transitions with probability 1 to $v_{j,k}$. Each $v_{j,k}$ transitions with probability 1 to $w_{j,k}$. Each $w_{j,k}$, except those with $k = 6$, transitions with probability 1 to $u_{j,k+1}$. Include also states s_1 , u' and t with label 0. State s_1 transitions with probability c_j to $u_{j,1}$, for each j . State u' transitions with probability 1 to t . State t is a sink state, that is, it transitions with probability 1 to itself. Also each $w_{j,6}$ transitions with probability 1 to u' .

The second part is an MDP. Include states s_2, u, t' with label 0. State s_2 transitions with probability 1 to u . Include also states v_1, \dots, v_n , each with label 0. State u transitions to each v_i and t' with probability $\frac{1}{n+1}$. State t' is a sink state. Include also states $w_{-n}, \dots, w_{-1}, w_1, \dots, w_n$, where each w_i has label i . Each v_i has two actions, one of which leads with probability 1 to w_i , the other one with probability 1 to w_{-i} . Each w_i transitions with probability 1 to u .

Each assignment $x \in [0, 1]^n$ corresponds to a memoryless strategy $\alpha(x)$ such that in state v_i the memoryless strategy $\alpha(x)$ takes with probability x_i the action that leads to w_i , and $\alpha(x)$ takes with probability $1 - x_i$ the action that leads to w_{-i} . In fact, this mapping α (from an assignment to a memoryless strategy) is a bijection. Fix an arbitrary $x \in [0, 1]^n$ and consider the distances in the LMC induced by $\alpha(x)$. For notational convenience, for any states s, s' let us write $\bar{d}(s, s') := 1 - d(s, s')$. Further, let us write $u_{j,7}$ for u' .

Let $j \in \{1, \dots, m\}$ and $k \in \{1, \dots, 6\}$. Then we have

$$\bar{d}(u_{j,k}, u) = \frac{1}{n+1} \bar{d}(v_{j,k}, v_{|\ell(j,k)|}) = \frac{1}{n+1} x_{\ell(j,k)} \bar{d}(w_{j,k}, w_{\ell(j,k)}) = \frac{1}{n+1} x_{\ell(j,k)} \bar{d}(u_{j,k+1}, u).$$

Since $\bar{d}(u_{j,7}, u) = \bar{d}(u', u) = \frac{1}{n+1}$, it follows

$$\bar{d}(u_{j,1}, u) = \left(\frac{1}{n+1} \right)^7 \prod_{k=1}^6 x_{\ell(j,k)}.$$

Hence,

$$\begin{aligned} \bar{d}(s_1, s_2) &= \sum_{j=1}^m c_j \bar{d}(u_{j,1}, u) = \sum_{j=1}^m c_j \left(\frac{1}{n+1} \right)^7 \prod_{k=1}^6 x_{\ell(j,k)} = \left(\frac{1}{n+1} \right)^7 \sum_{j=1}^m f_j(x) \\ &= \frac{p(x)}{(n+1)^7}. \end{aligned}$$

Thus, we have $p(x) > \theta$ if and only if $\bar{d}(s_1, s_2) > \frac{\theta}{(n+1)^7}$ if and only if $d(s_1, s_2) < 1 - \frac{\theta}{(n+1)^7}$. This completes the hardness proof. \blacktriangleleft

The following theorem, proved in [18, A.1], provides a matching upper bound.

► **Theorem 8.** *The memoryless distance minimisation problem is in $\exists\mathbb{R}$.*

Together with Theorem 7 we obtain:

► **Corollary 9.** *The memoryless distance minimisation problem is $\exists\mathbb{R}$ -complete.*

5 General Strategies: Distance Minimisation

In this section we consider the *general distance minimisation problem* which, given an MDP, two states s_1, s_2 of the MDP, and a rational number θ , asks whether there is a general strategy α such that $d(s_1, s_2) < \theta$ holds in the LMC induced by α .

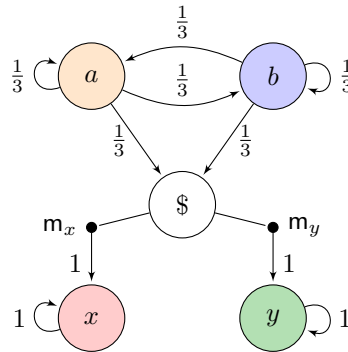
To show that the general distance minimisation problem is undecidable, we establish a reduction from the emptiness problem for probabilistic automata.

A probabilistic automaton is a tuple $\mathcal{A} = \langle Q, q_0, L, \delta, F \rangle$ consisting of a finite set Q of states, an initial state $q_0 \in Q$, a finite set L of letters, a transition function $\delta : Q \times L \rightarrow \text{Distr}(Q)$ assigning to every state and letter a distribution over states, and a set F of final states. We also extend δ to words, by letting $\delta(q_0, \varepsilon) = \mathbf{1}_{q_0}$ and $\delta(q_0, \sigma w) = \sum_{q \in Q} \delta(q_0, \sigma)(q) \delta(q, w)$ for $\sigma \in L$ and $w \in L^*$. For a state $q \in Q$, \mathcal{A}_q is the probabilistic automaton obtained from \mathcal{A} by making q the initial state.

We write $\Pr_{\mathcal{A}}(w) = \sum_{q \in F} \delta(q_0, w)(q)$ to denote the probability that \mathcal{A} accepts a word w . The emptiness problem asks, given a probabilistic automaton \mathcal{A} , whether there exists a word w such that $\Pr_{\mathcal{A}}(w) > \frac{1}{2}$ holds. The probabilistic automaton \mathcal{A} is called empty if no such word exists. This problem is known to be undecidable [10, 21], even for probabilistic automata with only two letters [3]¹.

Let $\mathcal{A} = \langle Q, q_0, L, \delta, F \rangle$ be a probabilistic automaton; without loss of generality we assume that $q_0 \notin F$ and $L = \{a, b\}$. We construct an MDP \mathcal{D} with states s_1 and s_2 and a number θ such that \mathcal{A} is nonempty if and only if there is a general strategy such that $d(s_1, s_2) < \theta$ in the induced LMC.

Let us first outline the idea of the construction. Our MDP includes the part shown in Figure 4, where after a random word $w \in L^*$ is produced, the strategy must choose between taking the transition to x or to y . Lemma 10 below characterises the distance of s_1 and s_2 under strategy α in terms of α and $\Pr_{\mathcal{A}}$. It follows from Lemma 10 that the following strategy minimises the distance: if the random word w satisfies $\Pr_{\mathcal{A}}(w) \leq \frac{1}{2}$, choose the transition to x ; otherwise choose the transition to y . Setting θ as the distance under the strategy that always chooses the transition to x , we obtain that the distance can be made less than θ if and only if there is a word w with $\Pr_{\mathcal{A}}(w) > \frac{1}{2}$.



■ **Figure 4** The first part of the MDP \mathcal{D} . The \$ state is the only one that has nondeterministic choices: it has two available actions, m_x and m_y . The default action m for the other states is omitted. Different colours indicate different state labels.

We now give the details of the construction. The MDP $\mathcal{D} = \langle S, Act, L', \varphi, \ell \rangle$ consists of two disjoint parts as follows; see Figure 4 and Figure 5. The set of actions is $Act = \{m, m_x, m_y\}$. The set of labels is $L' = \{a, b, \$, x, y\}$.

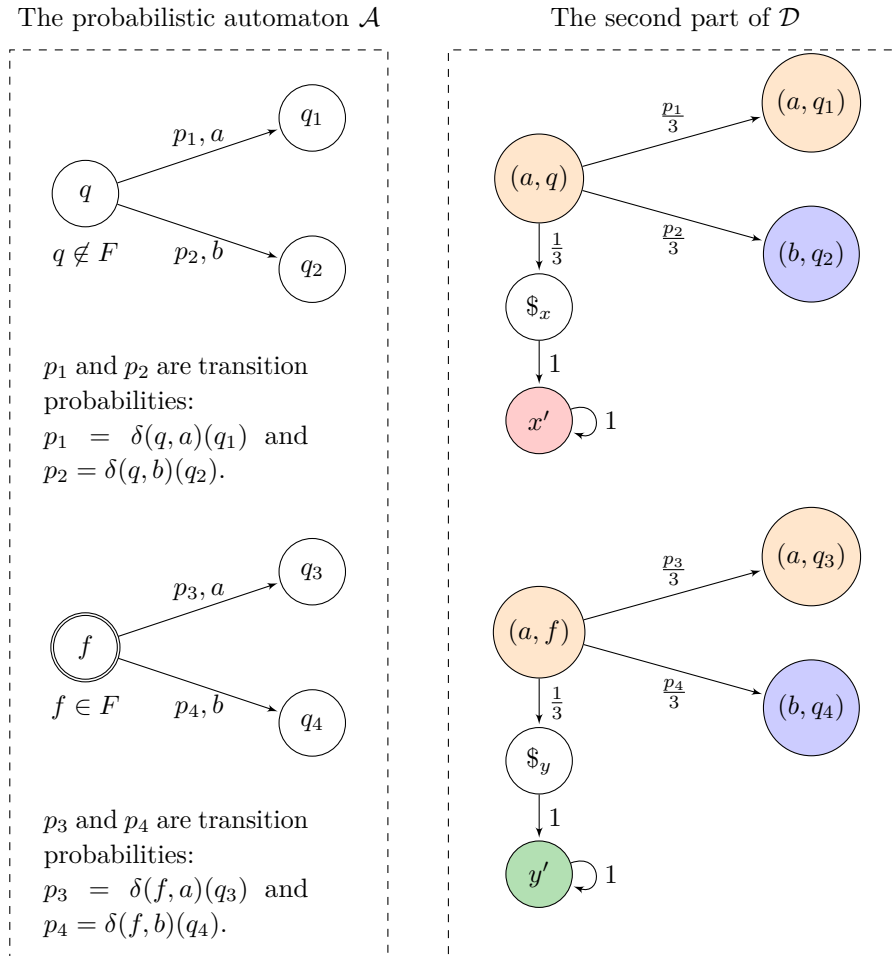
The first part is an MDP shown in Figure 4. Its set of states is $\{a, b, \$, x, y\}$. The state s_1 is defined to be a . The transitions φ are defined as follows:

¹ It is stated in [3, Theorem 2.1] that the emptiness problem with unfixed threshold λ , i.e., whether there exists a word w such that $\Pr_{\mathcal{A}}(w) > \lambda$, is undecidable for probabilistic automata with only two letters. It is easy to adapt the proof to show undecidability of the emptiness problem with fixed threshold $\frac{1}{2}$.

32:12 Minimising the Probabilistic Bisimilarity Distance

- The state a (resp. b) transitions with uniform probability to its three successors a , b and $\$,$ that is, $\varphi(s, m)(a) = \varphi(s, m)(b) = \varphi(s, m)(\$) = \frac{1}{3}$ for $s \in \{a, b\}$.
- The state $\$$ has two actions m_x and m_y ; the action m_x goes with probability 1 to x and the action m_y goes with probability 1 to y . That is, $\varphi(\$, m_x)(x) = \varphi(\$, m_y)(y) = 1$.
- The states x and y are sink states, that is, $\varphi(s, m)(s) = 1$ for $s \in \{x, y\}$.

Each of the states is labelled with its name, that is, $\ell(s) = s$ for $s \in \{a, b, \$, x, y\}$. This sub-MDP “is almost” an MC, in the sense that a strategy α does not influence its behaviour until eventually a transition to x or y is taken. Since a, b, x and y have only one available action, we may omit the default action m in the paths that contain m only. For example, we may write $s_1ab\$$ to represent the path $s_1mam\$,$



■ **Figure 5** The second part of the MDP \mathcal{D} is an LMC, constructed from the probabilistic automaton \mathcal{A} . The default deterministic action m for all states is omitted. The state (b, q) in the MDP \mathcal{D} , where $q \in Q$, has the same transitions as the state (a, q) ; it is labelled with b .

The other part of \mathcal{D} is an LMC constructed from \mathcal{A} as follows; see Figure 5. The set of states is $(L \times Q) \cup \{\$, x, y\}$. The state s_2 is defined to be (a, q_0) .

We describe the transitions of the LMC using the transition function δ of \mathcal{A} . Consider a letter $\sigma \in L$ and a state $q \in Q$. The state (σ, q) with probability $\frac{1}{3}$ simulates the probabilistic automaton \mathcal{A} reading the letter a , and with probability $\frac{1}{3}$ simulates the probabilistic automaton \mathcal{A} reading the letter b . That is, $\varphi((\sigma, q), m)((a, q')) = \frac{1}{3}\delta(q, a)(q')$ and $\varphi((\sigma, q), m)((b, q')) = \frac{1}{3}\delta(q, b)(q')$.

For the remaining probability of $\frac{1}{3}$, we distinguish the following two cases:

- If $q \notin F$, the state (σ, q) transitions to $\$x$ with probability $\frac{1}{3}$, that is, $\varphi((\sigma, q), \mathbf{m})(\$x) = \frac{1}{3}$.
- Otherwise, if $q \in F$, the state (σ, q) transitions to $\$y$ with probability $\frac{1}{3}$, that is, $\varphi((\sigma, q), \mathbf{m})(\$y) = \frac{1}{3}$.

The state $\$x$ (resp. $\$y$) transitions with probability one to the sink state x' (resp. y'). That is, $\varphi(\$x, \mathbf{m})(x') = \varphi(\$y, \mathbf{m})(y') = \varphi(x', \mathbf{m})(x') = \varphi(y', \mathbf{m})(y') = 1$.

A state $(\sigma, q) \in L \times Q$ is labelled with σ . The states $\$x$ and $\$y$ are labelled with $\$$. The states x' and y' are labelled with x and y , respectively.

Given a general strategy α , the next lemma expresses the distance between s_1 and s_2 in terms of α and $\text{Pr}_{\mathcal{A}}$. The proof is technical and can be found in [18, A.2].

► **Lemma 10.** *For any general strategy α , we have*

$$d_{\alpha}(s_1, s_2) = \sum_{w \in L^*} \frac{1}{3^{|w|+1}} ((1 - \text{Pr}_{\mathcal{A}}(w))\alpha(s_1 w \$)(\mathbf{m}_y) + \text{Pr}_{\mathcal{A}}(w)\alpha(s_1 w \$)(\mathbf{m}_x)).$$

Using Lemma 10, we prove the main theorem of this section:

► **Theorem 11.** *The general distance minimisation problem is undecidable.*

Proof. We reduce from the emptiness problem for probabilistic automata. Let $\mathcal{A} = \langle Q, q_0, L, \delta, F \rangle$ be a probabilistic automaton; without loss of generality we assume that $q_0 \notin F$ and $L = \{a, b\}$. Let \mathcal{D} be the MDP constructed from \mathcal{A} shown in Figures 4 and 5.

Let α_x be the memoryless strategy that chooses the action \mathbf{m}_x whenever it is in state $\$$, that is, $\alpha_x(s_1 w \$) = \mathbf{1}_{\mathbf{m}_x}$ for all $w \in L^*$. Let θ be the distance between s_1 and s_2 in the LMC $\mathcal{D}(\alpha_x)$. It can be computed in polynomial time [5]. We show in [18, A.3] that there is a word $w \in L^*$ such that $\text{Pr}_{\mathcal{A}}(w) > \frac{1}{2}$ (\mathcal{A} is nonempty) if and only if there is a general strategy α such that $d_{\alpha}(s_1, s_2) < \theta$ in the induced LMC. ◀

6 General Strategies: Distance Less Than One

In this section, we consider the distance less than one problem which, given an MDP and two states, asks whether there is a general strategy such that the two states have probabilistic bisimilarity distance less than one in the LMC induced by the general strategy. The challenge here is that general strategies induce, in general, LMCs with infinitely many states.

We show that the distance less than one problem is EXPTIME-complete. We prove the upper and lower bound in Sections 6.1 and 6.2, respectively.

6.1 Membership in EXPTIME

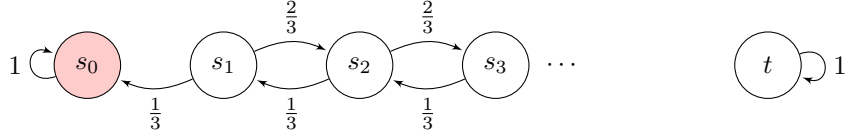
Let $\mathcal{M} = \langle S, L, \tau, \ell \rangle$ be a (possibly infinite) LMC. We partition the set S^2 of state pairs into

$$\begin{aligned} S_0^2 &= \{(s, t) \in S^2 \mid s \sim t\} \\ S_1^2 &= \{(s, t) \in S^2 \mid \ell(s) \neq \ell(t)\} \\ S_?^2 &= S^2 \setminus (S_0^2 \cup S_1^2). \end{aligned}$$

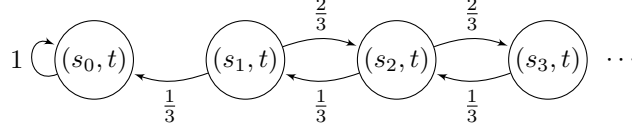
We call $T : S_?^2 \rightarrow \text{Distr}(S^2)$ a *policy* for the LMC if for all $(s, t) \in S_?^2$ we have $T(s, t) \in \Omega(\tau(s), \tau(t))$. We write \mathcal{T} for the set of policies. Given a policy $T \in \mathcal{T}$, the Markov chain $\mathcal{C}_{\mathcal{M}}^T = \langle S^2, \tau' \rangle$ induced by T is defined by

$$\begin{aligned} \tau'((u, v))((u, v)) &= 1 && \text{if } (u, v) \in S_0^2 \cup S_1^2; \\ \tau'((u, v))((x, y)) &= T(u, v)(x, y) && \text{otherwise.} \end{aligned}$$

32:14 Minimising the Probabilistic Bisimilarity Distance



(a) An infinite LMC \mathcal{M} .



(b) The Markov chain $\mathcal{C}_{\mathcal{M}}^T$.

■ **Figure 6** (a) An infinite state LMC \mathcal{M} with an infinite state space $S = \{s_i \mid i \in \{0, 1, 2, \dots\}\} \cup \{t\}$. All states have the same label except s_0 . The states s_0 and t are sink states, that is, $\tau(s_0)(s_0) = \tau(t)(t) = 1$. Each s_i where $i \in \{1, 2, \dots\}$ transitions to s_{i-1} with probability $\frac{1}{3}$ and s_{i+1} with probability $\frac{2}{3}$. (b) The Markov chain $\mathcal{C}_{\mathcal{M}}^T$ induced by an arbitrary policy T in which only the states reachable from (s_1, t) are shown. The shown part of $\mathcal{C}_{\mathcal{M}}^T$ is the same for every policy T .

For $(s, t) \in S^2$ and a set of state pairs $Z \subseteq S^2$ we write $\mathcal{R}_{\mathcal{M}}^T((s, t), Z) \in [0, 1]$ for the probability that in the Markov chain $\mathcal{C}_{\mathcal{M}}^T$ the state (s, t) reaches a state $(u, v) \in Z$.

By [31, Theorem 4, Proposition 5], the following proposition holds.

► **Proposition 12.** *Let $\mathcal{M} = \langle S, L, \tau, \ell \rangle$ be a finite LMC and $s, t \in S$. We have $d(s, t) < 1$ if and only if there exists a policy T such that $\mathcal{R}_{\mathcal{M}}^T((s, t), S_0^2) > 0$.*

The “only if” direction of Proposition 12 does not generally hold for LMCs with infinite state space, as the following example shows.

► **Example 13.** Consider the LMC \mathcal{M} in Figure 6a. Let T be an arbitrary policy for \mathcal{M} . We have $T(s_i, t)(s_{i-1}, t) = \frac{1}{3}$ and $T(s_i, t)(s_{i+1}, t) = \frac{2}{3}$ for all $i \in \{1, 2, \dots\}$. The Markov chain $\mathcal{C}_{\mathcal{M}}^T$ induced by T is shown in Figure 6b; we only show the states that are reachable from (s_1, t) . The shown part of $\mathcal{C}_{\mathcal{M}}^T$ is the same for every policy.

We have $d(s_i, t) = \frac{1}{2^i}$ for all $i \in \{0, 1, 2, \dots\}$. In the Markov chain $\mathcal{C}_{\mathcal{M}}^T$, all state pairs that (s_1, t) can reach have distances greater than zero: for all $i \in \{1, 2, \dots\}$ the pair (s_1, t) can reach (s_i, t) and we have $d(s_i, t) = \frac{1}{2^i} > 0$. \lrcorner

The following theorem follows from [30, Theorem 6.1.7] for LMCs with finite state space. The same proof, see [18, A.4], works for LMCs with infinite state space.

► **Theorem 14.** *Let $\mathcal{M} = \langle S, L, \tau, \ell \rangle$ be an LMC. There is a policy $T \in \mathcal{T}$ such that we have*

$$d(s, t) = \mathcal{R}_{\mathcal{M}}^T((s, t), S_1^2) \leq \mathcal{R}_{\mathcal{M}}^{T'}((s, t), S_1^2) \quad \text{for all } (s, t) \in S^2 \text{ and all } T' \in \mathcal{T}.$$

In short, $d = \min_{T \in \mathcal{T}} \mathcal{R}_{\mathcal{M}}^T(\cdot, S_1^2)$.

The following corollary of Theorem 14 is similar to Proposition 12 but holds even for infinite-state LMCs.

► **Corollary 15.** *Let $\mathcal{M} = \langle S, L, \tau, \ell \rangle$ be an LMC and $s, t \in S$. We have $d(s, t) < 1$ if and only if there exists a policy T such that $\mathcal{R}_{\mathcal{M}}^T((s, t), S_1^2) < 1$. In particular, if there is a policy T with $\mathcal{R}_{\mathcal{M}}^T((s, t), S_0^2) > 0$ then $d(s, t) < 1$.*

Corollary 15 falls short of an “if and only if” connection between distance less than one and bisimilarity. Indeed, as we have seen, Proposition 12 does not always hold in infinite-state LMCs. However, the key technical insight of this section is that a version of Proposition 12 holds for (finite-state) MDPs and general strategies. More precisely, the following proposition characterises the existence of a strategy such that the distance is less than one.

► **Proposition 16.** *Let $\mathcal{D} = \langle S, Act, L, \varphi, \ell \rangle$ be an MDP, and let $s, t \in S$. There exists a strategy α'' with $d_{\mathcal{D}(\alpha'')} (s, t) < 1$ if and only if there are strategies α, α' , a policy T for the LMC $\mathcal{D}(\alpha)$, two states $u, v \in S$ and two paths $\rho_1, \rho_2 \in \text{Paths}(\mathcal{D})$ with $u = \text{last}(\rho_1)$ and $v = \text{last}(\rho_2)$, such that $\mathcal{R}_{\mathcal{D}(\alpha)}^T((s, t), \{(\rho_1, \rho_2)\}) > 0$ and u and v are probabilistically bisimilar in the LMC $\mathcal{D}(\alpha')$.*

The more difficult direction of the proof is the “only if” direction. It is based on Lévy’s zero-one law, several applications of the Bolzano-Weierstrass theorem, and a characterisation of probabilistic bisimilarity in MDPs in terms of an “attacker-defender” game defined [17, Section 3.1].

The starting point of the proof of Proposition 16 is the following statement, which follows from Theorem 14 and Corollary 15 using Lévy’s zero-one law.

► **Corollary 17.** *Let $\mathcal{M} = \langle S, L, \tau, \ell \rangle$ be an LMC and $s, t \in S$ with $d(s, t) < 1$. There exists a policy T such that for all $\varepsilon > 0$ there is $(u, v) \in S^2$ with $d(u, v) \leq \varepsilon$ and $\mathcal{R}_{\mathcal{M}}^T((s, t), \{(u, v)\}) > 0$.*

Proof. Let $\mathcal{M} = \langle S, L, \tau, \ell \rangle$ be an LMC and $s, t \in S$ with $d(s, t) < 1$. By Corollary 15 there exists a policy T such that $\mathcal{R}_{\mathcal{M}}^T((s, t), S_1^2) < 1$. By Lévy’s zero-one law, the probability in $\mathcal{C}_{\mathcal{M}}^T$ is one that a random run $(s_0, t_0)(s_1, t_1) \dots$ started from $(s_0, t_0) = (s, t)$ satisfies one of the following conditions:

1. the sequence $\mathcal{R}_{\mathcal{M}}^T((s_0, t_0), S_1^2), \mathcal{R}_{\mathcal{M}}^T((s_1, t_1), S_1^2), \dots$ converges to 1 and S_1^2 is reached;
2. the sequence $\mathcal{R}_{\mathcal{M}}^T((s_0, t_0), S_1^2), \mathcal{R}_{\mathcal{M}}^T((s_1, t_1), S_1^2), \dots$ converges to 0 and S_1^2 is not reached.

Event 1 can be equivalently characterised by saying that S_1^2 is reached. Since $\mathcal{R}_{\mathcal{M}}^T((s, t), S_1^2) < 1$, Event 2 happens with a positive probability. It follows that in $\mathcal{C}_{\mathcal{M}}^T$ there exists a run $(s_0, t_0)(s_1, t_1) \dots$ started from $(s_0, t_0) = (s, t)$ such that $\mathcal{R}_{\mathcal{M}}^T((s_0, t_0), S_1^2), \mathcal{R}_{\mathcal{M}}^T((s_1, t_1), S_1^2), \dots$ converges to 0. Let $\varepsilon > 0$. Then there exists $(u, v) \in S^2$ such that $\mathcal{R}_{\mathcal{M}}^T((u, v), S_1^2) \leq \varepsilon$ and $\mathcal{R}_{\mathcal{M}}^T((s, t), \{(u, v)\}) > 0$. By Theorem 14 it follows that $d(u, v) \leq \varepsilon$. ◀

► **Example 18.** Consider again Example 13. We have $d(s_1, t) = \frac{1}{2}$. Corollary 17 asserts that there is a policy T such that for all $\varepsilon > 0$, in $\mathcal{C}_{\mathcal{M}}^T$ the pair (s_1, t) can reach $(u, v) \in S^2$ with $d(u, v) \leq \varepsilon$. Indeed, take an arbitrary policy T . Given any $\varepsilon > 0$ choose i with $\frac{1}{2^i} \leq \varepsilon$. Then (s_1, t) can reach (s_i, t) and $d(s_i, t) = \frac{1}{2^i} \leq \varepsilon$. ◻

See [18, A.5] for the rest of the proof of Proposition 16. Proposition 16 is the key to proving the following result.

► **Theorem 19.** *The distance less than one problem is in EXPTIME.*

Proof. Let $\langle S, Act, L, \varphi, \ell \rangle$ be an MDP. Abusing the notation from the beginning of Section 6.1, let us define

$$\begin{aligned} S_0^2 &= \{ (s, t) \in S^2 \mid \exists \alpha' \text{ such that } s, t \text{ are probabilistically bisimilar in } \mathcal{D}(\alpha') \} \\ S_1^2 &= \{ (s, t) \in S^2 \mid \ell(s) \neq \ell(t) \} \\ S_?^2 &= S^2 \setminus (S_0^2 \cup S_1^2). \end{aligned}$$

By [17, Theorem 7] the set S_0^2 can be computed in exponential time. Consider the elements of S^2 as vertices of a directed graph with set of edges

$$E := \{(z, z) \mid z \in S_0^2 \cup S_1^2\} \cup \{((s_1, s_2), (t_1, t_2)) \in S_1^2 \times S^2 \mid \forall i \in \{1, 2\} \exists m_i \in \text{Act}(s_i) : \text{support}(\varphi(s_i, m_i)) \ni t_i\}.$$

After S_0^2 has been computed (in exponential time), the directed graph $G := (S^2, E)$ can be computed in polynomial time, and given two states $s, t \in S$, it can be checked in polynomial time if S_0^2 can be reached from (s, t) in G . It follows from Proposition 16 that this is the case if and only if there exists a strategy α'' with $d_{\mathcal{D}(\alpha'')}(s, t) < 1$. ◀

6.2 EXPTIME-Hardness

Given an MDP and two (initial) states, the *bisimilarity problem* asks whether there is a general strategy such that the two states are probabilistically bisimilar in the induced LMC. The bisimilarity problem was shown EXPTIME-complete in [17, Theorem 7]. We show in [18, A.6] that it can be reduced to the distance less than one problem. This gives us the following theorem.

► **Theorem 20.** *The distance less than one problem is EXPTIME-hard.*

Together with Theorem 19 we obtain:

► **Corollary 21.** *The distance less than one problem is EXPTIME-complete.*

7 Conclusion

Motivated by probabilistic noninterference, a security notion, we have settled the decidability and complexity of the most natural bisimilarity distance minimisation problems of MDPs under memoryless and general strategies.

Specifically, we have proved that the distance minimisation problem for memoryless strategies is $\exists\mathbb{R}$ -complete (which implies, in particular, that it is NP-hard and in PSPACE). In contrast, we have shown that the distance minimisation problem for general strategies is undecidable, reducing from the emptiness problem for probabilistic automata.

We have also shown that it is EXPTIME-complete to decide if there are general strategies to make the probabilistic bisimilarity distance less than one. This extends a result from [17] that the bisimilarity equivalence problem under general strategies is EXPTIME-complete. The key technical link we need here is natural but nontrivial to establish under general strategies: if there are general strategies such that two states have distance less than one, these two states can reach another pair of states which can be made probabilistic bisimilar.

Distance *maximisation* problems also relate to probabilistic noninterference, but in terms of antagonistic schedulers wanting to maximise the information leakage. The decidability and complexity of several distance maximisation problems in MDPs is still open, including the distance equals one problem for general strategies.

References

- 1 Christel Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 50–61, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- 2 Patrick Billingsley. *Probability and measure*. Wiley Series in Probability and Statistics. Wiley, New York, NY, USA, 3rd edition, 1995.
- 3 Vincent D. Blondel and Vincent Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory Comput. Syst.*, 36(3):231–245, 2003. doi:10.1007/S00224-003-1061-2.

- 4 John Canny. Some algebraic and geometric computations in PSPACE. In *STOC*, pages 460–467, 1988.
- 5 Di Chen, Franck van Breugel, and James Worrell. On the complexity of computing probabilistic bisimilarity. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7213 of *Lecture Notes in Computer Science*, pages 437–451. Springer, 2012. doi:10.1007/978-3-642-28729-9_29.
- 6 Benoît Delahaye. Consistency for parametric interval Markov chains. In Étienne André and Goran Frehse, editors, *2nd International Workshop on Synthesis of Complex Parameters, SynCoP 2015, April 11, 2015, London, United Kingdom*, volume 44 of *OASICS*, pages 17–32. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.
- 7 Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.
- 8 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled Markov systems. In Jos Baeten and Sjouke Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 258–273, Eindhoven, The Netherlands, August 1999. Springer-Verlag.
- 9 Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Equivalence of labeled Markov chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008. doi:10.1142/S0129054108005814.
- 10 Nathanaël Fijalkow. Undecidability results for probabilistic automata. *ACM SIGLOG News*, 4(4):10–17, 2017. doi:10.1145/3157831.3157833.
- 11 Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *Int. J. Softw. Tools Technol. Transf.*, 13(1):3–19, 2011.
- 12 Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The probabilistic model checker Storm, 2020. arXiv:arXiv:2002.07080.
- 13 James W. Gray III. Probabilistic interference. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 7-9, 1990*, pages 170–179. IEEE Computer Society, 1990. doi:10.1109/RISP.1990.63848.
- 14 Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 266–277. IEEE Computer Society, 1991.
- 15 John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Van Nostrand, 1960.
- 16 Stefan Kiefer and Qiyi Tang. Comparing labelled Markov decision processes. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 182 of *LIPICs*, pages 49:1–49:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.FSTTCS.2020.49.
- 17 Stefan Kiefer and Qiyi Tang. Strategies for MDP Bisimilarity Equivalence and Inequivalence. In Bartek Klin, Sławomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.CONCUR.2022.32.
- 18 Stefan Kiefer and Qiyi Tang. Minimising the probabilistic bisimilarity distance, 2024. arXiv:2406.19830.
- 19 Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

- 20 Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
- 21 Azaria Paz. *Introduction to probabilistic automata*. Academic Press, 2014.
- 22 Andrei Popescu, Johannes Hölzl, and Tobias Nipkow. Formalizing probabilistic noninterference. In Georges Gonthier and Michael Norrish, editors, *Certified Programs and Proofs - Third International Conference*, volume 8307 of *Lecture Notes in Computer Science*, pages 259–275. Springer, 2013. doi:10.1007/978-3-319-03545-1_17.
- 23 James Renegar. On the computational complexity and geometry of the first-order theory of the reals. Parts I–III. *Journal of Symbolic Computation*, 13(3):255–352, 1992.
- 24 Peter Y. A. Ryan, John D. McLean, Jonathan K. Millen, and Virgil D. Gligor. Non-interference: Who needs it? In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 237–238. IEEE Computer Society, 2001. doi:10.1109/CSFW.2001.930149.
- 25 Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200–214. IEEE Computer Society, 2000. doi:10.1109/CSFW.2000.856937.
- 26 Marcus Schaefer. Realizability of graphs and linkages. In *Thirty Essays on Geometric Graph Theory*, pages 461–482. Springer, 2012.
- 27 Marcus Schaefer and Daniel Stefankovic. Fixed points, Nash equilibria, and the existential theory of the reals. *Theory Comput. Syst.*, 60(2):172–193, 2017. doi:10.1007/s00224-015-9662-0.
- 28 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
- 29 Geoffrey Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003)*, pages 3–13. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212701.
- 30 Qiyi Tang. *Computing Probabilistic Bisimilarity Distances*. Phd thesis, York University, Toronto, September 2018. Available at <https://yorkspace.library.yorku.ca/items/7640b6ad-edb3-4e60-8f09-3db33c817061>.
- 31 Qiyi Tang and Franck van Breugel. Deciding probabilistic bisimilarity distance one for labelled Markov chains. In Hana Chockler and Georg Weissenbacher, editors, *Proceedings of the 30th International Conference on Computer Aided Verification*, volume 10981 of *Lecture Notes in Computer Science*, pages 681–699, Oxford, UK, July 2018. Springer-Verlag. doi:10.1007/978-3-319-96145-3_39.
- 32 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.
- 33 Antti Valmari and Giuliana Franceschinis. Simple $O(m \log n)$ time Markov chain lumping. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 38–52. Springer, 2010.
- 34 Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2001. doi:10.1007/3-540-48224-5_35.
- 35 Tobias Winkler, Sebastian Junges, Guillermo A. Pérez, and Joost-Pieter Katoen. On the complexity of reachability in parametric Markov decision processes. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.14.

Automating Memory Model Metatheory with Intersections

Aristotelis Koutsouridis 

MPI-SWS, Kaiserslautern and Saarbrücken, Germany

Michalis Kokologiannakis 

MPI-SWS, Kaiserslautern and Saarbrücken, Germany

Viktor Vafeiadis 

MPI-SWS, Kaiserslautern and Saarbrücken, Germany

Abstract

In the weak memory consistency literature, the semantics of concurrent programs is typically defined as a constraint on execution graphs, expressed in relational algebra. Prior work has shown that basic metatheoretic questions about memory models are decidable as long as they can be expressed as irreflexivity and emptiness constraints over Kleene Algebra with Tests (KAT), a condition that rules out practical memory models such the C/C++ and the Linux kernel models.

In this paper, we extend these results to memory models containing arbitrary intersections with uninterpreted relations. We can thus automatically establish compilation correctness and derive efficient incremental consistency checkers for RC11, LKMM, and other memory models.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Concurrency

Keywords and phrases Kleene Algebra, Weak Memory Models

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.33

Supplementary Material *Text (Full paper with technical appendix):* <https://plv.mpi-sws.org/kater>

Funding This work was supported by a European Research Council (ERC) Consolidator Grant for the project “PERSIST” under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 101003349).

Acknowledgements We would like to thank the anonymous reviewers for their feedback.

1 Introduction

In the weak memory consistency literature, the semantics of a concurrent and/or distributed program is typically defined as a set of labeled directed graphs, each representing a single possible execution of the program. These execution graphs comprise a set of nodes recording the individual memory accesses performed and a set of edges recording various ordering constraints among them. Example constraints [2] include the *program order* (po), the *reads-from* relation (rf), and the *coherence order* (co).

Each memory model defines a “consistency” constraint on execution graphs, asserting which graphs are possible outcomes of any program. These constraints are conveniently expressed in relational algebra with the help of some additional built-in sets (e.g., the set of read events R , and the set of write events W) and relations (e.g., `sameLoc` relating events accessing the same memory location, and `diffThread` relating events originating from different threads). For example, *sequential consistency* (SC) [18] can be defined as the constraint (SC) in Fig. 1; *coherence* (a.k.a., SC-per-location) as (COH), or equivalently as (COH2); *release-acquire* (RA) as (RA), or equivalently as (RA2), or equivalently as the conjunction of (COH) and (RA3); and *Total Store Order* (TSO) [22] as the conjunction of (COH) and (TSO).



© Aristotelis Koutsouridis, Michalis Kokologiannakis, and Viktor Vafeiadis; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 33; pp. 33:1–33:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\text{irreflexive}((\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})^+)$ where $\text{fr} \triangleq \text{rf}^{-1}; \text{co}$	(SC)
$\text{irreflexive}((\text{po} \cap \text{sameloc} \cup \text{rf} \cup \text{co} \cup \text{fr})^+)$	(COH)
$\text{irreflexive}(\text{po}^?; (\text{rf} \cup \text{co} \cup \text{fr})^+)$	(COH2)
$\text{irreflexive}(((\text{po} \cup \text{rf})^+ \cap \text{sameloc} \cup \text{co} \cup \text{fr})^+)$	(RA)
$\text{irreflexive}((\text{po} \cup \text{rf})^+; (\text{co} \cup \text{fr})^?)$	(RA2)
$\text{irreflexive}((\text{ppo} \cup \text{rf})^+; (\text{co} \cup \text{fr})^?)$ where $\text{ppo} \triangleq \text{po} \setminus (\text{W} \times \text{R})$	(RA3)
$\text{irreflexive}((\text{ppo} \cup \text{rf} \cap \text{diffthread} \cup \text{co} \cup \text{fr})^+)$	(TSO)

■ **Figure 1** Sample consistency constraints.

Kokologiannakis et al. [14] present KATER, a framework that can automatically answer certain fundamental questions about such definitions, but only for the case where the models are expressed purely as irreflexivity constraints over *Kleene Algebra with Tests* (KAT) [16]. This restriction to KAT, however, is a severe limitation of KATER: many common model definitions do not fall into this fragment (e.g., COH, RA, TSO), and although some of the simpler definitions can be equivalently expressed in KAT, more advanced practical models such as RC11 [17] and the Linux kernel memory model (LKMM) [1], cannot.

In response, we present KATI, an extension of KAT with *intersections* with *uninterpreted relations*, as well as a *top element*. KATI can express terms like $- \cap \text{sameloc}$ and $- \cap \text{diffthread}$ in Fig. 1, and supports all the aforementioned memory models. However, KATI also makes answering the following questions more difficult:

(Incremental) consistency checking: Is a given execution graph G consistent according to a model M ? Moreover, given an execution graph G and an event $e \in G$ such that $G \setminus \{e\}$ is M -consistent, is G also M -consistent?

Inclusion: Is memory model A *stronger* than a memory model B , i.e., does the consistency predicate of A imply that of B ?

Incremental consistency checking is important for testing and automated verification of concurrent programs (e.g., via stateless model checking [7, 15]). The problem admits a straightforward cubic solution (in the size of the execution graph) that calculates the relation appearing in the irreflexivity constraints in a bottom-up fashion. For acyclicity constraints of KAT expressions, KATER provides a better solution of linear complexity: it performs a custom DFS of the cross product of the execution graph with a finite state automaton corresponding to the KAT expression. We extend KATER’s linear-time solution to KATI with register automata [11], which extend standard finite state automata with a finite set of registers, which can store arbitrary values and compare them for equality.

Inclusion is not only an important metatheoretical question, but it actually also underlies the correctness proofs of compilation from one model to another and of local program transformations (compiler optimizations). Unfortunately, however, we cannot simply use our encoding into register automata because inclusion between register automata is generally undecidable [11]. We therefore follow another approach, and reduce relational intersection to KAT expressions over an extended alphabet with additional “bracket” letters. We prove that the resulting inclusion algorithm remains decidable (PSPACE-complete for a bounded number of intersections).

Our contributions can be summarized as follows:

- §2 We review KAT and show how it encodes consistency constraints of weak memory models.
- §3–§5 We present KATI, an extension of KAT that supports intersections with primitive relations, prove equivalence between its relational and language interpretation, and provide a decision procedure for language inclusion based on NFAs.

§6 We show how KATI can be used to check consistency of execution graphs in linear time. We conclude the paper with a presentation of related work (§7) and a note about future work (§8).

2 Kleene Algebra with Tests

In this section, we review the syntax and semantics of *Kleene Algebra with Tests* (KAT) [16].

2.1 Syntax and Interpretation

Syntax. KAT has two kinds of terms: tests and expressions.

Tests, $t \in \text{Test}$, form a boolean algebra over a set of primitive predicates, $p \in \text{P}$, i.e., they are constructed using the standard boolean/set operators: true (\top), false (\perp), union (\cup), intersection (\cap), and complement ($\bar{}$).

$$t ::= \top \mid \perp \mid p \mid t_1 \cup t_2 \mid t_1 \cap t_2 \mid \bar{t}$$

Expressions, $e \in \text{KAT}$, form a Kleene algebra over primitive relations, $r \in \text{R}$, and tests; i.e., they are constructed using relational composition (sequencing), union, and repetition.

$$e ::= r \mid [t] \mid e_1 ; e_2 \mid e_1 \cup e_2 \mid e^*$$

Unlike plain Kleene Algebra, KAT does not need special constructs for the empty string and the empty set, as these are given by the KAT expressions $[\top]$ and $[\perp]$ respectively.

Relational Interpretation. KAT terms can be interpreted in the context of a graph G , which defines the interpretations of primitive tests and relations. Formally, a graph G is a tuple $\langle \mathbf{E}_G, \mathcal{I}_G^{\text{P}}, \mathcal{I}_G^{\text{R}} \rangle$ where \mathbf{E}_G is a set of nodes (events) and \mathcal{I}_G^{P} is a function interpreting primitive tests over subsets of \mathbf{E}_G and \mathcal{I}_G^{R} primitive relations over binary relations on \mathbf{E}_G .

$$\mathcal{I}_G^{\text{P}} : \text{P} \rightarrow \mathcal{P}(\mathbf{E}_G) \quad \mathcal{I}_G^{\text{R}} : \text{R} \rightarrow \mathcal{P}(\mathbf{E}_G \times \mathbf{E}_G)$$

We extend these interpretations to arbitrary KAT terms as follows:

$$\begin{aligned} \llbracket p \rrbracket_G &\triangleq \mathcal{I}_G^{\text{P}}(p) & \llbracket r \rrbracket_G &\triangleq \mathcal{I}_G^{\text{R}}(r) \\ \llbracket \top \rrbracket_G &\triangleq \mathbf{E}_G & \llbracket [t] \rrbracket_G &\triangleq \{ \langle a, a \rangle \mid a \in \llbracket t \rrbracket_G \} \\ \llbracket \perp \rrbracket_G &\triangleq \emptyset & \llbracket [e_1 ; e_2] \rrbracket_G &\triangleq \{ \langle a, c \rangle \mid \exists b. \langle a, b \rangle \in \llbracket [e_1] \rrbracket_G \wedge \langle b, c \rangle \in \llbracket [e_2] \rrbracket_G \} \\ \llbracket [t] \rrbracket_G &\triangleq \mathbf{E}_G \setminus \llbracket [t] \rrbracket_G & \llbracket [e^*] \rrbracket_G &\triangleq (\llbracket [e] \rrbracket_G)^* \\ \llbracket [t_1 \cup t_2] \rrbracket_G &\triangleq \llbracket [t_1] \rrbracket_G \cup \llbracket [t_2] \rrbracket_G & \llbracket [e_1 \cup e_2] \rrbracket_G &\triangleq \llbracket [e_1] \rrbracket_G \cup \llbracket [e_2] \rrbracket_G \\ \llbracket [t_1 \cap t_2] \rrbracket_G &\triangleq \llbracket [t_1] \rrbracket_G \cap \llbracket [t_2] \rrbracket_G & & \end{aligned}$$

Language Interpretation. The main property of KAT is that inclusion and equivalence between KAT expressions is *decidable* (PSPACE-complete). This can be shown either with an algebraic axiomatization of KAT [16] or, as we show below, via an equivalent model of KAT expressions as a regular language.

Specifically, KAT expressions can be seen as regular languages over *guarded strings*, which we shall define below. To do so, we first define the *atoms* of a set of primitive tests.

► **Definition 1 (Atom).** An atom over $\text{P} = \{p_1, \dots, p_k\}$ is a string of literals $c_1 c_2 \dots c_k$ such that $c_i \in \{p_i, \bar{p}_i\}$, $1 \leq i \leq k$. Furthermore, the set of all 2^k atoms over P is denoted A_{P} .

33:4 Automating Memory Model Metatheory with Intersections

We use the greek lowercase letters α, β, \dots to denote atoms. For an atom α and a test t we write $\alpha \leq t$ to denote that $\alpha \rightarrow t$ is a propositional tautology.

► **Definition 2** (Guarded String). *A guarded string is a string over $\text{GS} \triangleq (\text{A}_P; \text{R})^*; \text{A}_P$, i.e., consists of a non-empty, alternating sequence of atoms and primitive relations, starting and ending with an atom.*

Concatenation and Kleene closure can be lifted to languages of guarded strings:

$$\begin{aligned} X \ ; Y &\triangleq \{u \cdot \alpha \cdot v \mid u \cdot \alpha \in X, \alpha \cdot v \in Y\} \\ X^{(0)} &\triangleq \text{A}_P \quad X^{(n+1)} \triangleq X \ ; X^{(n)} \quad X^{\otimes} \triangleq \bigcup_{n \geq 0} X^{(n)} \end{aligned}$$

Observe that concatenation is guarded, i.e., it is only defined if the two strings are composable.

The language interpretation, $\llbracket \cdot \rrbracket_{\text{L}}$, maps tests to sets of atoms and KAT expressions to (regular) sets of guarded strings.

$$\begin{aligned} \llbracket p \rrbracket_{\text{L}} &\triangleq \{\alpha \in \text{A}_P \mid \alpha \leq p\} & \llbracket r \rrbracket_{\text{L}} &\triangleq \{\alpha \cdot r \cdot \beta \mid \alpha, \beta \in \text{A}_P\} \\ \llbracket \top \rrbracket_{\text{L}} &\triangleq \text{A}_P & \llbracket [t] \rrbracket_{\text{L}} &\triangleq \llbracket t \rrbracket_{\text{L}} \\ \llbracket \perp \rrbracket_{\text{L}} &\triangleq \emptyset & \llbracket e_1 \cup e_2 \rrbracket_{\text{L}} &\triangleq \llbracket e_1 \rrbracket_{\text{L}} \cup \llbracket e_2 \rrbracket_{\text{L}} \\ \llbracket [t] \rrbracket_{\text{L}} &\triangleq \text{A}_P \setminus \llbracket t \rrbracket_{\text{L}} & \llbracket e_1 ; e_2 \rrbracket_{\text{L}} &\triangleq \llbracket e_1 \rrbracket_{\text{L}} \ ; \ \llbracket e_2 \rrbracket_{\text{L}} \\ \llbracket t_1 \cup t_2 \rrbracket_{\text{L}} &\triangleq \llbracket t_1 \rrbracket_{\text{L}} \cup \llbracket t_2 \rrbracket_{\text{L}} & \llbracket e^* \rrbracket_{\text{L}} &\triangleq (\llbracket e \rrbracket_{\text{L}})^{\otimes} \\ \llbracket t_1 \cap t_2 \rrbracket_{\text{L}} &\triangleq \llbracket t_1 \rrbracket_{\text{L}} \cap \llbracket t_2 \rrbracket_{\text{L}} & & \end{aligned}$$

2.2 Interpretation Equivalence

The language and relational interpretations of KAT expressions are equivalent in the sense that e_1 is included in e_2 according to the one interpretation if and only if it is included according to the other.

► **Theorem 3** (Interpretation Equivalence). $\llbracket e_1 \rrbracket_{\text{L}} \subseteq \llbracket e_2 \rrbracket_{\text{L}}$ if and only if $\forall G. \llbracket e_1 \rrbracket_G \subseteq \llbracket e_2 \rrbracket_G$.

Proof sketch. For the “ \Rightarrow ” direction, we define a function $\rho_G : \text{GS} \rightarrow \mathcal{P}(\text{E}_G \times \text{E}_G)$ that interprets guarded strings as relations on a graph G as follows:

$$\rho_G(\alpha) \triangleq \{\langle a, a \rangle \mid a \in \llbracket \alpha \rrbracket_G\} \quad \rho_G(\alpha \cdot r \cdot w) \triangleq \rho_G(\alpha) ; \llbracket r \rrbracket_G ; \rho_G(w)$$

Here, $\llbracket \alpha \rrbracket_G$ interprets the atom α as the composition of its primitive tests. We show that $\llbracket e \rrbracket_G = \bigcup_{w \in \llbracket e \rrbracket_{\text{L}}} \rho_G(w)$ (by induction on e). Then,

$$\llbracket e_1 \rrbracket_G = \bigcup_{w \in \llbracket e_1 \rrbracket_{\text{L}}} \rho_G(w) \subseteq \bigcup_{w \in \llbracket e_2 \rrbracket_{\text{L}}} \rho_G(w) = \llbracket e_2 \rrbracket_G.$$

For the “ \Leftarrow ” direction, from a word $w \in \llbracket e_1 \rrbracket_{\text{L}}$, we construct a “canonical” graph G_w as a sequence of nodes n_0, \dots, n_k , such that the only guarded string w' such that $\langle n_0, n_k \rangle \in \rho_{G_w}(w')$ is $w' = w$. Then it follows that $\langle n_0, n_k \rangle \in \llbracket e_1 \rrbracket_{G_w} \subseteq \llbracket e_2 \rrbracket_{G_w}$, and thus $w \in \llbracket e_2 \rrbracket_{\text{L}}$. ◀

Deciding Language Inclusion with NFAs. When deciding the inclusion $\llbracket e_1 \rrbracket_{\text{L}} \subseteq \llbracket e_2 \rrbracket_{\text{L}}$, it is convenient to use NFAs that accept guarded strings.

► **Definition 4.** *An NFA over an alphabet Σ is a tuple $\langle Q, \iota, F, \delta \rangle$, where Q is the set of states, $\iota \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation.*

Given an NFA, we abuse notation and write $\delta(S, a)$ for the set $\{q \in Q \mid \exists s \in S. \langle s, a, q \rangle \in \delta\}$. We also lift the transition relation to words as follows $\delta(S, \epsilon) \triangleq S$, and $\delta(S, aw) \triangleq \delta(\delta(S, a), w)$.

The *language* accepted by an NFA contains all words accepted by the NFA: $L(\langle Q, \iota, F, \delta \rangle) \triangleq \{w \in \Sigma \mid \delta(\{\iota\}, w) \cap F \neq \emptyset\}$.

Let us now define the function $\llbracket - \rrbracket_{\text{NFA}}$ to convert an expression $e \in \text{KAT}$ to an NFA over the alphabet of atoms and primitive relations: $\Sigma \triangleq \mathbf{A}_P \cup \mathbf{R}$.

$$\begin{aligned} \llbracket r \rrbracket_{\text{NFA}} &\triangleq \langle \{q_0, q_1, q_2, q_3\}, q_0, \{q_3\}, \{(q_1, r, q_2)\} \cup \bigcup_{\alpha \in \mathbf{A}_P} \{\langle q_0, \alpha, q_1 \rangle, \langle q_2, \alpha, q_3 \rangle\} \rangle \\ \llbracket [t] \rrbracket_{\text{NFA}} &\triangleq \langle \{q_0, q_1\}, q_0, \{q_1\}, \{\langle q_0, \alpha, q_1 \rangle \mid \alpha \in \llbracket t \rrbracket_L\} \rangle \\ \llbracket e_1; e_2 \rrbracket_{\text{NFA}} &\triangleq \langle Q_1 \uplus Q_2, \iota_1, F_2, \delta_1 \cup \delta_2 \cup \{\langle q_1, \alpha, q_2 \rangle \mid \delta_1(q_1, \alpha) \in F_1 \wedge (\iota_2, \alpha, q_2) \in \delta_2\} \rangle \\ &\quad \text{where } \llbracket e_i \rrbracket_{\text{NFA}} = \langle Q_i, \iota_i, F_i, \delta_i \rangle \text{ for } i \in \{1, 2\} \\ \llbracket e_1 \cup e_2 \rrbracket_{\text{NFA}} &\triangleq \langle Q_1 \uplus Q_2, \iota_1, F_1 \cup F_2, \delta_1 \cup \delta_2 \cup \{\langle \iota_1, \alpha, q_2 \rangle \mid \langle \iota_2, \alpha, q_2 \rangle \in \delta_2\} \rangle \\ &\quad \text{where } \llbracket e_i \rrbracket_{\text{NFA}} = \langle Q_i, \iota_i, F_i, \delta_i \rangle \text{ for } i \in \{1, 2\} \\ \llbracket e^* \rrbracket_{\text{NFA}} &\triangleq \langle Q \uplus \{q\}, \iota, F \cup \{q\}, \delta \cup \{\langle q_2, \alpha, q_1 \rangle \mid \langle \iota, \alpha, q_1 \rangle \in \delta, \langle q_2, \alpha, q_F \rangle \in \delta, q_F \in F\} \\ &\quad \cup \{\langle \iota, \alpha, q \rangle \mid \alpha \in \mathbf{A}_P\} \rangle \\ &\quad \text{where } \llbracket e \rrbracket_{\text{NFA}} = \langle Q, \iota, F, \delta \rangle \end{aligned}$$

By construction, the function $\llbracket - \rrbracket_{\text{NFA}}$ creates an NFA that accepts only guarded strings. In fact, $\llbracket e \rrbracket_{\text{NFA}}$ accepts precisely the words in $\llbracket e \rrbracket_L$.

► **Proposition 5** (NFA Equivalence). *For all $e \in \text{KAT}$, $\llbracket e \rrbracket_L = L(\llbracket e \rrbracket_{\text{NFA}})$.*

Language inclusion between KAT expressions can thus be checked via NFA automata and is PSPACE-complete.

2.3 Memory Models as KAT Constraints

Kokologiannakis et al. [14] observe that declarative *memory models* M can be formulated as a pair $\langle e_\emptyset, e_{\text{irr}} \rangle$ of an emptiness and an irreflexivity constraint over KAT. A memory model is interpreted as a set of execution graphs as follows $\llbracket \langle e_\emptyset, e_{\text{irr}} \rangle \rrbracket \triangleq \{G \mid \llbracket e_\emptyset \rrbracket_G \cup \llbracket e_{\text{irr}} \rrbracket_G \cap \text{id} = \emptyset\}$, where $\text{id} \triangleq \{\langle x, x \rangle \mid x \in \mathbf{E}_G\}$ is the identity relation.

Crucially, Kokologiannakis et al. [14] prove that various metatheoretic properties about memory models (such properties boil down to irreflexivity implications) can be decided in a sound and complete fashion:

► **Theorem 6** (KATER). *For every $e_1, e_2 \in \text{KAT}$, $\text{sameEnds}(\llbracket e_1 \rrbracket_L) \subseteq \text{DEDUP}(\text{ROT}(\llbracket e_2 \rrbracket_L))$ if and only if for all G , $\text{irreflexive}(\llbracket e_2 \rrbracket_G)$ implies $\text{irreflexive}(\llbracket e_1 \rrbracket_G)$.*

In the theorem above, $\text{sameEnds}(L) \triangleq \{\alpha \cdot v \cdot \alpha \mid \alpha \cdot v \cdot \alpha \in L\}$ restricts L so that its endpoints are compatible, $\text{ROT}(L) \triangleq \{\alpha \cdot u \cdot \beta \cdot v \cdot \alpha \mid \beta \cdot v \cdot \alpha \cdot u \cdot \beta \in L\}$ is the rotation closure of L , and $\text{DEDUP}(L) \triangleq \{\alpha \cdot w \cdot \alpha \mid \exists n. (\alpha \cdot w)^n \cdot \alpha \in L\}$ the deduplication closure.

Kokologiannakis et al. [14] further observe that the deduplication closure is never needed in practice, and so their tool, KATER, simply checks $\text{sameEnds}(\llbracket e_1 \rrbracket_L) \subseteq \text{ROT}(\llbracket e_2 \rrbracket_L)$.

3 KATI: Kleene Algebra with Tests and Intersections

In this section, we present our extension of KAT with relational intersection. KATI (Kleene Algebra with Tests and Intersections) extends KAT with relational intersection with *intersection relations*, $ir \in \text{IR}$, with the standard relational interpretation.

$$e \in \text{KATI} ::= \dots \mid e \cap ir \qquad \llbracket e \cap ir \rrbracket_G \triangleq \llbracket e \rrbracket_G \cap \llbracket ir \rrbracket_G$$

In this section, for simplicity, we assume that the set of primitive relations \mathbf{R} and the set of intersection relations \mathbf{IR} are disjoint. We will later lift this assumption in §5.

3.1 Language Interpretation

To show that inclusion between KATI expressions remains decidable, we need to suitably extend the language interpretation. To do so, we cannot employ the usual interpretation of intersection between formal languages because $\llbracket r \rrbracket_{\mathbf{L}} \cap \llbracket ir \rrbracket_{\mathbf{L}} = \{\alpha \cdot r \cdot \beta \mid \alpha, \beta \in \mathbf{A}_{\mathbf{P}}\} \cap \{\alpha \cdot ir \cdot \beta \mid \alpha, \beta \in \mathbf{A}_{\mathbf{P}}\} = \emptyset$.

Our idea is to introduce a set of bracket symbols $\mathbf{IR}_{()} \triangleq \bigcup_{ir \in \mathbf{IR}} \{(,_{ir},)_{ir}\}$ and interpret intersections as well-bracketed words over $\mathbf{IR}_{()} \cup \mathbf{R} \cup \mathbf{A}_{\mathbf{P}}$. Note, however, that we cannot simply interpret $e \cap ir$ as $\{(,_{ir} \cdot w \cdot)_{ir} \mid w \in \llbracket e \rrbracket_{\mathbf{L}}\}$, as such an interpretation fails to validate the following four important equivalences between KATI expressions that hold according to the relational interpretation.

$$\begin{aligned} (e \cap ir) \cap ir &= e \cap ir & (e \cap ir) \cap ir' &= (e \cap ir') \cap ir \\ ([t]; e) \cap ir &= [t]; (e \cap ir) & (e; [t]) \cap ir &= (e \cap ir); [t] \end{aligned}$$

Idempotence fails because the LHS has more brackets than the RHS, while the commutativity properties fail because the brackets (and the tests) appear in different orders. In addition, we sometimes want intersection relations, such as `sameLoc`, to be reflexive, in which case we would like to support the equivalence $[t] \cap ir = [t]$.

To resolve these problems, we assume a total order \prec on \mathbf{IR} and a function¹ $\text{id} : \mathbf{IR} \rightarrow \mathbf{Test}$ such that $\llbracket [\text{id}(ir)] \rrbracket_G = \llbracket [T] \cap ir \rrbracket_G$. Then, we can extend the notion of guarded strings to enforce a number of well-formed properties: (1) ignoring bracket symbols, words form a non-empty alternating sequence of atoms and primitive relations, starting and ending with an atom; (2) brackets are properly nested; (3) words inside brackets do not start or end with an atom, (4) directly nested brackets are sorted according to \prec . To do so, we introduce a set \mathbf{PGS} of non-empty words indexed by a set $S \subseteq \mathbf{IR}$ constraining any end-to-end bracket symbol to be indexed by an intersection relation in S ,

$$\begin{aligned} \mathbf{PGS}_S &\triangleq \{r \mid r \in \mathbf{R}\} \cup \{w_1 \cdot \alpha \cdot w_2 \mid w_1, w_2 \in \mathbf{PGS}_{\mathbf{IR}}, \alpha \in \mathbf{A}_{\mathbf{P}}\} \\ &\quad \cup \{(,_{ir} \cdot w \cdot)_{ir} \mid ir \in S, w \in \mathbf{PGS}_{\{>ir\}}\} \\ \mathbf{GS} &\triangleq \{\alpha \mid \alpha \in \mathbf{A}_{\mathbf{P}}\} \cup \{\alpha \cdot w \cdot \beta \mid \alpha, \beta \in \mathbf{A}_{\mathbf{P}}, w \in \mathbf{PGS}_{\mathbf{IR}}\} \end{aligned}$$

where $\{>ir\} \triangleq \{ir' \mid ir \prec ir'\}$.

Note that given $w \in \mathbf{PGS}_{\mathbf{IR}}$ and $ir \in \mathbf{IR}$, there exist $u \in (\{<ir\})^*$, $v \in (,_{ir}^?)$, $w' \in \mathbf{PGS}_{\{>ir\}}$ such that $w = u \cdot v \cdot w' \cdot \bar{v} \cdot \bar{u}$, where for a sequence of opening brackets u , we write \bar{u} for the corresponding sequence of closing brackets such that $u \cdot \bar{u}$ is well-nested.

We extend the language interpretation of KAT to KATI as follows:

$$\begin{aligned} \llbracket e \cap ir \rrbracket_{\mathbf{L}} &\triangleq \left\{ \alpha \cdot u \cdot (,_{ir} \cdot w \cdot)_{ir} \cdot \bar{u} \cdot \beta \mid \begin{array}{l} \alpha \cdot u \cdot v \cdot w \cdot \bar{v} \cdot \bar{u} \cdot \beta \in \llbracket e \rrbracket_{\mathbf{L}}, \\ u \in (\{<ir\})^*, v \in (,_{ir}^?), w \in \mathbf{PGS}_{\{>ir\}} \end{array} \right\} \\ &\quad \cup \{\alpha \mid \alpha \in \llbracket e \rrbracket_{\mathbf{L}}, \alpha \in \llbracket \text{id}(ir) \rrbracket_{\mathbf{L}}\} \end{aligned}$$

Using this definition, one can show that inclusion of the language interpretation implies inclusion of the relational interpretation.

¹ Such a function can always be defined by extending \mathbf{P} with additional primitive tests if necessary.

► **Proposition 7.** For all KATI expressions e_1, e_2 , if $\llbracket e_1 \rrbracket_L \subseteq \llbracket e_2 \rrbracket_L$, then $\forall G. \llbracket e_1 \rrbracket_G \subseteq \llbracket e_2 \rrbracket_G$.

Proof sketch. The conclusion follows by showing that $\llbracket e \rrbracket_G = \bigcup_{w \in \llbracket e \rrbracket_L} \rho_G(w)$, where the function $\rho_G : (\text{GS} \cup \text{PGS}_{\text{IR}}) \rightarrow \mathcal{P}(\mathbf{E}_G \times \mathbf{E}_G)$ is defined recursively as follows:

$$\begin{aligned} \rho_G(\alpha) &\triangleq \llbracket \llbracket \alpha \rrbracket \rrbracket_G & \rho_G(\alpha \cdot u \cdot \beta) &\triangleq \llbracket \llbracket \alpha \rrbracket \rrbracket_G ; \rho_G(u) ; \llbracket \llbracket \beta \rrbracket \rrbracket_G \\ \rho_G(r) &\triangleq \llbracket r \rrbracket_G & \rho_G(u \cdot \alpha \cdot v) &\triangleq \rho_G(u) ; \llbracket \llbracket \alpha \rrbracket \rrbracket_G ; \rho_G(v) \\ & & \rho_G(({}_{ir} \cdot u \cdot)_{ir}) &\triangleq \rho_G(u) \cap \llbracket ir \rrbracket_G \end{aligned} \quad \blacktriangleleft$$

The other direction, however, does not hold because $r \cap ir \subseteq r$ clearly holds according to the relational interpretation, but not according to the language interpretation. More generally, the issue is that RHS can have fewer intersections than the LHS and so its language interpretation can have fewer brackets than that of the LHS.

We therefore define a partial order \lesssim_B on guarded strings $(\text{GS} \cup \text{PGS}_{\text{IR}})$ that allows the LHS to contain more brackets than the RHS as the least structure-preserving partial order relating $({}_{ir} \cdot w \cdot)_{ir} \lesssim_B w$ for all $w \in \text{PGS}_{\text{IR}}$, where we call an order \lesssim_B *structure-preserving* if:

$$\frac{u \lesssim_B w}{\alpha \cdot u \cdot \beta \lesssim_B \alpha \cdot w \cdot \beta} \quad \frac{u_1 \lesssim_B w_1 \quad u_2 \lesssim_B w_2}{u_1 \cdot \alpha \cdot u_2 \lesssim_B w_1 \cdot \alpha \cdot w_2} \quad \frac{u \lesssim_B w}{({}_{ir} \cdot u \cdot)_{ir} \lesssim_B ({}_{ir} \cdot w \cdot)_{ir}}$$

We can easily define the bracketed saturation of a language $\text{BR}(L) \triangleq \{u \mid w \in L \wedge u \lesssim_B w\}$, and write $L_1 \lesssim_B L_2$ when $L_1 \subseteq \text{BR}(L_2)$, i.e., when for all $u \in L_1$, there exists $w \in L_2$ such that $u \lesssim_B w$.

With the above building blocks in place, we can prove the following equivalence between the two KATI representations.

► **Theorem 8 (Interpretation Equivalence).** $\llbracket e_1 \rrbracket_L \lesssim_B \llbracket e_2 \rrbracket_L$ if and only if $\forall G. \llbracket e_1 \rrbracket_G \subseteq \llbracket e_2 \rrbracket_G$.

Proof sketch. The “ \Rightarrow ” direction follows from Prop. 7 and the observation that $u \lesssim_B v$ implies $\rho_G(u) \subseteq \rho_G(v)$.

In the “ \Leftarrow ” direction, from a guarded string $w \in \llbracket e_1 \rrbracket_L$, we construct a “canonical” graph G_w as a sequence of nodes n_0, \dots, n_k , such that a guarded string u has $\langle n_0, n_k \rangle \in \rho_{G_w}(u)$ iff $w \lesssim_B u$. Then, it follows that $\langle n_0, n_k \rangle \in \llbracket e_1 \rrbracket_{G_w} \subseteq \llbracket e_2 \rrbracket_{G_w}$, and thus $w \in \text{BR}(\llbracket e_2 \rrbracket_L)$. \blacktriangleleft

Theorem 8 provides a way to use language-based techniques to reason about inclusion of KATI expressions. There are two remaining questions:

- How can we finitely represent $\llbracket e \rrbracket_L$?
- How can we finitely represent the bracketing closure, $\text{BR}(L)$?

We first tackle the former question in §3.2, and relegate the second to §3.3.

Before we do so, we present an improvement of the bracketing closure that does not blindly add further brackets, but only ones that appear in the LHS of the inclusion. We say that the *nesting context* at given index of a guarded string is the sequence of relations corresponding to unmatched open brackets up to that index. We will be mainly interested in the set of all nesting contexts of a string, $c(w)$, which can be defined inductively as follows:

$$\begin{aligned} c(\alpha) &\triangleq c(r) \triangleq \{\epsilon\} & c(({}_{ir} \cdot w \cdot)_{ir}) &\triangleq \{\epsilon\} \cup \{ir \cdot u \mid u \in c(w)\} \\ c(w_1 \cdot \alpha \cdot w_2) &\triangleq c(w_1) \cup c(w_2) & c(\alpha \cdot w \cdot \beta) &\triangleq c(w) \end{aligned}$$

Given a set of nesting contexts C and a language of guarded strings L , its restricted bracketing closure is $\text{BR}_C(L) \triangleq \{u \mid w \in L \wedge u \lesssim_B w \wedge c(u) \subseteq C\}$. Using the restricted bracketing closure suffices to show inclusion.

► **Proposition 9.** $L_1 \lesssim_B L_2$ if and only if $L_1 \subseteq \text{BR}_{c(L_1)}(L_2)$.

3.2 Converting KATI Expressions to Automata

As in §2, we will again use NFAs to compute $\llbracket \cdot \rrbracket_{\mathbb{L}}$ albeit with a much more complex construction. As it is difficult to provide a direct NFA construction corresponding to $\llbracket e \cap ir \rrbracket_{\mathbb{L}}$, we will first put e in a normal form that enables a straightforward construction.

Normalization. The idea of the normal form is to ensure that (1) there are no tests immediately inside a bracket, and (2) directly nested brackets appear in \prec -order. To arrive at such a form, we first convert an expression e into a form that makes all possible tests at the beginning and the end of a string explicit. For this, we define $\text{pred}(e)$, which returns a test t such that $[t] = e \cap [\top]$, and $\text{pull}(e)$, which makes explicit any tests at the beginning of e .

$$\begin{array}{ll}
\text{pred}([t]) \triangleq t & \text{pull}([t]) \triangleq \emptyset \\
\text{pred}(r) \triangleq \perp & \text{pull}(r) \triangleq r \\
\text{pred}(e \cap ir) \triangleq \perp & \text{pull}(e \cap ir) \triangleq e \cap ir \\
\text{pred}(e_1 \cup e_2) \triangleq \text{pred}(e_1) \cup \text{pred}(e_2) & \text{pull}(e_1 \cup e_2) \triangleq \text{pull}(e_1) \cup \text{pull}(e_2) \\
\text{pred}(e_1 ; e_2) \triangleq \text{pred}(e_1) \cap \text{pred}(e_2) & \text{pull}(e_1 ; e_2) \triangleq [\text{pred}(e_1)] ; \text{pull}(e_2) \cup \text{pull}(e_1) ; e_2 \\
\text{pred}(e^*) \triangleq \top & \text{pull}(e^*) \triangleq \text{pull}(e) ; e^*
\end{array}$$

► **Definition 10.** The converse of an expression $e \in \text{KATI}$, written e^{-1} , is defined as follows:

$$\begin{array}{lll}
[t]^{-1} \triangleq [t] & (e_1 \cup e_2)^{-1} \triangleq e_1^{-1} \cup e_2^{-1} & (e_1 ; e_2)^{-1} \triangleq e_2^{-1} ; e_1^{-1} \\
(e^*)^{-1} \triangleq (e^{-1})^* & (e \cap ir)^{-1} \triangleq e^{-1} \cap ir^{-1} & (x^{-1})^{-1} \triangleq x \text{ for } x \in \text{R} \cup \text{IR}.
\end{array}$$

► **Lemma 11.** $\llbracket e \rrbracket_{\mathbb{L}} = \llbracket [\text{pred}(e)] \cup \text{pull}(e) \rrbracket_{\mathbb{L}} = \llbracket [\text{pred}(e)] \cup \text{pull}((\text{pull}(e^{-1}))^{-1}) \rrbracket_{\mathbb{L}}$.

To convert an expression into normal form, we apply the following rewrite rules in a bottom-up fashion. The first rule is applied only once for each intersection in the KATI expression; the remaining rules as much as possible.

$$\begin{array}{l}
e \cap ir = [\text{pred}(e) \cap \text{id}(ir)] \cup \text{pull}((\text{pull}(e^{-1}))^{-1}) \cap ir \\
([t] ; e) \cap ir = [t] ; (e \cap ir) \\
(e ; [t]) \cap ir = (e \cap ir) ; [t] \\
(e_1 \cup e_2) \cap ir = e_1 \cap ir \cup e_2 \cap ir \\
((e_1 \cup e_2) ; e) \cap ir = (e_1 ; e) \cap ir \cup (e_2 ; e) \cap ir \\
(e ; (e_1 \cup e_2)) \cap ir = (e ; e_1) \cap ir \cup (e ; e_2) \cap ir \\
(e \cap ir) \cap ir = e \cap ir \\
(e \cap ir') \cap ir = (e \cap ir) \cap ir' \text{ if } ir' \prec ir
\end{array}$$

It is easy to show that all these rules are equivalences according to the language interpretation, and thus $\llbracket \text{normalize}(e) \rrbracket_{\mathbb{L}} = \llbracket e \rrbracket_{\mathbb{L}}$. We observe that the size of the normalized expression increases exponentially with the nesting depth of the expression. However, if we assume a bounded nesting of intersections in KATI expressions (as in all memory models), then our decision procedure for inclusion remains PSPACE-complete.

NFA Conversion. Once e is in normal form, conversion to an NFA is fairly straightforward. The only new case is that of the intersection of an automaton with ir , which adds $(_{ir}$ and $)_{ir}$ transitions at the start and end of the automaton, and ensures that any α -transition from the initial to a final state satisfies $\alpha \in \llbracket \text{id}(ir) \rrbracket_{\mathbb{L}}$. Assuming that $\llbracket e \rrbracket_{\text{NFA}} = \langle Q, \iota, F, \delta \rangle$, $\llbracket e \cap ir \rrbracket_{\text{NFA}}$ returns the NFA $\langle Q', \iota, F, \delta' \rangle$, where:

$$\begin{aligned}
Q' &= Q \uplus \{q_{open} \mid \langle \iota, _, q \rangle \in \delta\} \uplus \{q_{close} \mid \langle q, _, q_F \rangle \in \delta, q_F \in F\} \\
\delta' &= \{ \langle q, \sigma, q' \rangle \in \delta \mid q \neq \iota, q' \notin F \} \\
&\cup \{ \langle \iota, \alpha, q_{open} \rangle, \langle q_{open}, (ir, q) \mid \langle \iota, \alpha, q \rangle \in \delta \} \\
&\cup \{ \langle q, \rangle_{ir}, q_{close} \rangle, \langle q_{close}, \alpha, q_F \rangle \mid \langle q, \alpha, q_F \rangle \in \delta, q_F \in F \} \\
&\cup \{ \langle \iota, \alpha, q_F \rangle \mid \langle \iota, \alpha, q_F \rangle \in \delta, q_F \in F, \alpha \in \llbracket \text{id}(ir) \rrbracket_L \}
\end{aligned}$$

The correctness of the conversion is captured by the following proposition.

► **Proposition 12.** *For all KATI expressions e , $L(\llbracket \text{normalize}(e) \rrbracket_{\text{NFA}}) = \llbracket e \rrbracket_L$.*

3.3 Saturating NFAs with Brackets

We move on to define the bracketing saturation of an NFA. We begin by making an observation about the structure of the automaton $\llbracket e \rrbracket_{\text{NFA}}$ corresponding to a KATI expression e . Observe that every state q in $\llbracket e \rrbracket_{\text{NFA}}$ has a unique nesting context: all runs from the initial state(s) to q go through the same sequence of unmatched brackets. As such, we first define the function $c(\cdot) : Q \rightarrow \text{IR}^*$ returning the nesting context of each state.

► **Definition 13 (Nesting context).** *Given an NFA $\langle Q, \iota, F, \delta \rangle$ and a state $q \in Q$, the nesting context of q , written $c(q)$, is the word $ir_1 \cdots ir_k$ corresponding to the unmatched open bracket symbols $(ir_1 \cdots (ir_k$ along any run from an initial state ι to q .*

Then, we define the notion of *nesting context completion* (or nesting completion for short). Intuitively, a nesting completion is used to saturate a KATI expression with matching brackets. In practice, we want to saturate the right-hand side of an inclusion with brackets that exist in the left-hand side, and as such we define the nesting completion of a context d w.r.t. a set of nesting contexts C .

► **Definition 14 (Nesting completion).** *Given a nesting context $d = ir_1 \cdots ir_k$ and a set of nesting contexts C , the sequence $N = [w_1, \dots, w_{k+1}]$ of $k+1$ words $w_i \in \text{IR}^*$ is called a nesting completion of d with respect C , written $d \rightsquigarrow^N C$, if $w_1 \cdot ir_1 \cdots ir_k \cdot w_{k+1} \in C$.*

Given a sequence N of words $w_i \in \text{IR}^*$, we write:

- $N.\epsilon$ for the sequence that appends the empty string at the end of N : $[w_1, w_2, \dots, w_{k+1}, \epsilon]$.
- N/ir for the sequence that appends $ir \in \text{IR}$ at the last word of N : $[w_1, \dots, w_k, (w_{k+1} \cdot ir)]$.

At this point we are ready to define our bracketed substring saturation on NFAs. Using nesting completions, we can construct the saturated automaton. Given an NFA $\langle Q, \iota, F, \delta \rangle$ we define its *bracketed saturation* w.r.t. a set of nesting contexts C , written $\text{BR}_C(\langle Q, \iota, F, \delta \rangle)$, as the automaton $\langle Q_{sat}, \iota_{sat}, F_{sat}, \delta_{sat} \rangle$, where:

$$\begin{aligned}
Q_{sat} &\triangleq \{(q, N) \mid q \in Q, c(q) \rightsquigarrow^N C\} \\
\iota_{sat} &\triangleq (\iota, [\epsilon]) \\
F_{sat} &\triangleq \{(q, [\epsilon]) \mid q \in F\} \\
\delta_{sat} &\triangleq \{ ((q, N), a, (q', N)) \mid (q, a, q') \in \delta, (q, N), (q', N) \in Q_{sat} \} \\
&\cup \{ ((q, N), (ir, (q', N.\epsilon)) \mid (q, (ir, q') \in \delta, (q, N), (q', N.\epsilon) \in Q_{sat} \} \\
&\cup \{ ((q, N.\epsilon),)_{ir}, (q', N) \mid (q,)_{ir}, q' \in \delta, (q, N), (q', N.\epsilon) \in Q_{sat} \} \\
&\cup \{ ((q, N), (ir, (q, N/ir)) \mid (q, N/ir), (q, N) \in Q_{sat} \} \\
&\cup \{ ((q, N/ir),)_{ir}, (q, N) \mid (q, N/ir), (q, N) \in Q_{sat} \}
\end{aligned}$$

33:10 Automating Memory Model Metatheory with Intersections

As can be seen, the saturated NFA has the initial and final states of the original NFA with the empty completion as its initial states and final states, while its transition relation has three kinds of edges: (a) those maintaining the same nesting completion (modulo adding or removing an empty word at the end), when the original NFA performs the corresponding transition, (b) those incrementing the last word of the current nesting completion by reading an open bracket, and (c) those decrementing the last word of the current nesting completion by closing a bracket.

Correctness. We next prove that bracketing saturation at the level of NFAs is a sound and complete method for proving inclusion between KATI expressions, and thus inclusion is decidable.

► **Proposition 15** (Bracketing Saturation Correctness). *Let A be an automaton accepting only guarded strings and C be a set of nesting contexts. Then, $\text{BR}_C(\text{L}(A)) = \text{L}(\text{BR}_C(A)) \cap \text{GS}$.*

Proof sketch. In the “ \supseteq ” direction, let $w \in \text{L}(\text{BR}_C(A)) \cap \text{GS}$, and $(s, [\epsilon]) \xrightarrow{w} (t, [\epsilon])$ the respective accepting run on $\text{BR}_C(A)$. By induction on the structure of w , we show that there exists a corresponding run $s \xrightarrow{u} t$ in A such that $w \lesssim_{\text{B}} u$. This run is accepting on A , since s and t are an initial and final state of A , respectively, so $w \in \text{BR}_C(\text{L}(A))$.

In the “ \subseteq ” direction, let $s \xrightarrow{u} t$ be an accepting path in A and $w \lesssim_{\text{B}} u$ with $c(w) \subseteq C$. By induction on the structure of \lesssim_{B} , we show that there exists a corresponding path $(s, [\epsilon]) \xrightarrow{w} (t, [\epsilon])$ in $\text{BR}_C(A)$. Since $(s, [\epsilon]) \xrightarrow{w} (t, [\epsilon])$ are initial/final in $\text{BR}_C(A)$ by construction, we obtain the desired result. ◀

Putting Propositions 9, 12, and 15 together, we can derive the soundness and completeness of the NFA-based checking of inclusion.

► **Theorem 16** (Decidability of Inclusion). *For all $e_1, e_2 \in \text{KATI}$, $\llbracket e_1 \rrbracket_{\text{L}} \lesssim_{\text{B}} \llbracket e_2 \rrbracket_{\text{L}}$ if and only if $\text{L}(\llbracket \text{normalize}(e_1) \rrbracket_{\text{NFA}}) \subseteq \text{L}(\text{BR}_{c(e_1)}(\llbracket \text{normalize}(e_2) \rrbracket_{\text{NFA}}))$.*

Proof. We show that the LHS is equivalent to the RHS:

$$\begin{aligned}
 \text{L}(\llbracket \text{normalize}(e_1) \rrbracket_{\text{NFA}}) &= \llbracket e_1 \rrbracket_{\text{L}} && \text{by Prop. 12} \\
 &\subseteq \text{BR}_{c(e_1)}(\llbracket e_2 \rrbracket_{\text{L}}) && \text{by Prop. 9 and the LHS} \\
 &= \text{BR}_{c(e_1)}(\text{L}(\llbracket \text{normalize}(e_2) \rrbracket_{\text{NFA}})) && \text{by Prop. 12} \\
 &= \text{L}(\text{BR}_{c(e_1)}(\llbracket \text{normalize}(e_2) \rrbracket_{\text{NFA}})) && \text{by Prop. 15} \quad \blacktriangleleft
 \end{aligned}$$

4 Memory Models as KATI Constraints

Let us now revisit §2, and see how irreflexivity implications between model definitions in KATI can be proved in a sound fashion. Recall from Theorem 6 that KATER reduces irreflexivity implications to a language inclusion problem, after taking some closures on the involved expressions. We would of course like to follow the same strategy in KATI, but unfortunately the deduplication closure $\text{DEDUP}(L)$ cannot be easily adjusted to bracketed strings.

Nonetheless, we can adjust the rotation closure $\text{ROT}(L)$ which raises a problem when applied to bracketed strings. Indeed, assuming the previous definition of $\text{ROT}(L)$, if the language L contains the string $\alpha \cdot u_1 \cdot \beta \cdot ({}_{ir} \cdot w_1 \cdot \gamma \cdot w_2)_{ir} \cdot \alpha$, $\text{ROT}(L)$ will include strings that are not well-bracketed like $\gamma \cdot w_2 \cdot ({}_{ir} \cdot \alpha \cdot u_1 \cdot \beta \cdot ({}_{ir} \cdot w_1 \cdot \gamma$.

To retain well-bracketedness, we have to redefine $\text{ROT}(L)$. To that end, we first define a helper function $\text{split}()$ that splits a string into a prefix and a suffix, and inverts the unmatched brackets of each substring.

$$\begin{aligned} \text{split}(r) &\triangleq \emptyset \\ \text{split}((_{ir} \cdot w \cdot)_{ir}) &\triangleq \{ \langle \rangle_{ir^{-1}} \cdot u, \beta, v \cdot (_{ir^{-1}}) \mid \langle u, \beta, v \rangle \in \text{split}(w) \} \\ \text{split}(w_1 \cdot \alpha \cdot w_2) &\triangleq \{ \langle u, \beta, v' \cdot r \cdot \alpha \cdot ({}_S \cdot w_2) \mid \langle u, \beta, v' \cdot r \cdot ({}_S) \rangle \in \text{split}(w_1) \} \\ &\quad \cup \{ \langle w_1 \cdot ({}_S) \cdot \alpha \cdot r \cdot u', \beta, v \rangle \mid \langle \rangle_{{}_S} \cdot r \cdot u', \beta, v \rangle \in \text{split}(w_2) \} \\ &\quad \cup \{ \langle w_1, \alpha, w_2 \rangle \} \end{aligned}$$

Inverting a bracket, inverts the corresponding intersection relation; if the relation is symmetric, then $ir^{-1} = ir$. In the definition above, $({}_S \cdot)_S$ denotes a sequence of zero or more opening and closing brackets respectively and S is the sequence of intersection relations ir that appear in the bracket subscripts. We can easily verify that if $\langle u, \alpha, v \rangle \in \text{split}(w)$, then $u, v \in ((R \cup IR_{()}) \cdot \text{Ap})^* \cdot (R \cup IR_{()})$, i.e., they are in guarded form.

Given $\text{split}()$, we define $\text{ROT}(L)$ as follows:

$$\begin{aligned} \text{ROT}(\alpha) &\triangleq \{ \alpha \} \\ \text{ROT}(\alpha \cdot w \cdot \alpha) &\triangleq \left\{ \beta \cdot v \cdot r' \cdot \alpha \cdot r \cdot u \cdot \beta \mid \begin{array}{l} \langle \rangle_{{}_S} \cdot r \cdot u, \beta, v \cdot r' \cdot ({}_S) \rangle \in \text{split}(w) \\ \forall ir \in S. \alpha \leq \text{id}(ir) \end{array} \right\} \cup \{ \alpha \cdot w \cdot \alpha \} \\ \text{ROT}(L) &\triangleq \{ u \in \text{ROT}(w) \mid w \in L \} \end{aligned}$$

Observe that rotation produces only guarded strings because it commutes tests outside of brackets and $\text{split}()$ inverts the direction of brackets.

We obtain the following equivalences.

► **Proposition 17 (Irreflexivity Equivalence).** *Given a graph G and a language $L \subseteq \text{GS}$:*

$$\begin{aligned} \text{irreflexive}(\rho_G(L)) &\Leftrightarrow \text{irreflexive}(\rho_G(\text{sameEnds}(L))) \Leftrightarrow \text{irreflexive}(\rho_G(\text{BR}(L))) \\ &\Leftrightarrow \text{irreflexive}(\rho_G(\text{ROT}(L))), \end{aligned}$$

where $\rho_G(L) \triangleq \bigcup_{w \in L} \rho_G(w)$ and $\rho_G(w)$ is defined in the proof sketch of Prop. 7.

Proof sketch. The first equivalence can be shown in a similar fashion to that in [14]. The second equivalence follows directly from the observation that $w \lesssim_{\text{B}} u$ implies $\rho_G(w) \subseteq \rho_G(u)$. For the final one, the “ \Leftarrow ” direction is trivial because $L \subseteq \text{ROT}(L)$.

To prove that $\text{irreflexive}(\rho_G(L)) \Rightarrow \text{irreflexive}(\rho_G(\text{ROT}(L)))$, consider $\langle b, b \rangle \in \rho_G(w)$ for some $w \in \text{ROT}(L) \setminus L$. (If $w \in L$, the conclusion holds trivially.) Expanding the definition of rotation, $w = \beta \cdot v \cdot \alpha \cdot u \cdot \beta$ with $\langle u, \alpha, v \rangle \in \text{split}(w)$, where u, v are the result of inverting the unmatched brackets of u', v' respectively, and $w' = \alpha \cdot u' \cdot \beta \cdot v' \cdot \alpha \in L$. Here, b is the node of G that corresponds to the atom β , and let a be the node that corresponds to the atom α in the cycle $\langle b, b \rangle$. Let γ_1, γ_2 be the atom adjacent to a possible unmatched bracket (originating from a matching pair of brackets $(_{ir},)_{ir}$) in v and u respectively and g_1, g_2 the corresponding nodes of G for these atoms in the cycle $\langle b, b \rangle$. Also, since $\langle b, b \rangle \in \rho_G(w)$, we know that $\langle g_1, g_2 \rangle \in \llbracket ir \rrbracket_G$. When calculating $\rho_G(w')$ we would interpret this pair of brackets with an intersection of the tuple $\{ \langle g_2, g_1 \rangle \}$ with $\llbracket ir^{-1} \rrbracket_G$, which includes $\{ \langle g_2, g_1 \rangle \}$. Therefore, $\langle a, a \rangle \in \rho_G(w')$ contradicting that $\rho_G(L)$ is irreflexive. ◀

► **Theorem 18 (Irreflexivity Implications).** *For every $e_1, e_2 \in \text{KATI}$, if $\text{sameEnds}(\llbracket e_1 \rrbracket_L) \subseteq \text{ROT}(\text{BR}(\llbracket e_2 \rrbracket_L))$ then for all G , $\text{irreflexive}(\llbracket e_2 \rrbracket_G) \Rightarrow \text{irreflexive}(\llbracket e_1 \rrbracket_G)$.*

Proof sketch. Follows by repeated application of Prop. 17. ◀

5 KATI: Adding a “Top” Element

In this section, we extend KATI so that any relation $r \in \mathbf{R}$ can be used in intersections (and not only some dedicated relations).

The problem when doing so is that KATI’s language interpretation is inadequate when it comes to prove certain relational properties. For instance, even though $\llbracket r_1 \cap r_2 \rrbracket_G = \llbracket r_2 \cap r_1 \rrbracket_G$, our bracketed language interpretation will yield $\llbracket r_1 \cap r_2 \rrbracket_L = (r_2 \cdot r_1 \cdot)_{r_2}$ which in turn is not equal to $(r_1 \cdot r_2 \cdot)_{r_1} = \llbracket r_2 \cap r_1 \rrbracket_L$. Of course, this particular case could be handled as part of our normalization procedure, but more complicated relational inclusions (e.g., $\llbracket (r_1; r_2) \cap r_3 \rrbracket_G \subseteq \llbracket r_3 \rrbracket_G$) cannot be handled with said normalization.

To remedy this, we introduce a *top relation*, \mathbf{top} , and express all primitive relations as intersections with \mathbf{top} as follows:

$$\begin{aligned} \llbracket \mathbf{top} \rrbracket_L &\triangleq \{ \alpha \cdot \mathbf{top} \cdot \beta \mid \alpha, \beta \in \mathbf{A}_P \} \\ \llbracket r \rrbracket_L &\triangleq \llbracket \mathbf{top} \cap r \rrbracket_L = \{ \alpha \cdot (r \cdot \mathbf{top} \cdot)_r \cdot \beta \mid \alpha, \beta \in \mathbf{A}_P \} \cup \{ \alpha \mid \alpha \in \llbracket \mathbf{id}(r) \rrbracket_L \} \end{aligned}$$

Observe that using the definition above and assuming that \prec totally orders $\mathbf{R}_()$, we can already easily prove inclusions like $\llbracket r_1 \cap r_2 \rrbracket_L = \llbracket r_2 \cap r_1 \rrbracket_L$, since KATI’s language interpretation of intersections already imposes a total order on brackets: the language interpretation of both expressions is $(r_1 \cdot (r_2 \cdot \mathbf{top} \cdot)_{r_2} \cdot)_{r_1}$.

To be able to prove inclusions like $(r_1; r_2) \cap r_3 \subseteq r_3$, we introduce the top-closure $\lesssim_{\mathbf{T}}$ as the least structure-preserving partial order on $\mathbf{GS} \cup \mathbf{PGS}_{\mathbf{IR}}$ containing $w \lesssim_{\mathbf{T}} \mathbf{top}$ for all $w \in \mathbf{PGS}_{\emptyset}$, and define $\lesssim_{\mathbf{BT}} \triangleq (\lesssim_{\mathbf{B}} \cup \lesssim_{\mathbf{T}})^+$, which is in fact equivalent to $\lesssim_{\mathbf{B}} ; \lesssim_{\mathbf{T}}$. The top closure of a language $L \subseteq \mathbf{GS}$ is $\mathbf{T}(L) \triangleq \{ u_1 \cdot w \cdot u_2 \mid u_1 \cdot \mathbf{top} \cdot u_2 \in L, w \in \mathbf{PGS}_{\emptyset} \}$.

With the above definition for $\lesssim_{\mathbf{T}}$ we can prove equivalence between the language and the relational interpretation of KATI (Theorem 8).

As far as the decision procedure of §3.2 and §3.3 is concerned, we can extend it to handle the new top element by modifying the NFA conversion of expressions consisting of a single primitive relation r , and our bracketed saturation. For the former, we redefine $\llbracket r \rrbracket_{\mathbf{NFA}}$ as the automaton $\langle \{q_0, q_1, q_2, q_3, q_4, q_5\}, q_0, \{q_5\}, \delta_{\mathbf{top} \cap r} \rangle$ where

$$\delta_{\mathbf{top} \cap r} \triangleq \bigcup_{\alpha \in \mathbf{A}_P} \{ \langle q_0, \alpha, q_1 \rangle \} \cup \{ \langle q_1, (r, q_2), \langle q_2, \mathbf{top}, q_3 \rangle, \langle q_3,)_r, q_4 \rangle \} \cup \bigcup_{\alpha \in \mathbf{A}_P} \{ \langle q_4, \alpha, q_5 \rangle \}.$$

For the latter, given an NFA $A = \langle Q, \iota, F, \delta \rangle$, we define its top-closure $\mathbf{T}(A)$ as the automaton $\langle Q, \iota, F, \delta \cup \delta_{\mathbf{top}} \rangle$ where $\delta_{\mathbf{top}} = \{ \langle q', \alpha, q \rangle \mid \langle q, \mathbf{top}, q' \rangle \in \delta, \alpha \in \mathbf{A}_P \}$

► **Proposition 19** (Top Closure Correctness). *For every automaton A accepting only guarded strings, $\mathbf{T}(L(A)) = L(\mathbf{T}(A))$.*

Then, we take the combined bracketing-top closure as $\mathbf{BR}_C^{\mathbf{top}}(A) \triangleq \mathbf{BR}_C(\mathbf{T}(A))$, and we obtain as corollary of Theorem 16 and Prop. 19 our main decidability result.

► **Theorem 20.** $\llbracket e_1 \rrbracket_L \lesssim_{\mathbf{BT}} \llbracket e_2 \rrbracket_L$ iff $L(\llbracket \mathbf{normalize}(e_1) \rrbracket_{\mathbf{NFA}}) \subseteq L(\mathbf{BR}_{c(e_1)}^{\mathbf{top}}(\llbracket \mathbf{normalize}(e_2) \rrbracket_{\mathbf{NFA}}))$.

6 Consistency Checking

Similarly to KATER, KATI can also be used to generate consistency-checking code for a memory model’s acyclicity constraints. In this section, we briefly recall KATER’s code-generating infrastructure, and then show this infrastructure can be extended for the KATI language.

6.1 Consistency Checking with Kater

The key idea behind KATER’s consistency-checking infrastructure is twofold. First, given a constraint demanding that a KAT expression e be acyclic, any e -cycle in a given graph G will ultimately be composed of primitive relations and predicates $r \in \mathbf{R}$ and $\pi \in \mathbf{P}$, i.e., the same primitives used to express e in KAT. As such, to find e -cycles in G , one only has to find some cyclic path in G , a permutation of which is accepted by $\llbracket e \rrbracket_{\text{NFA}}$.

To determine whether a cyclic path is accepted by $\llbracket e \rrbracket_{\text{NFA}}$, KATER treats G as another automaton, and takes its intersection with $\llbracket e \rrbracket_{\text{NFA}}$ ². Given the intersection, KATER searches for strongly connected components (SCCs) that contain at least one accepting state of $\llbracket e \rrbracket_{\text{NFA}}$. (Observe that such SCCs are guaranteed to represent cycles in G that are accepted by $\llbracket e \rrbracket_{\text{NFA}}$.) By using a depth-first-search algorithm (e.g., Tarjan’s SCC algorithm [5]), the complexity of the generated consistency-checking code is $\mathcal{O}(nm)$, where $n = |G|$ and $m = |\llbracket e \rrbracket_{\text{NFA}}|$.

6.2 Consistency Checking in KATI

When generating code for KATI expressions, we can employ the language representation of §3, as in the weak memory literature there is a disjoint set of relations used in intersections.

As such, we can extend KATER’s code-generating infrastructure by making the following observation: the language representation of the KATI expressions $\llbracket e \rrbracket_{\text{L}}$ and $\llbracket e \cap ir \rrbracket_{\text{L}}$ is the same, modulo the $(_{ir}$ symbols. This observation implies that in order to check for acyclicity of $e \cap ir$, we can use the procedure of §6.1 to enumerate all e -paths, and then simply restrict to paths whose endpoints are ir -matching (e.g., have the same location, if $ir = \text{sameLoc}$).

Such a restriction can easily be performed by using dedicated variables $v_{c,ir}$ for $ir \in \mathbf{R}$ and $0 < c \leq c(e)$. Whenever the intersection of $\llbracket e \cap ir \rrbracket_{\text{NFA}}$ and G encounters the symbol $(_{ir}$, the corresponding information of the respective graph event is saved in v_{ir} (e.g., the event’s location, if $ir = \text{sameLoc}$), and the exploration proceeds as normal. Subsequently, when the intersection encounters the matching $)_{ir}$, the exploration only proceeds if the corresponding information of the respective graph event matches the information stored in v_{ir} .

Incremental Consistency Checking

In certain scenarios like testing or stateless model checking [15], we know that a given graph G' is consistent, and we want to check whether an event a can be added in a particular way maintaining consistency.

Even though we can use the algorithm of §6.2 to check whether the newly constructed graph G is consistent, we can devise a more efficient procedure for checking G ’s consistency, inspired by the respective algorithm of Kokologiannakis et al. [14]. The key idea is that, since G' is consistent, any inconsistency in G will be caused by a (cyclic) path that passes through a . As such, we only have to find a cyclic path in G that starts from a and is also a word accepted by $\llbracket e \cap ir \rrbracket_{\text{NFA}}$. The only problem is that the word accepted by $\llbracket e \cap ir \rrbracket_{\text{NFA}}$ might not have a in the beginning, but rather in the middle of the word.

To solve this, we perform a variation of the algorithm using the following construction. First, we enforce that $\llbracket e \cap ir \rrbracket_{\text{NFA}}$ has a single starting/accepting state q_0 (e.g., by taking its reflexive-transitive closure), and we assume that G has a as its single starting/accepting state. Then, we run the algorithm, but instead of following the algorithm of §6.2 and look

² In this construction, all of G ’s states are considered starting/final.

for SCCs starting from any state of the product (i.e., for each state $\langle e, q_0 \rangle$, where $e \in G$), we can instead only look for SCCs starting from the states $\langle a, q \rangle$ of the product, where q is a state in $\llbracket e \cap ir \rrbracket_{\text{NFA}}$.

Observe that any such SCC that we find represents a consistency violation, as some permutation of the respective path in G is guaranteed to be accepted by $\llbracket e \cap ir \rrbracket_{\text{NFA}}$. Such an algorithm leads to better performance, as it essentially corresponds to taking all rotations of $\llbracket e \cap ir \rrbracket_{\text{NFA}}$ (instead of taking all rotations of G), and typically $|\llbracket e \cap ir \rrbracket_{\text{NFA}}| \ll |G|$.

7 Related Work and Conclusion

There has been an abundance of work building on Kleene Algebra (with Tests) [16].

Many works focus on extending KA(T) to particular program domains. [8] support more program transformations than plain KAT by adding mutable tests. Anderson et al. [3] develop an instance of KAT called NETKAT to model packet transmission in networks, Wagemaker et al. [23] extend NETKAT for concurrency. Hoare et al. [9] presents Concurrent KA (CKA), an extension of Kleene Algebra with a built-in operator modeling parallel composition, and Jipsen [10] extends CKA with tests. Kappé et al. [12] present an alternative foundation for the concurrent setting called KA with Observations (KAO), to which they subsequently add tests [13]. Pous et al. [19] show that a lot of KA variants that have extra assumptions or impose additional structure (e.g., KAO, NETKAT) fit into the framework of KA with Hypotheses, and provide modular proofs for various such variants.

Others focus on handling a richer algebraic structure. Pous and Wagemaker [21] present two variants of KAT with an additional **top** element: one that only supports $\llbracket e \rrbracket_G \subseteq \llbracket \text{top} \rrbracket_G$, and one that has the additional property that $\llbracket e \rrbracket_G \subseteq \llbracket e; \text{top}; e \rrbracket_G$. Ěsik and L. Bernátsky [6] extend KA with a converse operator, and prove equivalence between the language, relational and algebraic models. Brunet and Pous [4] prove that the equational theory of relation algebras that support union, intersection (with arbitrary relations) and concatenation, but do not support converse or the identity relation is decidable. Pous and Vignudelli [20] show that the equational theory of relation algebras that support concatenation, converse, arbitrary intersections and the identity relation (but neither union nor star!) is decidable.

As Pous and Wagemaker [21] note, however: “*The case of intersection (with or without converse or the various constants) is significantly more difficult, and remains partly open [...]*”. KATI attempts to tackle a useful instance of this problem by providing a decision procedure for KAT with intersections, assuming that intersections are restricted to primitive relations. Such a restriction is common when using KAT to describe weak memory consistency models, as per the work of Kokologiannakis et al. [14], which forms the basis for KATI.

8 Conclusion

In this paper, we have extended the results of Kokologiannakis et al. [14] to handle memory models containing intersections with uninterpreted relations. While this restriction on intersections appears sufficient for existing memory model definitions, it would definitely be nice to devise a more general technique that can handle arbitrary intersections. We leave the exploration of such a technique for future work.

References

- 1 Jade Alglave, Luc Maranget, Paul E. McKenney, Andrea Parri, and Alan Stern. Frightening small children and disconcerting grown-ups: Concurrency in the Linux kernel. In *ASPLOS 2018*, pages 405–418, New York, NY, USA, 2018. ACM. doi:10.1145/3173162.3177156.
- 2 Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM Trans. Program. Lang. Syst.*, 36(2):7:1–7:74, July 2014. doi:10.1145/2627752.
- 3 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In Suresh Jagannathan and Peter Sewell, editors, *POPL 2014*, 2014, pages 113–126. ACM, 2014. doi:10.1145/2535838.2535862.
- 4 Paul Brunet and Damien Pous. Petri automata for Kleene allegories. In *LICS 2015*, pages 68–79. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.17.
- 5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: <http://mitpress.mit.edu/books/introduction-algorithms>.
- 6 Zoltán Ésik and L. Bernátsky. Equational properties of Kleene algebras of relations with conversion. *Theor. Comput. Sci.*, 137(2):237–251, 1995. doi:10.1016/0304-3975(94)00041-G.
- 7 Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In *POPL 2005*, pages 110–121, New York, NY, USA, 2005. ACM. doi:10.1145/1040305.1040315.
- 8 Niels Bjørn Bugge Grathwohl, Dexter Kozen, and Konstantinos Mamouras. KAT + B! In Thomas A. Henzinger and Dale Miller, editors, *LICS 2014*, pages 44:1–44:10. ACM, 2014. doi:10.1145/2603088.2603095.
- 9 C. A. R. Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene algebra. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009*, volume 5710 of *LNCS*, pages 399–414. Springer, 2009. doi:10.1007/978-3-642-04081-8_27.
- 10 Peter Jipsen. Concurrent Kleene algebra with tests. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *RAMiCS 2014*, volume 8428 of *LNCS*, pages 37–48. Springer, 2014. doi:10.1007/978-3-319-06251-8_3.
- 11 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 12 Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Kleene algebra with observations. In Wan J. Fokkink and Rob van Glabbeek, editors, *CONCUR 2019*, volume 140 of *LIPICs*, pages 41:1–41:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.41.
- 13 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent Kleene algebra with observations: From hypotheses to completeness. *CoRR*, abs/2002.09682, 2020. doi:10.48550/arXiv.2002.09682.
- 14 Michalis Kokologiannakis, Ori Lahav, and Viktor Vafeiadis. Kater: Automating weak memory model metatheory and consistency checking. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. doi:10.1145/3571212.
- 15 Michalis Kokologiannakis, Azalea Raad, and Viktor Vafeiadis. Model checking for weakly consistent libraries. In *PLDI 2019*, New York, NY, USA, 2019. ACM. doi:10.1145/3314221.3314609.
- 16 Dexter Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3), 1997. doi:10.1145/256167.256195.
- 17 Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. Repairing sequential consistency in C/C++11. In *PLDI 2017*, pages 618–632, New York, NY, USA, 2017. ACM. doi:10.1145/3062341.3062352.

33:16 Automating Memory Model Metatheory with Intersections

- 18 Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, September 1979. doi:10.1109/TC.1979.1675439.
- 19 Damien Pous, Jurriaan Rot, and Jana Wagemaker. On tools for completeness of Kleene algebra with hypotheses. *CoRR*, abs/2210.13020, 2022. doi:10.48550/arXiv.2210.13020.
- 20 Damien Pous and Valeria Vignudelli. Allegories: Decidability and graph homomorphisms. In Anuj Dawar and Erich Grädel, editors, *LICS 2018*, pages 829–838. ACM, 2018. doi:10.1145/3209108.3209172.
- 21 Damien Pous and Jana Wagemaker. Completeness theorems for Kleene algebra with tests and top. *CoRR*, abs/2304.07190, 2023. doi:10.48550/arXiv.2304.07190.
- 22 SPARC International Inc. *SPARC architecture manual - version 8*. Prentice Hall, 1992.
- 23 Jana Wagemaker, Nate Foster, Tobias Kappé, Dexter Kozen, Jurriaan Rot, and Alexandra Silva. Concurrent NetKAT - modeling and analyzing stateful, concurrent networks. In Ilya Sergey, editor, *ESOP 2022*, volume 13240 of *LNCS*, pages 575–602. Springer, 2022. doi:10.1007/978-3-030-99336-8_21.

On Continuous Pushdown VASS in One Dimension

Guillermo A. Pérez   

University of Antwerp – Flanders Make, Antwerp, Belgium

Shrisha Rao  

University of Antwerp – Flanders Make, Antwerp, Belgium

Abstract

A pushdown vector addition system with states (PVASS) extends the model of vector addition systems with a pushdown stack. The algorithmic analysis of PVASS has applications such as static analysis of recursive programs manipulating integer variables. Unfortunately, reachability analysis, even for one-dimensional PVASS is not known to be decidable. So, we relax the model of one-dimensional PVASS to make the counter updates continuous and show that in this case reachability, coverability, and boundedness are decidable in polynomial time. In addition, for the extension of the model with lower-bound guards on the states, we show that coverability and reachability are NP-complete, and boundedness is coNP-complete.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages; Theory of computation → Concurrency

Keywords and phrases Vector addition systems, Pushdown automata, Reachability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.34

Funding Work supported by the Flemish inter-university (iBOF) “DESCARTES” project.

Acknowledgements We thank Georg Zetsche for help with the hardness proofs in the model with lower bounds; A. R. Balasubramanian for his comments on an early version of this work, and Tim Leys and Ritam Raha for useful discussions on the topic of (continuous) counter automata.

1 Introduction

Vector addition systems with states (VASS) are commonly used to model distributed systems and concurrent systems with integer variables. A VASS consists of a set of (control) states and a set of counters. Transitions between states are labelled with vectors of integers (usually encoded in binary) that are added to the current values of the counters. Importantly, transitions resulting in a counter value becoming negative are disallowed.

An equivalent way of understanding this model is to see the counters as unary-alphabet stacks. This alternative formulation has a natural extension obtained by adding one general stack to it. Pushdown VASS (PVASS), as they are usually called, can be used to model recursive programs manipulating integer variables [17, Sec. 6.2]. Arguably the most basic question one can attempt to answer algorithmically in a computational model is that of *reachability*. In the context of (pushdown) VASS, we ask whether a given target configuration (formed by the current state and the values of the counters) can be seen along a run from a given source configuration. While the complexity of reachability for VASS is now better understood [15, 6], it is not known to be decidable for PVASS and the best known lower bound is HYPERACK-hardness [14]. The problem is not known to be decidable even for one dimension and the known lower bound is PSPACE-hardness [7].

Motivated by the (complexity) gap in our understanding of reachability for PVASS, researchers have studied the problem for different relaxations of the model: A PVASS is *bidirected* [9] if the effect (on the stack and counters) of every transition can be (immediately) reversed; A \mathbb{Z} -PVASS [11] allows counters to hold negative values; A *continuous* PVASS [2] instead allows them to hold nonnegative values and counter updates labelling a transition



© Guillermo A. Pérez and Shrisha Rao;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 34; pp. 34:1–34:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 1** Previously known complexity bounds (in black) and our bounds (in green) for problems in PVASS and relaxations thereof.

		PVASS	B-PVASS	\mathbb{Z} -PVASS	CPVASS	lb-CPVASS
Gen.	Reach	HYPACK-h	$\in \text{ACK/TOWER-h}$ [9]	NP-comp.	NEXP-c	Undec. [1]
1-dim.	Reach	PSPACE-h	$\in \text{PSPACE}$	NP-comp.	P TIME-c	NP-c
	Cover	$\in \text{EXPSPACE}$	$\in \text{PSPACE}$ [9]	$\in \text{NP}$	P TIME-c	NP-c
	Bound	$\in \text{HYPACK}$	$\in \text{HYPACK}$	$\in \text{HYPACK}$	P TIME-c	coNP-c

can be scaled by any $\gamma \in (0, 1]$ when taking the transition. For all of these, reachability is known to be decidable. For some of them, lower complexity bounds for the special case of one dimension have also been established. See Table 1 for a summary of known results.

In this work, we study reachability, coverability, and boundedness for continuous PVASS in one dimension. The boundedness problem asks whether the set of all reachable configurations, from a given source configuration, is bounded. In turn, coverability asks whether a vector at least as large as the given vector can be reached. In contrast to reachability, coverability is known to be decidable and in EXPSPACE for PVASS in one dimension [16]. Similarly, boundedness is known to be decidable and in HYPERACK , this time in general, not only in one dimension [14].

Contributions

In this paper, we prove that, for continuous PVASS in one dimension, reachability, coverability, boundedness and even computing the infimum bound are PTIME -complete. We further show that if one adds to the model lower-bound guards on the states for the counter (thus allowing for a “tighter” relaxation of the original model, since one can now partially control the counter values before a transition), then reachability and coverability are NP -complete while boundedness is coNP -complete.

2 Preliminaries

We first recall a definition of pushdown automata. Then, we extend it to continuous PVASS.

2.1 Pushdown automata and Context-free grammars

► **Definition 1** (Pushdown automata). *A pushdown automaton (PDA, for short) is a tuple $\mathcal{P} = (S, \Sigma, \Gamma, \delta, s_0, \perp, F)$ where S is a finite set of states, Σ a finite (possibly empty) alphabet, Γ a finite stack alphabet, $s_0 \in S$ the initial state, $\perp \in \Gamma$ the initial stack symbol, $F \subseteq S$ a set of accepting states, and $\delta : S \times S \rightarrow (\Sigma \cup \epsilon) \times (\{a, \bar{a} \mid a \in \Gamma \setminus \perp\} \cup \epsilon)$ a partial function, where, a and \bar{a} denote pushing and popping a from the stack respectively.*

A *configuration* of a PDA \mathcal{P} is of the form $(s, w, \alpha) \in S \times \Sigma^* \times \Gamma^*$ where s represents the current state of the PDA, w the word read by the PDA until reaching the state s and α the current stack contents of the PDA (with the right being the “top” from which we pop and onto which we push). The *initial configuration* q_0 is (s_0, ϵ, \perp) .

A *run* of a PDA \mathcal{P} is of the form $\pi = q_0 q_1 \dots q_n$ where the $q_i = (s_i, w_i, \alpha_i)$ are configurations and for all $0 \leq i < n$, $\delta(s_i, s_{i+1})$ is defined, $w_{i+1} = w_i \cdot \delta(s_i, s_{i+1})_1$, $\alpha_{i+1} = \alpha_i$ if $\delta(s_i, s_{i+1})_2 = \epsilon$, $\alpha_{i+1} = \alpha_i \cdot a$ if $\delta(s_i, s_{i+1})_2 = a$, and $\alpha_{i+1} = \alpha_i \cdot \bar{a}$ if $\delta(s_i, s_{i+1})_2 = \bar{a}$. Above, $\delta(_, _)_i$ represents the i -th component. For any $Q \subseteq S$, we say the run reaches Q if $s_n \in Q$.

We focus on state reachability, that is, a run π of a PDA is *accepting* if q_0 is the initial configuration and $s_n \in F$.

The language of a PDA \mathcal{P} , denoted by $L(\mathcal{P})$, is the set of all words $w_n \in \Sigma^*$ read by accepting runs $q_0 \dots (s_n, w_n, \alpha_n)$ of \mathcal{P} . The *Parikh image* $\Phi(w)$ of a word $w \in \Sigma^*$, i.e. the vector in $\mathbb{N}^{|\Sigma|}$ such that its i^{th} component is the number of times the i^{th} letter of Σ (assuming an arbitrary choice of total order) appears in w .

Context-free grammars

CFGs, for short, are a model that is expressively equivalent to PDAs in terms of their languages. The models are logspace reducible to each other [13, Section 5.3].

► **Definition 2** (Context-free grammars). *A CFG is a tuple $G = (V, \Sigma, P, S)$, where V is a set of variables; Σ , a set of terminals; $P \subset V \times \{V, \Sigma\}^*$, a set of productions; and $S \in V$, the start symbol.*

A production $(A, w) \in P$ is written as $A \rightarrow w$, where the *production symbol* “ \rightarrow ” separates the *head* (a variable) of the production, to the left of \rightarrow , from the *body* (a string of variables and terminals) of the production, to the right of \rightarrow . Each variable represents a language, i.e., a (possibly empty) set of strings of terminals. The body of each production represents one way to form strings in the language of the head.

► **Example 3.** The grammar $G = (\{A\}, \{a, b\}, P, S = A)$ represents the set of all palindromes over $\{a, b\}$ where the productions are $A \rightarrow \epsilon, A \rightarrow a, A \rightarrow b, A \rightarrow aAa$ and $A \rightarrow bAb$. The word $abaaaba$, for example, is in the language of A since it can be obtained by $A \rightarrow aAa \rightarrow abAba \rightarrow abaAaba \rightarrow abaaaba$ where the fourth, fifth, again the fourth, and finally, the second production rules are applied, in that order.

Chomsky normal form (or CNF) [13, Section 4.5] is a normal form for CFGs with the restriction that all production rules can only be of the form $A \rightarrow BC$, or $A \rightarrow a$, or $S \rightarrow \epsilon$. Every CFG has an expressively equivalent CFG in CNF.

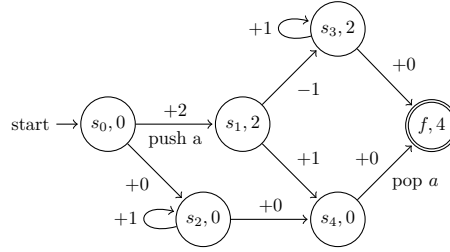
2.2 Continuous pushdown VASS

A CPVASS in one dimension is a PDA with a continuous counter.

► **Definition 4** (lb-C1PVASS). *A continuous pushdown VASS (with lower-bound guards) in one dimension is a tuple $\mathcal{A} = (S, \Sigma, \Gamma, \delta, \perp, s_0, F, \ell)$ where S is a finite set of states; $s_0 \in S$, the initial state; $F \subseteq S$, a set of accepting states; Σ , a finite alphabet; Γ , a finite stack alphabet; $\perp \in \Gamma$, the initial stack symbol; $\delta : S \times S \rightarrow (\Sigma \cup \epsilon) \times \mathbb{Z} \times (\{a, \bar{a} \mid a \in \Gamma \setminus \perp\} \cup \epsilon)$, a partial transition function; and $\ell : S \rightarrow \mathbb{N}$, a function that assigns lower bounds to states.*

Since we only study runs, and not languages, of lb-C1PVASS, we henceforth omit Σ . We also assume, without loss of generality, that the set F is a singleton. This can be done by adding a new final state f' to S and adding transitions for all $f \in F$ to f' which read ϵ , have a $+0$ counter update, and do not modify the stack. With these assumptions, we have a simpler representation of a lb-C1PVASS $\mathcal{A} = (S, \Gamma, \delta, \perp, s_0, f, \ell)$ where δ is now of the form $\delta : S \times S \rightarrow \mathbb{Z} \times (\{a, \bar{a} \mid a \in \Gamma \setminus \perp\} \cup \epsilon)$.

A *configuration* of a lb-C1PVASS is of the form (s, α, c) where s and α are as for PDAs, and $c \in \mathbb{R}_{\geq 0}$ is the current nonnegative value of the counter with the property that $c \geq \ell(s)$, that is, the counter value at a state must be at least the lower bound on that state. The *initial configuration* q_0 of the lb-C1PVASS is $(s_0, \perp, 0)$.



■ **Figure 1** An example of a lb-C1PVASS \mathcal{A} .

A *run* of the lb-C1PVASS \mathcal{A} is a sequence of configurations $\pi = q_0 q_1 \dots q_n$ with $q_i = (s_i, \alpha_i, c_i)$ such that $\pi|_{\mathcal{P}_{\mathcal{A}}}$, obtained by removing the counter values c_i , is a run in the PDA $\mathcal{P}_{\mathcal{A}}$, which is obtained by removing the counter updates from \mathcal{A} , and for all $0 \leq i < n$, $c_{i+1} = c_i + \gamma \delta(s_i, s_{i+1})_1$ holds for some $\gamma \in \mathbb{R} \cap (0, 1]$ where γ are the *scaling factors*.

► **Example 5.** Figure 1 shows a lb-C1PVASS with 6 states. The second component of the tuple inside the states denotes the lower bound on that state. For instance, $\ell(s_1) = 2$. This lb-C1PVASS does not have any run reaching f . This is because the only way to make the counter reach 4 is via s_2 or s_3 . The run through s_2 does not push an a into the stack which has to be popped later in order to reach f . Also, s_3 cannot be reached, since there are only two updates $+2$ and -1 before s_3 and $\gamma_1 \cdot 2 + \gamma_2 \cdot (-1) < 2$ for all $\gamma_1, \gamma_2 \in (0, 1]$.

Acceptance conditions of a lb-C1PVASS

There are two classical ways of extending (state reachability) acceptance from runs of a PDA to runs $\pi = q_0 \dots q_n$ of lb-C1PVASS, namely: *reachability* and *coverability* for $k \in \mathbb{R}_{\geq 0}$.

k -Reachability says the run is accepting if $\pi|_{\mathcal{P}_{\mathcal{A}}}$ is accepting in $\mathcal{P}_{\mathcal{A}}$ and $c_n = k$.

k -Coverability says the run is accepting if $\pi|_{\mathcal{P}_{\mathcal{A}}}$ is accepting in $\mathcal{P}_{\mathcal{A}}$ and $c_n \geq k$.

Like in PDAs, $q_0 = (s_0, \alpha_0, c_0) = (s_0, \perp, 0)$ means that accepting runs start with the initial configuration. We refer to accepting runs according to the above conditions as k -reaching and k -covering runs, respectively.

We make the following simplifying assumption. All the counter updates in the transition function are in the set $\{-1, +0, +1\}$. This is no loss of generality, due to the following lemma.

► **Lemma 6.** *Given a lb-C1PVASS $\mathcal{A} = (S, \Gamma, \delta, \perp, s_0, f, \ell)$, there exists an equivalent lb-C1PVASS¹ with counter updates in the set $\{-1, +0, +1\}$, which is quadratic in the size of the encoding of \mathcal{A} , thus polynomial even if the counter updates are encoded in binary.*

The proof follows from the construction of a simple gadget that takes as input the binary encoding of the update and outputs that exact number of $+1$ (or -1) updates.

We also study lb-C1PVASS where all lower-bound guards are 0 (a looser approximation of PVASS) where we give simpler algorithms to solve the decision problems we consider.

► **Definition 7 (C1PVASS).** *A C1PVASS is a lb-C1PVASS $\mathcal{A} = (S, \Gamma, \delta, \perp, s_0, f)$ where $\ell(s) = 0$ for all $s \in S$.*

For C1PVASS, we omit ℓ . *Configurations, runs, and accepting runs* are defined similarly to lb-C1PVASS. Note that, in a configuration (s, α, c) , instead of $c \geq \ell(s)$, we now only have the restriction that $c \geq 0$, that is, the counter values never go below 0.

¹ To be precise: there is a clear relation between their sets of reachable configurations.

► **Example 8.** In Figure 1, if all the lower bounds were 0, then we would be able to reach f with a counter value of at least 4 by taking the run to s_3 and taking the self loop a few times before entering f with a counter value of at least 4. However, the run via s_2 would still not be a 4-reaching run since there is no a to pop from the stack when the run reaches s_4 .

2.3 Decision problems

We focus on the computational complexity of two decision problems we call reachability and coverability, respectively: Given a lb-C1PVASS and $k \in \mathbb{N}$ (in binary), determine whether it has a k -reaching run. Given a lb-C1PVASS and $k \in \mathbb{N}$ (in binary), determine whether it has a k -covering run. In addition, we also study the complexity of the boundedness problem: Given a lb-C1PVASS, determine whether for some $k \in \mathbb{R}_{\geq 0}$ it has no k -covering run.

► **Remark 9.** Note that our definition of lb-C1PVASS is equivalent to the one where δ and ℓ map to rationals and $k \in \mathbb{Q}_{\geq 0}$, that is, $\delta : S \times S \rightarrow (\Sigma \cup \epsilon) \times \mathbb{Q} \times (\{a, \bar{a} \mid a \in \Gamma \setminus \perp\} \cup \epsilon)$ and $\ell : S \rightarrow \mathbb{Q}_{\geq 0}$, as one can multiply all counter updates, lower bounds and k by the product of all the denominators. Since the numbers are encoded in binary, the representation of the new integers will be polynomial in the size of the rationals (i.e. the bitsize of integer pairs). This preserves all the properties studied in this paper.

3 Counter properties of C1PVASS

We first show a relation between reachability and coverability.

► **Lemma 10.** *The reachability and coverability problems are equivalent for C1PVASS with $k > 0$, but 0-coverability does not imply 0-reachability.*

Proof. By definition, k -reachability implies k -coverability. To show the converse, take any covering run with counter value at the end of the run being $k + c$, for some $c \geq 0$. Now, we modify the run by scaling all of the counter updates in that run by $\frac{k}{k+c}$. The reader can easily verify that this is indeed a reaching run.

This proof does not work for $k = 0$ since we cannot scale the counter updates by 0. A simple example for the second part of the lemma would be a C1PVASS with a single transition, which goes from s_0 to f with a +1 counter update (and no stack update). In this case, 0 can be covered but not reached. ◀

From the proof above we directly get the following.

► **Remark 11.** Let $k \in \mathbb{N}_{>0}$. Then, for all $k' \in (0, k]$, k -reachability implies k' -reachability.

We also have the following simple observation about the first nonzero counter update due to our choice of q_0 .

► **Remark 12.** Along any run, the first nonzero counter update must be positive, since the updates cannot be scaled to 0 and the counter values must always be nonnegative

Since we have shown that k -reachability and coverability are different for $k = 0$ but the same for $k > 0$, we first analyse the complexity of 0-reachability and 0-coverability in Section 3.1. We then show, in Section 3.2, that boundedness is decidable in PTIME and that, if a C1PVASS is bounded, computing the infimum upper bound is also in PTIME. Finally, in Section 3.3, we leverage the size of the upper bound along with Remark 11 to show that k -reachability and k -coverability, for $k > 0$, are also in PTIME.

3.1 0-reachability and 0-coverability

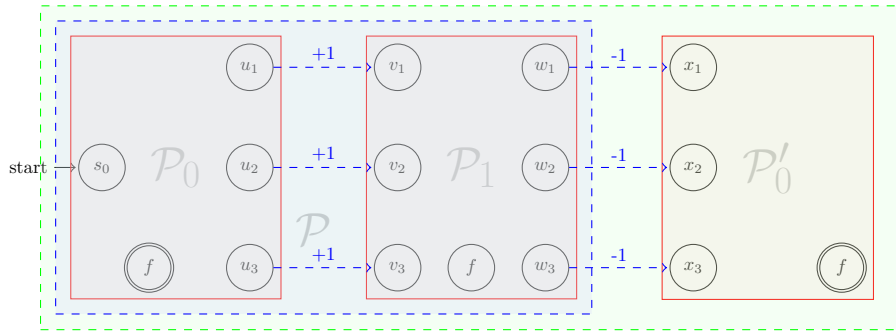
In this section, reduce both 0-reachability and 0-coverability to checking nonemptiness of PDAs (with an empty alphabet) to show the following:

► **Theorem 13.** *The 0-reachability and 0-coverability problems for a C1PVASS are decidable in PTIME.*

The result follows from the fact that checking nonemptiness of the language of a PDA can be done in polynomial time (see, e.g. [13, Proof of Lemma 4.1]) and the following lemma.

► **Lemma 14.** *The 0-reachability and 0-coverability problems for C1PVASS are polynomial time reducible to the nonemptiness of the language of a PDA.*

Proof sketch. For 0-coverability, first note the fact that any run in the underlying PDA of the C1PVASS \mathcal{A} corresponds to a 0-covering run in \mathcal{A} if and only if the first non-zero counter update in the run is a +1. We design a PDA as shown in Figure 2 (the blue dashed box) which simulates this property of 0-covering runs, where \mathcal{P}_0 and \mathcal{P}_1 are copies of the underlying PDA where \mathcal{P}_0 only has transitions corresponding to +0 updates in \mathcal{A} , and the run can enter \mathcal{P}_1 only after a +1 update.



■ **Figure 2** The construction of the PDA \mathcal{P} where \mathcal{P}_0 and \mathcal{P}'_0 are copies of \mathcal{A} obtained by the counter and removing all the transitions that have a nonzero counter update; the transitions from \mathcal{P}_0 to \mathcal{P}_1 are exactly the transitions in \mathcal{A} with a positive counter update and those from \mathcal{P}_1 to \mathcal{P}'_0 are exactly the ones with a negative counter update. For coverability, \mathcal{P} consists of just \mathcal{P}_0 and \mathcal{P}_1 and the copies of accepting states in \mathcal{P}_1 are also accepting in this case.

For reachability, note that a run in the underlying PDA of \mathcal{A} corresponds to a 0-reaching run in \mathcal{A} if and only if the first non-zero counter update is +1 and the last non-zero counter update is -1. Figure 2 (the green dashed box) simulates this by creating 3 copies of the underlying PDA of \mathcal{A} , namely, \mathcal{P}_0 , \mathcal{P}_1 and \mathcal{P}'_0 where \mathcal{P}_0 and \mathcal{P}'_0 only contain transitions corresponding to +0 updates in \mathcal{A} . ◀

3.2 Boundedness for C1PVASS

In this section, we first analyze the complexity of deciding whether a C1PVASS is bounded or not. If it is bounded, we provide a bound which is polynomial (when encoded in binary) in the size of the encoding of the C1PVASS. We next show that, for a bounded C1PVASS, the “tight” bound, that is,

$$b = \inf\{k \in \mathbb{R} \mid \mathcal{A} \text{ has no } k\text{-covering run}\} \tag{1}$$

is an integer and the natural decision problem associated to finding b is in PTIME. First, we convert the C1PVASS into a PDA \mathcal{P}' as we did in the proof of Lemma 14 in Figure 2 (the blue dashed box), further modify its alphabet, and observe some properties about the resulting PDA.

Let \mathcal{A} be the C1PVASS. Make two copies \mathcal{P}_0 and \mathcal{P}_1 of \mathcal{A} without the counter. Next, remove from \mathcal{P}_0 all the transitions that were not a +0 counter update in \mathcal{A} and add, for each transition in \mathcal{A} with a positive counter update, a transition from \mathcal{P}_0 to \mathcal{P}_1 . The copies of accepting states in \mathcal{P}_0 and \mathcal{P}_1 are all accepting in the resulting PDA, which we call \mathcal{P} (see the blue dashed box in Figure 2). To obtain \mathcal{P}' from \mathcal{P} , we modify its alphabet. The alphabet Σ of \mathcal{P}' is unary, i.e. $\Sigma = \{a\}$. The transitions of \mathcal{P}' read a if they had a +1 counter update in \mathcal{A} and read the empty letter ϵ otherwise.

One can see a relation between accepting runs in \mathcal{A} and \mathcal{P}' . Let π be an accepting run in \mathcal{P}' . The corresponding run in \mathcal{A} has the property that the first nonzero update is a positive update (i.e., a transition from \mathcal{P}_0 to \mathcal{P}_1) which makes it an accepting run in \mathcal{A} . Similarly, an accepting run in \mathcal{A} must have a +1 as the first nonzero update, hence, it is also an accepting run in \mathcal{P}' by construction.

The lemma below follows immediately from the construction.

► **Lemma 15.** *$a^m \in L(\mathcal{P}')$ if and only if there is an accepting run in \mathcal{A} with exactly m many +1 updates.*

For all $0 < \varepsilon < 1$, and an accepting run in the PDA \mathcal{P}' , in the corresponding run in \mathcal{A} , one can choose $\gamma = 1$ for all the +1 updates and $\gamma \in (0, 1]$ small enough, for all negative updates, so that their sum is in the interval $[0, \varepsilon)$. This leads to the following result.

► **Lemma 16.** *The cardinality of $L(\mathcal{P}')$ is bounded if and only if the C1PVASS \mathcal{A} is bounded. Moreover, if the maximum length of a word accepted by \mathcal{P}' is $p \in \mathbb{N}$ then $b = p$, where b is as in Equation (1).*

There are PTIME algorithms (see, e.g., [13, Theorem 6.6]) to determine whether the language of a PDA is finite. We thus get:

► **Theorem 17.** *Deciding boundedness of a C1PVASS \mathcal{A} is in PTIME. Moreover, the bound can be at most $2^{O(|\mathcal{A}|^6)}$, where $|\mathcal{A}|$ is the size of the encoding of \mathcal{A} .*

The first part of the proof follows from the discussion above. The bound is due to the facts that any PDA of size n can be converted to a CFG in CNF of size at most $O(n^6)$ (at most $2500n^6$ to be exact), the size of \mathcal{P} is at most twice that of \mathcal{A} , and 2^m is a bound on the length of the words accepted by a (bounded) CFG in CNF of size m .

Using this upper bound on the largest reachable counter for a bounded C1PVASS, we argue the tight upper bound is an integer and give an algorithm to compute it.

► **Remark 18.** Using Lemma 15 and the fact that nonnegative updates can be scaled down arbitrarily, one can see that the bound b defined in Equation (1) is a nonnegative integer when it exists.

► **Theorem 19.** *The tight upper bound of a bounded C1PVASS can be computed in PTIME.*

Proof. The idea for the proof comes from [8] which gives a PTIME algorithm to find the shortest word accepted by a CFG.

Assume the language is not empty. Construct the PDA described in Lemma 16. We know that if m is the length of a longest word accepted by the PDA \mathcal{P} , then m is the tight bound. We also know, by Theorem 17, that $m \leq 2^k$ where $k = O(|\mathcal{A}|^6)$. We construct a

grammar (V, Σ, P, S) in CNF for the PDA. This grammar has size at most $2^{O(|\mathcal{A}|)^6}$, as shown in Theorem 17. Since the grammar is in CNF, all productions are of the form $A \rightarrow BC$ or $A \rightarrow a$ and the language of all variables is nonempty.

Define the function $N : V \rightarrow \mathbb{N}$ such that $N(A)$ is the length of the longest word produced by the variable A , for all $A \in V$. The following algorithm computes $N(A)$ for all $A \in V$.

1. Initialize $W(A) = 0$ for all $A \in V$, $W(a) = 1$ for all $a \in T$.
2. Repeat, for all A and all productions with head A :

$$W(A) = \begin{cases} \max\{W(B) + W(C), W(A)\} & \text{if } A \rightarrow BC; \\ \max\{W(a), W(A)\} & \text{if } A \rightarrow a. \end{cases}$$

until we reach a fixed point (we know a fixed point will be reached eventually since the length of words is bounded).

3. Output the vector $W(V)$.

We know that the above algorithm terminates since the length of the longest word is bounded. It remains to show that it terminates in polynomially many iterations. Each iteration has $|V||P|$ comparisons of numbers bounded by $2^{O(|\mathcal{A}|)^6}$, and we know that such numbers can be compared in time polynomial in $|\mathcal{A}|$. Hence, showing that the fixed point is obtained in polynomially many iterations of the algorithm suffices to establish that the tight bound can be obtained in polynomial time.

Consider the directed graph with $V \cup \Sigma$ as vertices and where we add the edge (A, β) , where $A \in V$ and $\beta \in V \cup \Sigma$, if and only if there is a production in P whose head is A and with β in its body. The graph can be shown to be acyclic, since the language of the grammar is finite and the language of every variable is nonempty. Now, every iteration of the algorithm induces a labelling of the vertices of the graph via W . Observe that the label of a vertex only changes if the label of one of its immediate successors changes. It follows that the fixed point is reached after at most $|V|$ iterations. ◀

► **Lemma 20.** *The set of all reachable values in a C1PVASS is closed on the right (i.e., the bound b can be reached) if and only if there is an accepting run for a^b in \mathcal{P}' which does not contain any -1 transitions from \mathcal{A} .*

The proof follows from the simple fact that any -1 update in \mathcal{A} cannot be scaled down to 0, and b is an upper bound on the counter value in the final configuration of any accepting run.

Lemma 20 gives us an easy way to check whether the interval of all reachable counter values is closed on the right. Remove all transitions from \mathcal{P}' which correspond to a -1 update transition in \mathcal{A} . This PDA \mathcal{P}'' will accept a^m if and only if \mathcal{A} has an accepting run with exactly m many $+1$ updates and no negative updates. This can be checked in PTIME due to [5].

3.3 k -reachability and k -coverability for $k > 0$

The following stronger theorem implies that both k -reachability and coverability are in PTIME for all $k \geq 0$.

► **Theorem 21.** *The interval of all reachable counter values of a C1PVASS is computable in polynomial time.*

Proof. Use Theorem 13 to decide whether 0 is reachable. If so, the interval is closed on the left, open otherwise. Next, use Theorem 17 to decide if the highest reachable counter value is bounded. If not, the upper bound will be ∞ (and thus open). If it is bounded, use Theorem 19 to compute the tight bound b . Finally, use Lemma 20 to find whether the interval is closed on the right. ◀

PTIME-hardness for all the problems in this section follows from the nonemptiness problem for PDAs being PTIME-hard (see, e.g. [5, Prop. 1]). For coverability and reachability this is immediate, for boundedness one can add a self loop with a positive counter update on accepting states.

4 Counter properties of lb-C1PVASS

In this section, we show that coverability and reachability for lb-C1PVASS are decidable in NP and comment on how our treatment of boundedness from the previous section adapts almost identically to lb-C1PVASS to yield, in this case, a complexity of CONP. Finally, we provide a two-step reduction from the subset-sum problem to show completeness in the respective complexity classes.

Both for coverability and reachability, we proceed as follows. First, we convert the given lb-C1PVASS into a PDA \mathcal{P} such that the *Parikh image* of a word accepted by the PDA satisfies some quantifier-free Presburger formula φ if and only if the lb-C1PVASS has an accepting run. Then, we use a construction from [18, Theorem 4] (later corrected in [12]) to obtain, in polynomial time, an existential Presburger formula φ_L whose models correspond to the Parikh images of words in the language of \mathcal{P} . The problem thus reduces to checking satisfiability of the existential Presburger formula $\varphi \wedge \varphi_L$. The result follows since satisfiability for such formulas is known to be NP-complete [10].

► **Theorem 22.** *Deciding k -coverability and k -reachability for lb-C1PVASS is NP-complete, and boundedness is CONP-complete.*

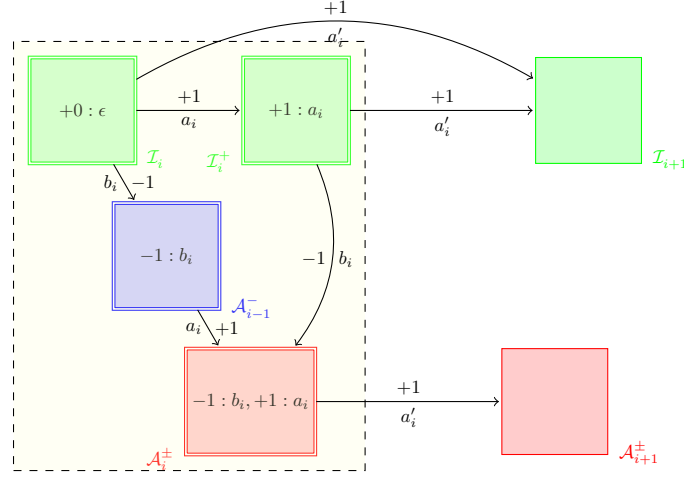
4.1 k -coverability for lb-C1PVASS

For lb-C1PVASS, k -coverability is equivalent to *state reachability*, i.e. without asking for the final counter value to be at least some given value: to check k -coverability, we add a new final state with lower-bound guard k and transitions from the old final state(s) to this new state with $+0$ counter updates and no stack update. Because of this, we focus on state reachability as acceptance condition and omit k when speaking of coverability in the sequel.

Let $\mathcal{A} = (S, \Gamma, \delta, \perp, s_0, f, \ell)$ be the lb-C1PVASS. Recall that $\ell : S \rightarrow \mathbb{N}$ is the mapping from states to the lower bounds on those states. That is, $\ell(s) = x$ implies that the counter value must be at least x in order to enter the state s . Let $n = |S|$ be the number of states. We have the assumption, from Lemma 6, that the only counter updates in the lb-C1PVASS are in the set $\{-1, +0, +1\}$. Let $m + 1 \leq n$ be the size of the range of ℓ . That is, there are $m + 1$ distinct lower bounds $0 = \ell_0 < \ell_1 < \ell_2 < \dots < \ell_m$ that occur in the lb-C1PVASS \mathcal{A} . Note that 0 must be one of the lower bounds since $\ell(s_0) = 0$ in order for any run to exist.

Now, we construct the PDA, followed by the Presburger formula. The PDA \mathcal{P} has $4(m + 1)$ “blocks” and its alphabet is $\Sigma = \{a_i, a'_i, b_i \mid 0 \leq i \leq m + 1\}$. Each block is a subPDA (so, we ignore counter updates) of the lb-C1PVASS with some restrictions. For each $0 \leq i \leq m$, the 4 types of blocks we use all have copies of the same set of states: all $s \in S$ such that $\ell(s) \leq \ell_i$.

1. \mathcal{I}_i The transitions come from those in \mathcal{A} with counter update $+0$ and they read ϵ in \mathcal{P} ;
2. \mathcal{I}_i^+ The transitions come from those in \mathcal{A} with $+0$ or $+1$ updates and the PDA \mathcal{P} reads an a_i on the $+1$ transitions;
3. \mathcal{A}_i^- The transitions come from those in \mathcal{A} with $+0$ or -1 updates and the PDA \mathcal{P} reads a b_i on the -1 transitions;
4. \mathcal{A}_i^\pm And here, all transitions in \mathcal{A} are present and the PDA will read b_i on -1 and a_i on the $+1$ transitions.



■ **Figure 3** A slice of the PDA \mathcal{P} constructed for k -coverability of a lb-C1PVASS. The subscript being i for $0 \leq i \leq m$ of a block (for example, i in \mathcal{I}_i) denotes that all the states in the block have lower bounds at most ℓ_i . Note $+0 : \epsilon$ is omitted unless it is the only option for transitions in the block.

Figure 3 depicts how the blocks are connected in what we henceforth call a *slice* (depicted by the dashed box), i.e. \mathcal{I}_i , \mathcal{I}_i^+ , \mathcal{A}_{i-1}^- and \mathcal{A}_{i+1}^\pm , for some $0 \leq i \leq m$. It also shows how the slices themselves are connected. Note that the transitions in the PDA do not actually have the counter updates $-1, +0, +1$, but we include them in the explanation and figure for clarity. The accepting states of \mathcal{P} are all the copies of accepting state in \mathcal{A} .

Now, we define a Presburger formula for the Parikh images of accepting runs in \mathcal{P} that correspond to the accepting runs in \mathcal{A} ($\#_a$ denotes the number of a 's read during the run).

$$\bigwedge_{k=1}^m \left(\left(\#_{a'_{k-1}} = 0 \right) \vee \left(\left(\sum_{i=0}^{k-1} \#_{a_i} + \#_{a'_i} \geq \ell_k \right) \wedge \left(\sum_{i=0}^{k-1} \#_{b_i} = 0 \right) \right) \vee \left(\left(\sum_{i=0}^{k-1} \#_{a_i} + \#_{a'_i} > \ell_k \right) \wedge \left(\sum_{i=0}^{k-1} \#_{b_i} > 0 \right) \right) \right) \quad (2)$$

Intuitively, the second disjunct ensures that if the run saw no negative updates (stayed in the green layer), then the number of $+1$ updates in order to enter the k^{th} slice must be at least the k^{th} lower bound; The third disjunct ensures that if there was a negative update seen then the number of $+1$ updates seen must be strictly greater than the k^{th} lower bound; and the first disjunct is if the run never enters the k^{th} slice.

► **Theorem 23.** *For all runs π in \mathcal{A} , there is one in \mathcal{P} with the same sequence of states whose Parikh image satisfies the Presburger formula from Equation (2) if and only if π is accepting in \mathcal{A} .*

The following auxiliary lemmas are helpful in the intuition for the proof of the theorem.

► **Lemma 24.** *Let $0 < \varepsilon < 1$. For any run π in \mathcal{A} , there is another run π' with the same sequence of states such that all the $+1$ counter updates in the run are scaled up to 1 and all the -1 updates in the run are scaled down so as to add up to $-\varepsilon$.*

The proof follows from a few simple observations: Since π is a run and π' is a run with the same sequence of states, the stack will behave the same in both runs. For the counter, since we only have lower bounds, and we chose small coefficients for negative updates, all the counter updates in π' are greater than, or equal to the counter updates in π , hence satisfying all lower bounds along the run.

► **Lemma 25.** *A run ends in a green state (i.e., a state in \mathcal{I}_i or \mathcal{I}_i^+ for some $0 \leq i \leq m$) in \mathcal{P} if and only if there was no b_j read along the run for all $0 \leq j \leq m$.*

This follows from the construction since there is no path from the blue states (states in \mathcal{A}_i^- for $0 \leq i \leq m$) or the red states (states in \mathcal{A}_i^\pm for $0 \leq i \leq m$) to any green state. Intuitively, the counter values are integers when reaching green states and nonintegers when reaching blue or red states.

This gives us NP inclusion for deciding coverability in lb-C1PVASS.

4.2 k -reachability for lb-C1PVASS

Here, we show that k -reachability for $k \in \mathbb{N}$ is also in NP for lb-C1PVASS. Unlike for C1PVASS, k -coverability does not imply k -reachability in lb-C1PVASS: scaling down the vectors along the entire run can lead to some lower bounds being violated. E.g., consider a lb-C1PVASS with 3 states s_0, s_1 and f with $\ell(s_1) = 1$ and a $+1$ update on both $s_0 \rightarrow s_1$ and $s_1 \rightarrow f$. For any accepting run π , the counter value at s_1 must be 1. This means that even if the second $+1$ update is scaled down, the counter value at f must be strictly greater than 1. For this lb-C1PVASS, 1-coverability holds but 1-reachability does not.

Like in the previous section, we construct a PDA and a Presburger formula such that the PDA accepts a word that satisfies the Presburger formula if and only if the lb-C1PVASS has a k -reaching run, for some given $k \in \mathbb{N}$. However, the construction is more involved since it is not always possible to reach a specific counter value by scaling down all negative counter updates to arbitrarily small numbers. Instead, we first introduce a normal form of scaled runs (i.e. the sequences of coefficients) that guides our construction for a PDA with no block cycles in the same way Lemma 24 guided our construction for reachability.

4.2.1 The Dense Normal Form (DNF)

We show that all k -reaching runs have a normal form which scales the counter updates in the run so that the positive updates are concentrated towards the start of the run and the negative updates towards the end of the run. Formally, let π be a run in the lb-C1PVASS which reaches the counter value $k \in \mathbb{N}$. Let P_π be the sum of all positive updates in π and N_π be the sum of all negative updates. Define $I_P^\pi = \lfloor P_\pi \rfloor$, $F_P^\pi = P_\pi - I_P^\pi$, $I_N^\pi = \lceil N_\pi \rceil$ and $F_N^\pi = N_\pi - I_N^\pi$. Clearly $I_P^\pi + F_P^\pi + F_N^\pi + I_N^\pi = k$ and, moreover, $I_P^\pi + I_N^\pi = k$ and $F_P^\pi = -F_N^\pi$ since k is an integer. To define a normal form and argue all runs can be put in it, we scale consecutive positive updates and consecutive I_N^π negative updates at the start and at the end the run in full, i.e. $\gamma = 1$ (except for a special case where we scale one of the negative updates by Δ close to 1). The remaining positive and negative updates can be scaled arbitrarily (small) as long as their sum adds up to 0. For the latter, we will scale down positive and negative updates by coefficients from E and D respectively, where E and D are finite sets of arbitrarily small “epsilons” (ε) and “deltas” (δ) in the interval $(0, 1)$. The updates are concentrated at the start and the end of the run, hence the name *dense* normal form. For simplicity, we also add Δ to D .

► **Lemma 26.** *Let π be a run in the lb-C1PVASS which reaches the counter value $k \in \mathbb{N}$. Then, there exists a run π' such that:*

1. *The sequence of states in π' is the same as in π , and*
2. *π' is also a k -reaching run.*
3. *All positive and negative updates in π' are scaled from the set $\{1\} \cup E$ and $\{1\} \cup D \cup \Delta$, respectively.*
4. *The sequence of all nonzero updates in π' is of the form $(\{+1\} \cup -D)^*(-D \cup +E)^*(\{-1\} \cup +E)^*$ or $(\{+1\})^*(-D)^+(-\Delta)(\{-1\})^*$.*
5. *Let $\pi'|_{E \cup D}$ be the sequence of counter updates restricted to $+E$'s and $-D$'s. Then, $\pi'|_{E \cup D}$ is of the form:*
 - *$(-D)^*(+E)(+E \cup -D)^*(-D)$, in which case, for any proper prefix of $\pi'|_{E \cup D}$ with at least one epsilon, the sum of all epsilons and deltas is positive; or*
 - *$(+E \cup -D)^*(-D)(+E)^+$, and for any proper prefix of $\pi'|_{E \cup D}$ that contains an epsilon and not the final delta, the sum of all epsilons and deltas is positive while for any proper prefix with the final delta, the sum is negative; or*
 - *$(-D)^+(-\Delta)$, and none of the deltas occur before a $+1$ counter update.*
6. *For configurations $q = (s, \alpha, c)$ in π and $q' = (s, \alpha, c')$ in π' which occur in the same position in both runs, $c' \geq \lfloor c \rfloor$.*

Note that the DNF (in particular Item 4) precludes -1 updates before a $+1$ update. It also visits the same sequence of states, satisfies all lower-bound guards along the run and reaches the same counter value k as the given run.

► **Example 27.** Let the sequence of nonzero updates in a 1-reaching run be $+1, +0.8, -0.9, -0.9, +1, -1, +1$. This run is not in the dense normal form. To transform it into the normal form, we choose different values of γ to scale the updates along the run to obtain $+1, +1, -\delta_1, -\delta_2, +\varepsilon_1, -1, +\varepsilon_2$. It is easy to see that this is also a 1-reaching run and the integer parts of the counter values along the run are at least those along the original run.

Since, if a k -reaching run exists, a k -reaching run in dense normal form exists, we construct a PDA and a Presburger formula which simulate runs of the lb-C1PVASS in DNF.

4.2.2 Constructing the PDA \mathcal{P}

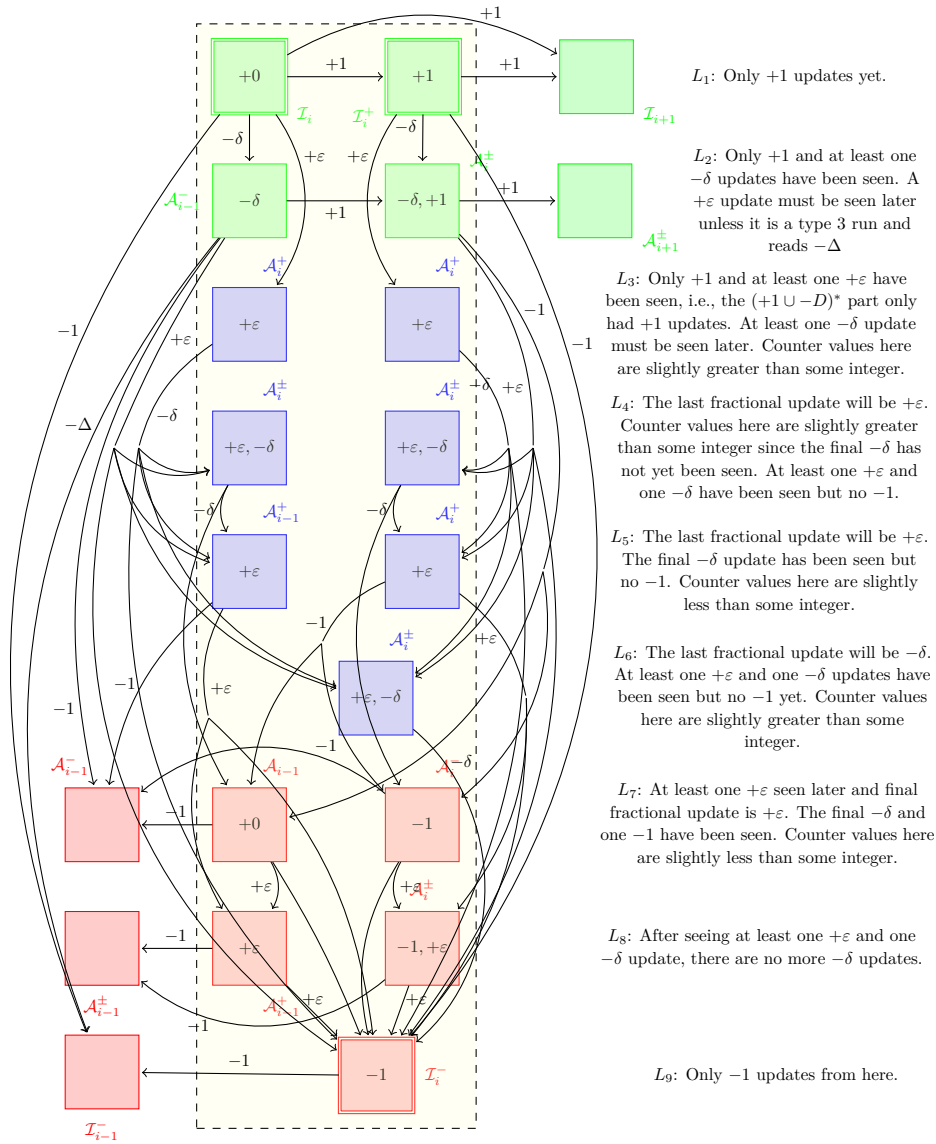
The construction of \mathcal{P} is shown in Figure 4. Each slice consists of 9 layers, namely L_1, \dots, L_9 , each of which contains one or two blocks. Each block, like in Section 4.1 is a copy of the lb-C1PVASS \mathcal{A} restricted to states with lower bounds at most ℓ_i , where i is the subscript of the block, and transitions with updates written inside the block. The transitions labelled $+\varepsilon$ and $-\delta$ correspond to $+1$ and -1 update transitions respectively, but we label them differently to make the explanation later easier to read. Each block also has all transitions from \mathcal{A} with a $+0$ update. Note that the connections between blocks allow exactly the transitions on the labels and not the $+0$ transitions from \mathcal{A} .

In the figure, we are assuming that $\ell_{i-1} < \ell_i - 1$. If $\ell_{i-1} = \ell_i - 1$, there will be the following changes:

1. All transitions entering \mathcal{A}_{i-1}^- will now go to \mathcal{A}_{i-2}^- instead (both in L_7), and
2. All transitions entering \mathcal{I}_{i-1}^- will now go to \mathcal{I}_{i-1} instead.

Checking whether $\ell_{i-1} = \ell_i - 1$ can be done beforehand for all $0 < i \leq m$, and \mathcal{P} can be constructed accordingly. The discussion that follows still holds.

Due to the complexity of the PDA, a fully formal proof like in Section 4.1 would obscure rather than support our claims. Instead, we focus on providing the high-level intuition behind the construction of the PDA and the Presburger formula.



■ **Figure 4** The i^{th} slice of the PDA \mathcal{P} constructed for k -reachability of a lb-C1PVASS. The slice itself is inside the dashed box, the text to the right provides intuition for the layer. The subscript being i for $0 \leq i \leq m$ of a block (eg, i in \mathcal{I}_i) denotes that all the states in the block have lower bounds at most ℓ_i . Note $+0$ is omitted unless it is the only option for transitions in the block.

► **Remark 28.** By construction, there are no block cycles in \mathcal{P} . That is, once a run exits a block, it cannot enter the same block later. This follows due to the simple observation that, from every block, there are transitions only to a block either to the left or to the right on the same level (but never both), or a block in a lower layer.

Note that, since there is no way to enter a block in an upper layer from a lower one, and a run always starts from \mathcal{I}_0 (the leftmost green block), any run will first traverse green blocks, moving to the right, then it will either enter a blue block and move down or directly enter a red block and start moving to the left.

34:14 On Continuous Pushdown VASS in One Dimension

► **Lemma 29.** *The sequence of states in any run in \mathcal{P} is a sequence of states in green blocks, followed by a (possibly empty) sequence of states in blue blocks, followed by a (possibly empty) sequence of states in red blocks. Furthermore, while the run is visiting green, blue and red blocks, the index of the slices is non-decreasing, constant and non-increasing, respectively.*

The proof follows by construction.

We now show how the PDA \mathcal{P} is split into 3 main components and the letters read on the transitions in slice i .

- The **green** component consists of the first two layers. This component corresponds to the $(+1 \cup -D)^*$ part of the run in Item 4. This is easy to see since this component looks exactly like Figure 3, with the exception that the states in the second layer (which were the states in blue and red states in Figure 3), are not accepting. The PDA reads alphabet a_i on all $+1$ transitions in and to green blocks in Figure 4, except the transitions entering the next slice, on which it reads a'_i , and it reads d_i on all the $-\delta$ transitions.
- The **blue** component consists of layers 3, 4, 5 and 6 which correspond to the $(+E \cup -D)^*$ part of the run. This component is entered after reading the first $+\varepsilon$ update. Due to Item 5, the run stays in a single slice during this part of the run. The empty letter ϵ is read on all transitions in and to this component.
- The **red** component, consisting of the last 3 layers, corresponds to the $(-1 \cup +E)^*$ part of the run, and is entered after the first -1 or the last fractional update. Note that the run can never exit this component once it is entered. The PDA reads b'_i on all the -1 transitions in and to L_7 , b_i on all -1 transitions in and to L_8 and L_9 , ϵ on all other transitions and a d'_i on the $-\Delta$ transition entering L_9 . Note that $\sum_{i=1}^m \#_{d'_i} \in \{0, 1\}$ since a $-\Delta$ transition can occur at most once in any run.

► **Lemma 30.** *If the lb-C1PVASS accepts some run in DNF which reaches some $k \in \mathbb{N}$, there exists an accepting run in the PDA \mathcal{P} such that $\sum_{i=1}^m \#_{a_i} + \#_{a'_i} - \#_{b_i} - \#_{b'_i} - \#_{d'_i} = k$.*

The proof follows from the fact that the run (in DNF) can be simulated on the PDA \mathcal{P} by staying in the green component for the $(-1 \cup -D)^*$ portion of the run, then going to the $(+E \cup -D)^*$ portion of the run, and finally the run enters the red portion for the $(-1 \cup +E)^*$ part of the run (except for the case where the run is of the form $(\{+1\})^*(-D)^+(-\Delta)(\{-1\})^*$, in which case the run enters the last layer on the $-\Delta$ update). The run moves between layers as the counter values move between different slice bounds.

4.2.3 The Presburger formula

The main intuition for having the Presburger formula is to make sure that the run stays in the correct block. For example, on reading a $+1$ in \mathcal{I}_i^+ , there is a choice to either stay within the block or move to \mathcal{I}_{i+1} . The formula ensures that it stays in \mathcal{I}_i^+ when the counter value is in the interval (ℓ_i, ℓ_{i+1}) and moves to \mathcal{I}_{i+1} when the counter value reaches ℓ_{i+1} .

We also use a formula to ensure that the sum of all the $+1$ and -1 updates is exactly k .

Using Lemma 29, we are able to split the Presburger formula into 4 conjuncts as well.

$$\begin{aligned}
\varphi_G &= \bigwedge_{k=1}^m \left(\left(\#_{a'_{k-1}} = 0 \right) \vee \left(\left(\sum_{i=0}^{k-1} \#_{a_i} + \#_{a'_i} \geq \ell_k \right) \wedge \left(\sum_{i=0}^{k-1} \#_{d_i} = 0 \right) \right) \vee \right. \\
&\quad \left. \left(\left(\sum_{i=0}^{k-1} \#_{a_i} + \#_{a'_i} > \ell_k \right) \wedge \left(\sum_{i=0}^{k-1} \#_{d_i} > 0 \right) \right) \right) \\
\varphi'_R &= \bigwedge_{k=1}^m \left(\left(\#_{b'_k} = 0 \right) \vee \left(\sum_{i=1}^m \#_{a_i} + \#_{a'_i} - \sum_{j=k}^m \#_{b'_j} > \ell_k \right) \right) \\
\varphi_R &= \bigwedge_{k=1}^m \left(\left(\#_{b_k} + \#_{d'_{k+1}} = 0 \right) \vee \left(\sum_{i=1}^m \#_{a_i} + \#_{a'_i} - \sum_{j=k}^m \#_{b'_j} + \#_{b_j} + \#_{d'_{j+1}} \geq \ell_k \right) \right) \\
\varphi_k &= \sum_{i=0}^m \#_{a_i} + \#_{a'_i} - \#_{b_i} - \#_{b'_i} - \#_{d'_i} = k
\end{aligned}$$

The final formula φ will be a conjunction of all the formulas described above.

$$\varphi = \varphi_G \wedge \varphi'_R \wedge \varphi_R \wedge \varphi_k. \quad (3)$$

Now, with the PDA \mathcal{P} and formula φ , we get the main result of this section.

► **Theorem 31.** *For all runs π in \mathcal{A} , there is one in \mathcal{P} with the same sequence of states whose Parikh image satisfies the Presburger formula φ from Equation (3) if and only if π is k -reaching in \mathcal{A} .*

The proof is similar to that of Theorem 23, simulating the k -reaching runs of \mathcal{A} using accepting runs in \mathcal{P} satisfying φ and vice-versa.

4.3 Boundedness for lb-C1PVASS

For boundedness, we first note that the set of all reachable counter values from a lb-C1PVASS \mathcal{A} is a subset of all reachable values of the C1PVASS \mathcal{A}' , where \mathcal{A}' is obtained by replacing the lower bounds in \mathcal{A} by 0. It follows, from Theorem 17, that $2^{O(|\mathcal{A}|^6)}$ is an upper bound on the largest reachable counter value in \mathcal{A} , if \mathcal{A} is bounded, where $|\mathcal{A}|$ is the size of the encoding of \mathcal{A} . Hence, we can ask for k -coverability with $k = 2^{O(|\mathcal{A}|^6)}$ and if the answer is yes, the set of all reachable counter values is unbounded. Hence, boundedness is in CONP.

4.4 Hardness of k -reachability, k -coverability and boundedness

We now show that reachability and coverability are NP-hard for lb-C1PVASS and CONP-hard for boundedness. This gives us NP-completeness for reachability and coverability, and CONP-completeness for boundedness.

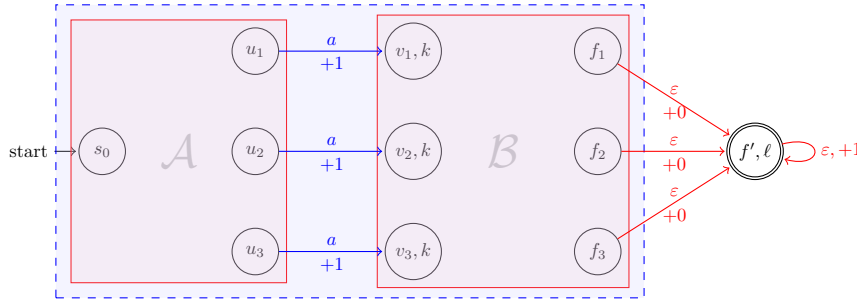
► **Lemma 32** (Context-free sum problem). *Given a context-free language $\mathcal{L} \subseteq a^*b^*$ and two numbers $k, \ell \in \mathbb{N}$ (encoded in binary), determining whether there exists a word $a^m b^n \in \mathcal{L}$ such that $m \geq k$ and $m + n \geq \ell$ is NP-hard.*

Proof. We reduce from the subset sum problem: Given a set of positive integers $S = \{x_1, \dots, x_n\}$ and an integer X , is there a subset $S' \subseteq S$ such that $\sum_{x_i \in S'} x_i = X$?

34:16 On Continuous Pushdown VASS in One Dimension

We construct the CFG as follows: We have non-terminals A_0, \dots, A_n where A_0 is the start symbol, and productions $A_i \rightarrow a^{x_{i+1}} A_{i+1}$, $A_i \rightarrow A_{i+1} b^{2x_{i+1}}$, for all $0 \leq i \leq n-1$, and $A_n \rightarrow \varepsilon$. This CFG defines the following language: $\mathcal{L} = \{a^r b^s \mid \exists S' \subseteq S : r = \sum_{x_i \in S'} x_i \text{ and } s = 2 \sum_{j \notin S'} x_j\}$. One can easily verify that a word $a^n b^m$ satisfying $n \geq X$ and $n + m \geq 2 \sum_{i=1}^n x_i - X$ is in \mathcal{L} if and only if the subset sum instance is positive. ◀

We now give the reduction from this problem to reachability, coverability and unboundedness in an lb-C1PVASS. Without loss of generality, we assume that the context-free language is given as a PDA such that there is a partition between all the states with an outgoing transition labelled by an a and those with a transition labelled by a b as shown in Figure 5.



■ **Figure 5** The construction of the lb-C1PVASS where \mathcal{A} is the part of the PDA consisting of only the transitions which do not read b 's and on the last a , there is a check verifying that at least k many a 's were seen, before seeing the b 's. From the final states, there is a transition to the new final state f' with a lower bound of ℓ which checks that the total number of a 's and b 's is at least ℓ .

To show that this construction establishes a reduction from the context-free sum problem to reachability, coverability and unboundedness, it is enough to see that an accepting run from the PDA will be accepting in this C1PVASS if and only if the total number of a 's is at least k and the total number of a 's and b 's is at least ℓ . Thus ℓ is reachable if and only if it is coverable if and only if there exists a solution to the context-free sum problem. Finally, due to the $+1$ self loop on the final state, there are accepting runs with unbounded value if and only if the context-free sum instance is positive. This gives us NP-hardness for reachability and coverability, and CONP-hardness for boundedness.

5 Conclusion

In this work we established reachability, coverability, and boundedness are decidable in polynomial time for continuous PVASS in one dimension (C1PVASS). When the model is extended with lower-bound guards for the counter on the states, we proved reachability and coverability are NP-complete while boundedness is CONP-complete. There are cells of Table 1 for which complexity bounds can be tightened in the future. Our algorithms can be used as heuristics to guide the exact algorithm for the reachability problem if it is decidable (cf. [3]). In the direction of using C1PVASS as approximations of PVASS, we posit the most interesting direction is to add both upper and lower bounds to the values the counter can take (cf. [4]) towards an approximation of one-counter pushdown automata. This model with upper and lower bounds can be seen as a generalization of one clock pushdown timed automata without resets.

References

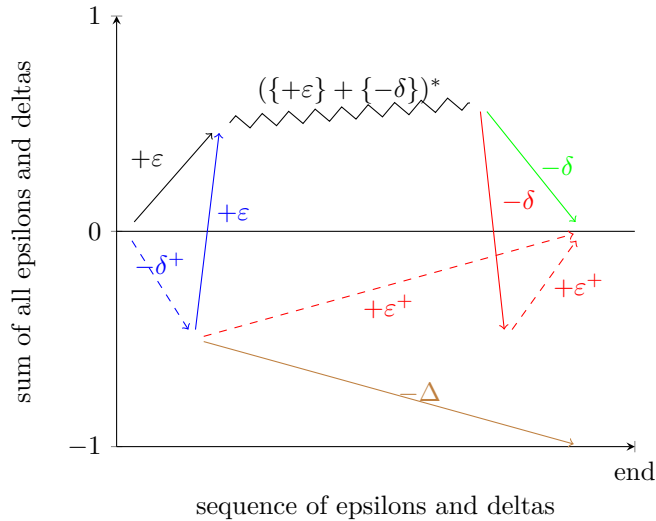
- 1 A. R. Balasubramanian. Decidability and complexity of decision problems for affine continuous VASS. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 7:1–7:13. ACM, 2024. doi:10.1145/3661814.3662124.
- 2 A. R. Balasubramanian, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. Reachability in continuous pushdown VASS. *Proc. ACM Program. Lang.*, 8(POPL):90–114, 2024. doi:10.1145/3633279.
- 3 Michael Blondin, Christoph Haase, and Philip Offtermatt. Directed reachability for infinite-state systems. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021. doi:10.1007/978-3-030-72013-1_1.
- 4 Michael Blondin, Tim Leys, Filip Mazowiecki, Philip Offtermatt, and Guillermo A. Pérez. Continuous one-counter automata. *ACM Trans. Comput. Log.*, 24(1):3:1–3:31, 2023. doi:10.1145/3558549.
- 5 Dmitry Chistikov and Rupak Majumdar. Unary pushdown automata and straight-line programs. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 146–157, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- 6 Wojciech Czerwinski, Sławomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. *J. ACM*, 68(1):7:1–7:28, 2021. doi:10.1145/3422822.
- 7 Matthias Englert, Piotr Hofman, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Juliusz Straszyński. A lower bound for the coverability problem in acyclic pushdown vas. *Information Processing Letters*, 167:106079, 2021.
- 8 Yuval Filmus. Hardness of finding a word of length at most k accepted by a nondeterministic pushdown automaton. <https://cstheory.stackexchange.com/questions/4429/hardness-of-finding-a-word-of-length-at-most-k-accepted-by-a-nondeterministic>, 2011.
- 9 Moses Ganardi, Rupak Majumdar, Andreas Pavlogiannis, Lia Schütze, and Georg Zetsche. Reachability in bidirected pushdown VASS. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 124:1–124:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.124.
- 10 Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.
- 11 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 743–759. Springer, 2011. doi:10.1007/978-3-642-22110-1_60.
- 12 Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2012.
- 13 John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

- 14 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-ackermannian bounds for pushdown vector addition systems. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 63:1–63:10. ACM, 2014. doi:10.1145/2603088.2603146.
- 15 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.
- 16 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *International Colloquium on Automata, Languages, and Programming*, pages 324–336. Springer, 2015.
- 17 Sylvain Schmitz. The complexity of reachability in vector addition systems. *ACM SIGLOG News*, 3(1):4–21, 2016. doi:10.1145/2893582.2893585.
- 18 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005.

A Appendix

A.1 Proof of Lemma 26

Proof. We first construct a run π'' which satisfies $I_P^{\pi''} = I_P^{\pi}$ and $I_N^{\pi''} = I_N^{\pi}$. Notice that none of the lower bounds along the run π can exceed I_P^{π} .



■ **Figure 6** A graphical representation of the multiple cases.

π must have at least I_P^{π} positive and I_N^{π} negative updates due to Lemma 6. The idea is to scale up the first I_P^{π} many +1 updates and the last I_N^{π} many -1 updates in the π'' by 1 (call this the *1-scaling*), and scale the remainder of +1 and -1 updates by updates in E and D . There are two major cases for this:

- 1. The I_P^{π} 'th 1-scaled +1 update occurs before the first 1-scaled -1 update.
- 2. The I_P^{π} 'th 1-scaled +1 update occurs after the first 1-scaled -1 update.

If the first case holds, π'' is of the form $(\{+1\} \cup -D)^* (-D \cup +E)^* (\{-1\} \cup +E)^*$ and we scale the updates in E and D according to Figure 6 on a case-by-case basis:

- There are exactly I_P^π many positive updates and I_N^π many negative updates in π (which means that all of them must have been scaled to 1 and $F_P^\pi = F_N^\pi = 0$), $\pi'' = \pi' = \pi$ and the bounds are satisfied.
- The number of positive updates in π is strictly greater than I_P^π and has exactly I_N^π many negative updates. π'' is of the form $(+1)^*(\{-1\} \cup E)^*$. Note that π'' does not reach k , but $k + e$ where e is the sum of all epsilons in the run. Since all the -1 updates are 1-scaled even in π , there must be some update in the first $I_P^\pi + 1$ updates which is not 1-scaled. Also, the lower bound in any state in π is at most $I_P^\pi - 1$. For π' , we scale down the I_P^π 'th $+1$ update in π'' to $+\varepsilon$ (the solid black edge in Figure 6) and the first 1-scaled -1 update to $-\delta$, such that the sum of all the epsilons and deltas is now 0. The lower bounds until the I_P^π 'th positive update (i.e., the first epsilon update) are satisfied since they were satisfied in π and the counter value after this sequence of update will be $I_P^\pi - 1 + \varepsilon$. The counter values until the (only) delta updates will be satisfied since the counter values will be between $I_P^\pi - 1$ and I_P^π . After the delta update, the counter values are satisfied in π' because they were satisfied in π , both runs reach k and π' scales up the -1 updates by 1 and scaled down the $+1$ updates to ε in this section of the run (intuitively, reaching the same number with lower positive updates and higher negative updates implies that the counter values must be better).

- The number of negative updates in π is strictly greater than I_N^π and has exactly I_P^π many positive updates. π'' is of the form $(\{+1\} \cup D)^*(-1)^*$. Note that π'' does not reach k , but $k - d$ where d is the sum of all deltas in the run. Here, we encounter the special case which calls for the need of Δ , that is there might be no negative update in π before a positive update, in which case, the I_P^π 'th $+1$ update may not be able to be scaled down in π' since the lower bound after the I_P^π 'th $+1$ update might be I_P^π . If this is the case, π' 1-scales all of the $+1$ updates and the last $I_N^\pi - 1$ many -1 updates. The remaining -1 updates are scaled to some small values in D (the dotted blue edge in Figure 6) except the last one, which is scaled to Δ (the brown edge in Figure 6). Thus, the sum is still k since all of the negative updates which are not 1-scaled add up to exactly -1 . The lower bounds in this case are satisfied since they were satisfied in π and all of the $+1$ updates are scaled to 1 and the negative updates are postponed as much as possible.

If this is not the case, that is the lower bound after the I_P^π 'th positive update is not I_P^π , it follows that none of the lower bounds along π are I_P^π . In π' , we 1-scale the first I_P^π many $+1$ updates and the last $I_N^\pi - 1$ many -1 updates, scale down the I_P^π 'th $+1$ update to a value in E (the solid black edge or the solid blue edge in Figure 6) in π' , and scale the -1 updates which were not 1-scaled to small values in D (the zigzag line and the green edge in Figure 6). The lower bounds are satisfied until the (only) $+\varepsilon$ since they were satisfied in π , all $+1$ updates are scaled to 1 and all negative updates are scaled down to small values in D . The lower bound immediately after the $+\varepsilon$ update is satisfied since the counter value at this point is between $I_P^\pi - 1$ and I_P^π . The lower bounds in the remainder are satisfied since π and π' both reach k , there are no positive updates and π' postpones the negative updates as much as possible.

For the remaining cases, we assume the number of positive and negative updates in π is strictly greater than I_P^π and I_N^π respectively. We also fix $\pi' = \pi''$ for these cases and hence omit the use of π'' .

- After the 1-scaling, the first non-zero update remaining is a $+1$ and the last one is a -1 (i.e., the first scaled down update in π' is a $+\varepsilon$ and the last one is a $-\delta$). Since the first nonzero update in the run apart from the 1-scaled ones is a $+1$ and this update must occur after all the 1-scaled $+1$ updates (because we scale the first I_P^π consecutive updates), this implies the absence of a negative update until the I_P^π 'th $+1$ update. The bounds in π' until this point are satisfied because they were satisfied in π , π' scales all the



+1 updates to 1 and the remaining updates are +0. The counter value at this point is I_P^π . The first +1 update after this sequence is scaled down to some $\varepsilon \in E$ (denoted by the first black edge in Figure 6). Similarly, since the last nonzero update apart from the 1-scaled ones is a 11 and this update must occur before all the 1-scaled +1 updates (because we scale the last I_N^π consecutive updates), this implies the absence of a negative update after the I_N^π 'th -1 update from the end. The last -1 update after this sequence is scaled down to some $\delta \in D$ (denoted by the green edge in Figure 6). The lower bounds until the first -1 update which was 1-scaled are satisfied since the counter values always stayed above I_P^π . The lower bounds in the final section of the string of -1 1-scaled updates are also satisfied since they were satisfied in π , π' delayed the -1 updates as much as possible and they both reach the same value k .

- After the 1-scaling, the first non-zero update remaining is a -1 and the last one is a +1 (i.e., the first scaled down update in π' is a $+\delta$ and the last one is a $-\varepsilon$). The lower bounds in the first section with the I_P^π many 1-scaled +1 updates (and possibly some $-\delta$ updates) are satisfied because they were satisfied in π and π' scales the +1 updates to 1 and the -1 updates to $-\delta$ whose sum never crosses -1 (as depicted by the dashed blue lines in Figure 6). The bounds are satisfied up to the first $+\varepsilon$ update by the same argument, even if there are more $-\delta$ updates after the I_P^π 'th 1-scaled +1 update. Similarly, the bounds after the last $-\delta$ update are satisfied since they are satisfied in π , both π and π' reach k , and all -1 updates are scaled up by 1 and all +1 updates are scaled down to $+\varepsilon$. Now, we only need to show that the bounds were satisfied from the first $+\varepsilon$ to the final $-\delta$. If there was exactly one $+\varepsilon$ (the solid red edge labelled ε in Figure 6), then the bounds are satisfied since it must occur after the last $-\delta$ update. Else, the first $+\varepsilon$ (the solid blue edge in Figure 6) update takes the counter value over I_P^π , it stays there until the final $-\delta$ and the counter values are satisfied.
- For the remaining 2 cases, namely, the first and last scaled down updates in π' are both in $+E$ or $-D$, the proof follows from a permutation of the arguments in the previous two cases.

Now that we have shown that the bounds are satisfied in π' , one can easily check that the rest of the properties are satisfied by π' by construction.

Next, if the second case holds, that is there exists a 1-scaled -1 update before a 1-scaled +1 update, π'' does not satisfy Item 4 of the normal form. The run π'' is of the form $(\{+1\} \cup -D)^* \cdot -1 \cdot \{+1, -1\}^* \cdot +1 \cdot (\{-1\} \cup +E)^*$. π'' does not necessarily satisfy the lower bounds, nor reach k , but it does have the property that $I_P^{\pi''} - I_N^{\pi''} = k$. We construct π' using π'' and show that the lower bounds hold sequentially. First, the bounds along the $(\{+1\} \cup -D)^*$ section of π'' are satisfied since they were satisfied in π , all +1 updates are 1-scaled and all -1 updates are scaled down to values in D . Similarly, the bounds in the $(\{-1\} \cup +E)^*$ of π'' are satisfied. These sections are the same for π' . We scale down the last 1-scaled +1 and the first 1-scaled -1 update in π'' to a $+\varepsilon$ and $-\delta$ respectively. Thus, these two updates are “absorbed” into the $(\{-1\} \cup +E)^*$ and $(\{+1\} \cup -D)^*$ sections of the run respectively. Call this new run π_1'' . This run also has the property that the +1 updates are concentrated on the left, the -1 updates are concentrated on the right, and $I_P^{\pi_1''} - I_N^{\pi_1''} = k$ (since $I_P^{\pi_1''} = I_P^{\pi''} - 1$ and $I_N^{\pi_1''} = I_N^{\pi''} - 1$). If π_1'' is of the form $(\{+1\} \cup -D)^* \cdot (-D \cup +E)^* \cdot (\{-1\} \cup +E)^*$, construct a k -reaching run in DNF as in case 1, but with π_1'' as input instead of π . Otherwise, π_1'' is also of the form $(\{+1\} \cup -D)^* \cdot -1 \cdot \{+1, -1\}^* \cdot +1 \cdot (\{-1\} \cup +E)^*$, but the size of the $\{+1, -1\}^*$ section of the run will be strictly smaller than that of π'' , and we make π_2'' by absorbing the first 1-scaled -1 and the last 1-scaled +1 update. Continue this until a run where the first 1-scaled -1 update occurs after the last 1-scaled +1 update. ◀

Nominal Tree Automata with Name Allocation

Simon Prucker  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

Data trees serve as an abstraction of structured data, such as XML documents. A number of specification formalisms for languages of data trees have been developed, many of them adhering to the paradigm of register automata, which is based on storing data values encountered on the tree in registers for subsequent comparison with further data values. Already on word languages, the expressiveness of such automata models typically increases with the power of control (e.g. deterministic, non-deterministic, alternating). Language inclusion is typically undecidable for non-deterministic or alternating models unless the number of registers is radically restricted, and even then often remains non-elementary. We present an automaton model for data trees that retains a reasonable level of expressiveness, in particular allows non-determinism and any number of registers, while admitting language inclusion checking in elementary complexity, in fact in parametrized exponential time. We phrase the description of our automaton model in the language of nominal sets, building on the recently introduced paradigm of explicit name allocation in nominal automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Data languages, tree automata, nominal automata, inclusion checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.35

Related Version *Full Version:* <https://arxiv.org/abs/2405.14272> [32]

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) -- Projektnummer 517924115.

1 Introduction

Letters from infinite alphabets generally serve as an abstraction of data values in formalisms for the specification and verification of structured data such as data words or data trees (e.g. [29]). They have variously been used to represent data values in XML documents [29], object identities [14], parameters of method calls [17], or nonces in cryptographic protocols [24]. There are, by now, quite a number of automata models for data languages, including register automata [20], data walking automata [27], and data automata [3]. A typical phenomenon in such models is that expressiveness increases strictly with the power of control (ranging from deterministic to alternating models). In such models, the key reasoning problem of inclusion checking tends to be either undecidable or computationally very hard unless stringent restrictions are imposed on either the power of control or on key parameters such as the number of registers. For instance, inclusion checking of nondeterministic register automata is undecidable unless one restricts either to unambiguous automata [28, 7] or to automata with at most two registers [20] (no complexity bound being given in the latter case); inclusion checking for alternating register automata is undecidable unless one restricts to only one register, and even then fails to be primitive recursive [11]; the inclusion problem of data walking automata is decidable but at least as hard as Petri net reachability [8], which by recent results is Ackermann-complete [25, 10, 26]; non-emptiness of data automata [3] is decidable but, again, at least as hard as Petri net reachability; and emptiness of variable automata [15] is undecidable unless one restricts to the (less expressive) deterministic fragment.



© Simon Prucker and Lutz Schröder;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 35; pp. 35:1–35:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Register-based automata models are often essentially equivalent to automata models in nominal sets [31]; for instance, register automata with nondeterministic reassignment [22] are equivalent to *nondeterministic orbit-finite automata* [4]. In the nominal setting, one way to ameliorate the mentioned degree of computational hardness while retaining a reasonable level of expressiveness is provided by the paradigm of explicit *name allocation* in nominal automata, which first appeared in *regular nondeterministic nominal automata (RNNA)* [34] and has subsequently been used in Büchi RNNA [40] and in a linear-time nominal μ -calculus [16]. In this paradigm, notions of freshness are based on α -equivalence, i.e. renaming of bound names as in λ -calculus, which blocks renaming of bound names into names that have free occurrences later in the word; in terms of the equivalent register-based setting, this amounts to a lossiness condition saying that at every point, register contents may nondeterministically be erased (thus freeing the register). At the same time, name-allocating models impose finite branching. Inclusion checking in these models is typically elementary, and in fact has low parametrized complexities with the parameter being the *degree*, which in the register-based setting corresponds to the number of registers.

Our present contribution is a nominal non-deterministic top-down tree automaton model that follows the name allocation paradigm. As our main result, we show that our model of *regular nominal tree automata (RNTA)* admits inclusion checking in doubly exponential time, more precisely in parametrized singly exponential time with the degree as the parameter (recall that the problem is already EXPTIME-complete for finite-alphabet non-deterministic top-down tree automata). We thus obtain an efficiently decidable formalism for the specification data words that admits full non-determinism and unboundedly many registers.

Omitted proofs can be found in the full version [32].

Related Work. We have already mentioned work on register word automata. Kaminski and Tan [21] introduce top-down and bottom-up register tree automata with or without non-deterministic reassignment; computational hardness of inclusion and universality is inherited from register word automata (while membership and emptiness are decidable in elementary complexity NP and EXPTIME, respectively [36]). Without non-deterministic reassignment, the top-down model and the bottom-up model have incomparable expressiveness. Our model of regular nominal tree automata (RNTA) relates, along the usual correspondence [4], to non-deterministic top-down register tree automata without non-deterministic reassignment. Van Heerdt et al. [41] treat bottom-up nominal tree automata with name allocation in a general algebraic setting, describing minimization and determinization constructions (without considering inclusion checking; determinization produces orbit-infinite automata). Since the finite-branching condition for the bottom-up variant differs from top-down finite branching as imposed in RNTAs, we expect similar incomparability as in the register-based setting; we leave the investigation of this point to future work. A register automata model for data trees with a different navigation pattern has been introduced by Jurdziński and Lazić [18, 19] and extended by Figueira [12]; in this model, the automaton moves downwards or rightwards on the tree instead of passing copies of itself to child nodes. The emptiness problem of the alternating model is decidable but not primitive recursive when the number of registers is restricted to 1; expressiveness is incomparable to Kaminski and Tan’s model [19]. One formalism for data trees that does allow inclusion checking in elementary complexity is the logic $FO^2(+1, \sim)$ [5], whose satisfiability problem is in 3NEXPTIME. Register tree automata have been extended to cover an ordering on the data values [37, 38].

2 Preliminaries

We assume basic familiarity with category theory (e.g. [1]). We briefly recall some background on nominal sets (see [31] for a textbook treatment), as well as on tree automata (e.g. [9]), register automata [20], and nominal automata [4].

Group Actions and Nominal sets. We fix a countably infinite set \mathbb{A} of *names*. Throughout, we let G denote the group of finite permutations on \mathbb{A} , which is generated by the permutations (ab) that swap two names $a, b \in \mathbb{A}$; we write id for the identity permutation, and $(-)\cdot(-)$ for the group operation, i.e. applicative-order composition of permutations. A G -set consists of a set X and a left action $(-)\cdot(-): G \times X \rightarrow X$ of G on X (subject to $id \cdot x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi \cdot \pi') \cdot x$). Given G -sets X, Y , a map $f: X \rightarrow Y$ is *equivariant* if $f(\pi \cdot x) = \pi \cdot f(x)$ for all x, y , and a subset $S \subseteq X$ is *equivariant* if $\pi \cdot s \in S$ for all $\pi \in G$ and $s \in S$, i.e. S is closed under the group action. The *orbit* of $x \in X$ is $G \cdot x = \{\pi \cdot x \mid \pi \in G\}$. The orbits form a disjoint decomposition of X ; the G -set X is *orbit-finite* if it has only finitely many orbits.

We write $\text{fix}(x) = \{\pi \in G \mid \pi(x) = x\}$ for $x \in X$, and $\text{Fix}(A) = \bigcap_{x \in A} \text{fix}(x)$ for $A \subseteq X$. The set \mathbb{A} itself is a G -set in a canonical manner. A set $S \subseteq \mathbb{A}$ is a *support* of an element $x \in X$ if

$$\text{Fix}(S) \subseteq \text{fix}(x),$$

that is, if every permutation that fixes all names in S also fixes x , which we understand as x depending only on the names in S .

Then, a G -set X is a *nominal set* if every element of X has a finite support. It follows that every $x \in X$ has a *least* finite support $\text{supp}(x)$, also just called the *support* of x . A name $a \in \mathbb{A}$ is *fresh* for x if $a \notin \text{supp}(x)$, in which case we write $a \# x$. We write Nom for the category of nominal sets and equivariant maps. Examples of nominal sets include \mathbb{A} itself (with $\text{supp}(a) = \{a\}$ for $a \in \mathbb{A}$); the product $X \times Y$ of nominal sets X, Y (with $\text{supp}(x, y) = \text{supp}(x) \cup \text{supp}(y)$); and the finitely supported powerset $\mathcal{P}_{\text{fs}}X$ of a nominal set X , which consists of the subsets of X that have finite support under the pointwise action of G on the full powerset. A set $A \subseteq X$ is *uniformly finitely supported* if $\bigcup_{x \in A} \text{supp}(x)$ is finite; we write $\mathcal{P}_{\text{ufs}}X = \{A \subseteq X \mid A \text{ uniformly finitely supported}\}$ for the *uniformly finitely supported powerset* of X . If X is orbit-finite, then the uniformly finitely supported subsets of X are precisely the finite subsets.

An important role in the technical development is played by the *abstraction functor* $[\mathbb{A}](-)$ on Nom . For a nominal set X , $[\mathbb{A}]X$ is defined as the quotient $\mathbb{A} \times X / \sim$ of $\mathbb{A} \times X$ by the equivalence relation \sim defined by

$$(a, x) \sim (b, y) \quad \text{iff} \quad (ca) \cdot x = (cb) \cdot y \quad \text{for } c \# (a, x, b, y).$$

Equivalently, $(a, x) \sim (b, y)$ iff either $(a, x) = (b, y)$ or $y = (ab) \cdot x$ and $b \# x$. We write $\langle a \rangle x$ for the equivalence class of (a, x) under \sim . Thus, $\langle a \rangle x$ may be read as arising from x by binding the name a . The equivalence \sim then captures α -equivalent renaming of the bound name a ; in particular, note that by the alternative description of \sim , renaming a into b in $\langle a \rangle x$ is *blocked* when $b \in \text{supp}(x)$.

Nominal automata and register automata. The classical notion of nondeterministic finite automaton can be transferred canonically to the category of nominal sets, where finiteness corresponds to orbit-finiteness; this gives rise to the notion of *nondeterministic orbite-finite automaton (NOFA)* [4]. For simplicity, we use the set \mathbb{A} of names as the alphabet (more

generally, one can work with any orbit-finite alphabet). Then, a NOFA consists of an orbit-finite set Q of states; an equivariant transition relation $\Delta \subseteq Q \times \mathbb{A} \times Q$; an equivariant initial state q_0 (or more generally a set of initial states); and an equivariant set $F \subseteq Q$ of final states. NOFAs accept finite words over \mathbb{A} , with the notions of run and acceptance defined exactly as in the classical case.

NOFAs are equivalent to a flavour of *nondeterministic register automata with nondeterministic reassignment*, roughly described as follows (see [4] for details). A register automaton has a finite set Q of control states; a fixed finite number of registers, which at any point in time can be either empty or contain a letter (i.e. a name from \mathbb{A}); an initial control state q_0 ; a set F of final control states; and a transition relation δ consisting of triples (q, ϕ, q') where $q, q' \in Q$ and where ϕ is a boolean combination of equality constraints concerning register contents before and after the transition and the current input letter. For simplicity, we require that all registers are initially empty. A *configuration* of the automaton consists of a control state and an assignment of letters to some of the registers (the others are empty). A *run* of the automaton is a sequence of configurations starting in the initial configuration (consisting of q_0 and all registers empty) such that every next configuration is justified by some transition (q, ϕ, q') in the sense that the boolean combination ϕ of equality constraints is satisfied by the register contents in the configurations before and after the transition and by the current input letter. A run is accepting if it ends in a configuration over a final control state. The language of the automaton is the set of accepted words. What this means is that the automaton can, in each transition step, perform any combination of the following actions as specified by the transition constraint ϕ : store the current letter in a register; copy content among registers; delete content from a register; nondeterministically store any name in a register (nondeterministic reassignment). These actions are conditioned on tests for equality or inequality among the registers and the input letter. In the correspondence between NOFAs and register automata, a configuration c of a register automaton becomes a state in the corresponding NOFA, whose support contains precisely the letters stored in the registers in c .

Both nondeterminism and nondeterministic reassignment strictly increase expressive power. For instance, the language L_2 “some letter occurs twice” can be accepted by a nondeterministic register automaton but not by a deterministic one; and the language “the last letter has not been seen before” can only be accepted using nondeterministic reassignment. Also, the nondeterministic model is not closed under complement: The complement of the above-mentioned language L_2 is the language “all letters are distinct”, which cannot be accepted by a nondeterministic register automaton.

Tree automata. Throughout, we fix a finite *ranked alphabet* (or *signature*) Σ , consisting of finitely many (*function*) *symbols*, each equipped with an assigned finite *arity*; we write $f/n \in \Sigma$ to indicate that f is a symbol of arity n in Σ . We assume that Σ contains at least one *constant*, i.e. a symbol of arity 0. The set $\mathcal{T}(\Sigma)$ of (ground) Σ -*Terms* is defined inductively by stipulating that whenever $f/n \in \Sigma$ and $t_1, \dots, t_n \in \mathcal{T}(\Sigma)$, then $f(t_1, \dots, t_n)$ in Σ . Terms are regarded as a representation of trees, with each node of the tree labelled with a symbol from Σ whose arity determines the number of child nodes.

A (classical, finite-alphabet) *nondeterministic top-down tree automaton (NFTA)* (e.g. [9]) over Σ is a tuple $A = (Q, q_0, \Delta)$ (we elide the fixed signature Σ) where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, and Δ is a set of *rewrite rules* or *transitions* of the form

$$q(f(x_1, \dots, x_n)) \rightarrow f(q_1(x_1), \dots, q_n(x_n))$$

where $f/n \in \Sigma$, $q, q_1, \dots, q_n \in Q$, and the x_i are variables; these rules thus manipulate *extended terms* containing automata states as unary symbols (with at most one occurrence of such a symbol per tree branch). Much as usual (e.g. [2, 9]), such rewrite rules are applied within a surrounding context, and with the variables x_i substituted with ground terms; we continue to write \rightarrow for the arising rewrite relation on extended terms. When $f/n \in \Sigma$ has arity $n > 1$, then a rewrite rule of the above shape transforms a single automaton state q into several automaton states q_1, \dots, q_n . One may view this phenomenon as the automaton creating copies of itself, which continue to operate independently on the children of the present node. The NFTA A *accepts* a term t if $q_0(t) \rightarrow^* t$. The *language* $L(A)$ is the set of terms (trees) accepted by A . NFTAs have the same expressiveness as *bottom-up tree automata*, in which the rewrite rules propagate automata states from the leaves to the root. Deterministic top-down tree automata, on the other hand, are strictly less expressive than NFTA. In the present work, our focus is on nondeterministic top-down models. The *inclusion problem* for NFTA, i.e. to decide whether $L(A) \subseteq L(B)$ for given NFTAs A, B , is EXPTIME-complete [35].

3 A Nominal View on Data Tree Languages

We proceed to introduce the relevant notion of nominal tree language, viewed as representing a data tree language. Trees are represented as a form of algebraic terms, and carry data values on their nodes. Throughout the technical development, we fix a finite algebraic *signature* Σ , i.e. a finite set of operation symbols f, g, \dots , each equipped with a finite *arity*. We write $f/n \in \Sigma$ to indicate that f is an operation symbol of arity n in Σ .

► **Definition 3.1** (Terms). We define the set $\mathcal{T}_{\mathbb{A}}(\Sigma)$ of *nominal Σ -terms*, or just *(Σ -)terms*, t by the grammar

$$t ::= a.f(t_1, \dots, t_n) \mid \nu a.f(t_1, \dots, t_n) \quad (t_1, \dots, t_n \in \mathcal{T}_{\mathbb{A}}(\Sigma), f/n \in \Sigma, a \in \mathbb{A}) \quad (3.1)$$

For uniformity of notation, we occasionally write $\bar{\mathbb{A}}$ for the set $\mathbb{A} \cup \{\nu a \mid a \in \mathbb{A}\}$; in particular, all terms then have the form $\gamma.f(t_1, \dots, t_n)$ with $\gamma \in \bar{\mathbb{A}}$.

► **Remark 3.2.** Like in the finite-alphabet case as recalled in Section 2, the base case of the above definition is the one where f is a constant (i.e. has arity 0). The case of words is recovered by taking Σ to consist of unary operations and an end-of-word constant. When viewing terms as trees, we see the operations of Σ as spanning the tree structure, with every node being labelled with an element of $\bar{\mathbb{A}}$. We understand terms of the form $a.f(t_1, \dots, t_n)$ as being labelled with a free name a , and terms of the form $\nu a.f(t_1, \dots, t_n)$ as allocating a new name a with scope $f(t_1, \dots, t_n)$. In the latter case, the name a is bound by the ν -operator, in the same style as in the π -calculus or in nominal Kleene algebra [13], with formal definitions to be provided presently. (In work on nominal word automata with name allocation [34], the notation $|a$ has been used in place of νa .)

► **Remark 3.3.** For brevity of presentation, we have opted for a setup where every node either binds a name or carries a free name. A generalization that allows nodes not labelled with any name is easily encoded into the present setup by means of a fixed free dummy name that appears in place of the absent name. We thus do use this generalization in the examples. In particular, it allows for trees not containing any name, while terms according to Definition 3.1 always contain at least one name.

The notion of free name informally used above is formally defined in the expected way:

► **Definition 3.4.** The set $\text{FN}(t)$ of *free names* of a term t is defined recursively by the clauses

$$\begin{aligned}\text{FN}(a.f(t_1, \dots, t_n)) &= \text{FN}(t_1) \cup \dots \cup \text{FN}(t_n) \cup \{a\} \\ \text{FN}(\nu a.f(t_1, \dots, t_n)) &= (\text{FN}(t_1) \cup \dots \cup \text{FN}(t_n)) \setminus \{a\}.\end{aligned}$$

Contrastingly, we refer to the name a in a term $\nu a.f(t_1, \dots, t_n)$ as a *bound name*. A term t is *closed* if $\text{FN}(t) = \emptyset$. The sets $\overline{\mathbb{A}}$ and $\mathcal{T}_{\mathbb{A}}(\Sigma)$ become nominal sets under the expected actions of G , defined on $\overline{\mathbb{A}}$ by $\pi \cdot a = \pi(a)$ and $\pi \cdot \nu a = \nu \pi(a)$, and on $\mathcal{T}_{\mathbb{A}}(\Sigma)$ recursively by

$$\pi \cdot (\delta.f(t_1, \dots, t_n)) = \pi(\delta).f(\pi \cdot t_1, \dots, \pi \cdot t_n).$$

From this view, we obtain the expected notion of α -equivalence of terms: α -*equivalence* \equiv_{α} is the least congruence on terms such that $\nu a.f(t_1, \dots, t_n) \equiv_{\alpha} \nu b.f(t'_1, \dots, t'_n)$ whenever $\langle a \rangle(t_1, \dots, t_n) = \langle b \rangle(t'_1, \dots, t'_n)$, which means that in $\nu a.f(t_1, \dots, t_n)$, a can be renamed into b provided that b does not occur in t_1, \dots, t_n (by temporary renaming of inner bound names, one can then also rename a into b if b is not free in t_1, \dots, t_n). We write $[t]_{\alpha}$ for the equivalence class of a term t under α -equivalence. A term t is *clean* if all its bound names are mutually distinct and not free in t , and *non-shadowing* if on every branch of t , all bound names are mutually distinct and not free in t (in particular, no bound name is ever shadowed in t). Clearly, every term is α -equivalent to a clean one (hence, a fortiori, to a non-shadowing one).

► **Example 3.5.** Under the extension where names are made optional (Remark 3.3), we can represent λ -terms as terms for the signature $\{\text{app}/2, \lambda/1, \text{var}/0\}$. For instance, the λ -term $\lambda a.aa$ is represented as the Σ -term $\nu a.\lambda(\text{app}(a.\text{var}, a.\text{var}))$. (Of course, there are Σ -terms not corresponding to any λ -term, such as $\nu a.\text{var}$; in our automaton model, it will be no problem to exclude such terms by just letting them get stuck.) Similarly, we can represent π -calculus expressions as terms for a suitable signature; we will return to this in Example 4.7. The notion of α -equivalence defined above is exactly the standard one in these examples.

As indicated in the introduction, the notion of α -equivalence determines how we interpret allocating a new name as “reading a name” in a paradigm where we see bound names as placeholders for arbitrary free names. For this purpose, we distinguish between *literal tree languages*, which consist of terms and hence are subsets of $\mathcal{T}_{\mathbb{A}}(\Sigma)$, and *alphatic tree languages*, which consist of α -equivalence classes of terms and hence are subsets of $\mathcal{T}_{\mathbb{A}}(\Sigma) / \equiv_{\alpha}$. (The latter generalize the *bar languages* used in work on nominal word automata [34] but in the absence of the bar notation la , that term seems no longer appropriate.) In both cases, we say that a language is *closed* if it consists of closed (equivalence classes of) words only. For brevity, we restrict the main line of the technical treatment to closed languages. The notion of bound names representing free names is captured by the function $d\nu$, which removes all occurrences of ν from a term, and is recursively defined by

$$d\nu(\nu a.f(t_1, \dots, t_n)) = d\nu(a.f(t_1, \dots, t_n)) = a.f(d\nu(t_1), \dots, d\nu(t_n)).$$

Thus, $d\nu$ returns terms without name allocation νa . We refer to such terms as *data trees*; they correspond essentially to the notion of data tree found in the literature (which is often restricted to binary trees for simplicity, e.g. [21, 18]), with \mathbb{A} serving as the infinite alphabet of data values. We capture notions of freshness by imposing different disciplines on variable management in α -renaming and subsequently applying $d\nu$. Specifically, a *global freshness semantics*, a *branchwise freshness semantics*, and a *local freshness semantics* for alphatic tree languages L are embodied in the operators \mathbf{N} , \mathbf{B} , and \mathbf{D} , respectively, defined by

$$\begin{aligned} \mathbf{N}(L) &= \{d\nu(t) \mid t \text{ clean}, [t]_\alpha \in L\} & \mathbf{B}(L) &= \{d\nu(t) \mid t \text{ non-shadowing}, [t]_\alpha \in L\} \\ \mathbf{D}(L) &= \{d\nu(t) \mid [t]_\alpha \in L\}. \end{aligned}$$

That is, N and B insist on clean or non-shadowing α -renaming, respectively, while D allows unrestricted α -renaming. The languages $\mathbf{N}(L)$, $\mathbf{B}(L)$, and $\mathbf{D}(L)$ are what we term *data tree languages*, i.e. languages consisting only of data trees. This makes one additional semantics in comparison to the case of words [34] where, of course, N and B coincide.

► **Example 3.6.** Consider $\Sigma = \{f/2, k/0\}$ and the term

$$t = \nu a.f(\nu b.f(a.k, b.k), \nu b.f(b.k, b.k)).$$

Then

$$\begin{aligned} \mathbf{N}(\{t\}) &= \{a.f(b.f(a.k, b.k), c.f(c.k, c.k)) \mid a, b, c \in \mathbb{A}, a \neq b, a \neq c, b \neq c\} \\ \mathbf{B}(\{t\}) &= \{a.f(b.f(a.k, b.k), c.f(c.k, c.k)) \mid a, b, c \in \mathbb{A}, a \neq b, a \neq c\} \\ \mathbf{D}(\{t\}) &= \{a.f(b.f(a.k, b.k), c.f(c.k, c.k)) \mid a, b, c \in \mathbb{A}, a \neq b\}. \end{aligned}$$

Thus, N instantiates bound names by free names that are fresh w.r.t. the entire tree, while B only enforces freshness w.r.t. the current branch, and allows siblings to instantiate bound names by the same free name (i.e. allows $b = c$ in $\mathbf{B}(\{t\})$). Finally, D , enforces freshness only where α -renaming is blocked. In the case of t , renaming b into a is blocked in the left-hand subterm $\nu b.f(a.k, b.k)$ because of the free occurrence of a , while there is no such occurrence in the right-hand subterm $\nu b.f(b.k, b.k)$ so that b can be renamed into a in that subterm; this is why $\mathbf{D}(\{t\})$ requires $a \neq b$ but not $a \neq c$. Thus, N and B are variants of global freshness as found, for instance, in session automata [6], while D is indeed a notion of local freshness: We will see later (Lemma 4.2) that the presence of the free name a in $\nu b.f(a.k, b.k)$ implies that at the time of reading νb in a run of a regular nominal tree automaton, the name a is still kept in memory, so D essentially enforces freshness w.r.t. currently stored names in the spirit of register automata models. As indicated in the introduction, we trade some of the expressiveness of register automata models for computational tractability. In our example, this is apparent in the right-hand subterm $\nu b.f(b.k, b.k)$, in which freshness of b w.r.t. a cannot be enforced under D because there is no free occurrence of a ; as a slogan, D enforces freshness w.r.t. stored names only if these are still expected to be seen again. In work on nominal word automata [34], it is shown that this phenomenon relates to a lossiness property stating that register contents may be non-deterministically lost at any time.

It turns out that both variants of global freshness remain essentially equivalent to the original alphabetic language, in the sense that they do not affect language inclusion:

► **Lemma 3.7.** *Both N and B preserve and reflect language inclusion: For alphabetic languages L_1, L_2 , we have $L_1 \subseteq L_2$ iff $\mathbf{N}(L_1) \subseteq \mathbf{N}(L_2)$ iff $\mathbf{B}(L_1) \subseteq \mathbf{B}(L_2)$.*

That is, for purposes of checking language inclusion, it does not matter whether we consider alphabetic languages, their global freshness semantics, or their branchwise freshness semantics.

4 Regular Nominal Tree Automata

We cast our model of *regular nominal tree automata* (RNTAs) as an extension of *regular nondeterministic nominal automata* (RNNAs) [34], which differ from NOFAs (Section 2) in essentially two ways: Branching is restricted to be finite (in NOFAs, the set of successors of a state only needs to be finitely supported, as implied by equivariance of the transition relation); this is partially made up for by including *bound transitions* which read bound names. RNTAs natively accept alphabetic tree languages, which as discussed in Section 3 may be seen as representing languages of data trees in a number of ways differing w.r.t. notions of freshness: Under global or branchwise freshness semantics, RNTAs may be seen as a generalization of session automata [6], while under local freshness semantics, they will be seen to correspond to a subclass of nondeterministic register tree automata [21] characterized by a lossiness condition (Remark 5.6). As indicated in the introduction and in Example 3.6, we thus incur a decrease in expressiveness in comparison to the register model, which however buys elementary complexity of inclusion checking.

► **Definition 4.1** (Regular nominal tree automata). A *regular nominal tree automaton* (RNTA) over our fixed signature Σ is a tuple

$$A = (Q, \Delta, q_0)$$

where Q is a orbit-finite nominal set of *states*, $q_0 \in Q$ is the *initial state* and Δ is an equivariant set of *rewrite rules* or *transitions* of the form

$$q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))$$

where $q_1, \dots, q_n \in Q$, $\gamma \in \overline{\mathbb{A}}$, $f/n \in \Sigma$, and the x_i are variables. When no confusion is likely, we just write $q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))$ to indicate that $(q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))) \in \Delta$. We impose two properties on Δ :

■ *α -invariance*: For $q, q_1, \dots, q_n, q'_1, \dots, q'_n \in Q$, if $\langle a \rangle(q_1, \dots, q_n) = \langle b \rangle(q'_1, \dots, q'_n)$, then

$$\begin{aligned} q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n)) & \text{ implies} \\ q(\nu b.f(x_1, \dots, x_n)) \rightarrow \nu b.f(q'_1(x_1), \dots, q'_n(x_n)). \end{aligned}$$

■ *Finite branching up to α -equivalence*: For all $q \in Q$ and $f/n \in \Sigma$, the sets $\{(a, (q_1, \dots, q_n)) \mid q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))\}$ and $\{\langle a \rangle(q_1, \dots, q_n) \mid q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n))\}$ are finite.

Like in the classical case (Section 2), the rewrite rules in Δ may be applied within a surrounding context and with variables substituted by (ground) terms. A state $q \in Q$ *accepts* a term t if there exists a sequence of applications of the rewrite rules in Δ , called a *run*, that starts from $q(t)$ and ends in t , symbolized as $q(t) \xrightarrow{*} t$. We define the *literal tree language* $L(q)$ and the *alphabetic tree language* $L_\alpha(q)$ accepted by q by

$$L(q) = \{t \in \mathcal{T}_{\mathbb{A}}(\Sigma) \mid q \text{ accepts } t\} \quad L_\alpha(q) = \{[t]_\alpha \mid t \in L(q)\}$$

We put $L(A) = L(q_0)$ and $L_\alpha(A) = L_\alpha(q_0)$, i.e. the RNTA A *accepts* t if its initial state accepts t . Moreover, A *accepts* a data tree s *under global, branchwise, or local freshness semantics* if $s \in \mathbf{N}(L_\alpha(A))$, $s \in \mathbf{B}(L_\alpha(A))$, or $s \in \mathbf{D}(L_\alpha(A))$, respectively (cf. Section 3).

The *degree* of A is the maximal cardinality of supports of states in A .

We think of the support of an RNTA state as consisting of finitely many stored names. In examples, we typically write states in the form

$$q(a_1, \dots, a_n)$$

where q indicates the orbit and a_1, \dots, a_n are stored names. Thus, the degree of an RNTA corresponds morally to the number of registers. As an important consequence of finite branching, stored names can come about only by either inheriting them from a predecessor state or by reading (i.e. binding) them:

► **Lemma 4.2.** *In an RNTA, we have the following properties:*

1. *If $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))$, then $\text{supp}(q_i) \cup \{a\} \subseteq \text{supp}(q)$ for $i = 1, \dots, n$.*
2. *If $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n))$, then $\text{supp}(q_i) \subseteq \text{supp}(q) \cup \{a\}$ for $i = 1, \dots, n$.*

► **Corollary 4.3.** *If state q accepts term t , then $\text{FN}(t) \subseteq \text{supp}(q)$.*

► **Remark 4.4.** For brevity, we restrict the further technical treatment to the case where *the initial state has empty support*, which by Corollary 4.3 implies that *the accepted language is closed*. We also assume this without further mention in the examples, with the possible exception of the dummy name needed in examples with unlabelled nodes (cf. Remark 3.3), in which the dummy name is assumed to be in the support of all states.

► **Example 4.5.** Let $\Sigma = \{f/2, k/0\}$ (so Σ -terms are just $\bar{\mathbb{A}}$ -labelled binary trees).

1. The universal data tree language, i.e. the language consisting of all (non-empty, cf. Remark 3.3) data trees, is accepted under local freshness semantics by the RNTA with only one state q and transitions $q(\nu a.f(x, y)) \rightarrow \nu a.f(q(x), q(y))$, $q(\nu a.k) \rightarrow \nu a.k$. On the other hand, it is easy to see that the universal data tree language cannot be accepted by an RNTA under global or branchwise freshness semantics. Under the latter semantics, the above RNTA accepts the language of all data trees in which all names are distinct or in which all names found on the same branch are distinct, respectively.
2. The data tree language “the letter at the root of the tree (which moreover is not a leaf) reappears in all leaves, but not in any other node of the tree” is accepted under local freshness semantics by the RNTA with states $q_0, q_1(a)$ ($a \in \mathbb{A}$) and transitions

$$\begin{aligned} q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_1(a)(x), q_1(a)(y)) \\ q_1(a)(\nu b.f(x, y)) &\rightarrow \nu b.f(q_1(a)(x), q_1(a)(y)) \quad q_1(a)(a.k) \rightarrow a.k \end{aligned}$$

where we mean this and all further examples to be implicitly closed under equivariance and α -invariance. (Regarding notation, read $q_1(a)(y)$ as “state $q_1(a)$ processing term y ”.)

3. The data tree language “there is some letter that appears twice on the same branch” (which, in analogy to the word case [4, 34], cannot be accepted by a deterministic register tree automaton) is accepted under local freshness semantics by the RNTA with states $q_0, q_1(a), q_2$ ($a \in \mathbb{A}$), transitions

$$\begin{aligned} q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_0(x), q_2(y)) & q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_2(x), q_0(y)) \\ q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_1(a)(x), q_2(y)) & q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_2(x), q_1(a)(y)) \\ q_1(a)(\nu b.f(x, y)) &\rightarrow \nu b.f(q_1(a)(x), q_2(y)) & q_1(a)(\nu b.f(x, y)) &\rightarrow \nu b.f(q_2(x), q_1(a)(y)) \\ q_1(a)(a.f(x, y)) &\rightarrow a.f(q_2(x), q_2(y)), \end{aligned}$$

and transitions ensuring that q_2 accepts the universal data tree language as in item 1. It is easy to see that the complement of this language, while acceptable under branchwise freshness semantics as seen in item 1., cannot be accepted by an RNTA under local freshness semantics.

35:10 Nominal Tree Automata with Name Allocation

► **Example 4.6** (Structured data). We use $\Sigma = \{\text{!elem}/2, \text{\#data}/1, \text{eof}/0\}$ to support an XML-like syntax for structured data. We want to recognize the language of Σ -trees where every occurrence of **!elem** is properly closed by **eof** in the subtree below it, at a leaf as far to the left in the tree as possible under the policy that later occurrences of **!elem** are closed further to the left. Occurrences of **eof** and **!elem** are matched by binding a name at **!elem** and labelling the corresponding **eof** with this name (moreover, one unlabelled **eof** closes the entire term), as in the term

```

νa.!elem(
  νb.#data(
    νc.!elem(
      νd.#data(
        νd.#data(
          νd.#data(
            c.eof))),
      νb.#data(
        a.eof),
    eof)))

```

Under local freshness semantics, the data elements b in the **#data** fields of this term can be any names except a , similarly for d and c . This language is accepted by the RNTA with states $q_0, q_1(a), q_1(c)$ ($a, c \in \mathbb{A}$) and transitions

$$\begin{aligned}
q_0(\nu a.!\text{elem}(x_1, x_2)) &\rightarrow \nu a.!\text{elem}(q_1(a)(x_1), q_0(x_2)) \\
q_0(\nu b.\#\text{data}(x_1)) &\rightarrow \nu b.\#\text{data}(q_0(x_1)) & q_0(\text{eof}) &\rightarrow \text{eof} \\
q_1(a)(\nu c.!\text{elem}(x_1, x_2)) &\rightarrow \nu c.!\text{elem}(q_1(c)(x_1), q_1(a)(x_2)) \\
q_1(a)(\nu d.\#\text{data}(x_1)) &\rightarrow \nu d.\#\text{data}(q_1(a)(x_1)) & q_1(a)(a.\text{eof}) &\rightarrow a.\text{eof}.
\end{aligned}$$

Notice here that although every state stores at most one name, the automaton is able to track an unbounded number of **!elem** markers as it effectively creates copies of itself when reading an input tree.

► **Example 4.7** (π -Calculus expressions). We use $\Sigma = \{\text{par}/2, \text{rw}/1, \text{ch}/1, 0/0\}$, $\mathbb{A} = \{\perp, a, b, c, \dots\}$ to represent the syntax (only!) of a small fragment of the π -calculus, with *par* standing for parallel composition, and with *ch* and *rw* working in combination to represent writing or reading on a channel. Here, we model reading and allocation of channel names natively as name binding: $a.\text{ch}$ communicates on an existing channel a , and $\nu a.\text{ch}$ on a newly allocated channel a . Similarly, $a.\text{rw}$ writes a , while $\nu a.\text{rw}$ reads a value a . E.g., $\nu a.\text{ch}(\nu b.\text{rw}(x))$ reads b from a newly allocated channel a , and continues with x . Let L be the language of all Σ -terms that are parallel composites of $k \geq 1$ processes that each read a name b from a newly allocated channel a and then may, maybe repeatedly, read a new name from b and use that name as the input channel in the next round, before terminating (0). This language is accepted by the RNTA with states $q_0, q_1, q_2(a)$ ($a \in \mathbb{A}$) and transitions

$$\begin{aligned}
q_0(\text{par}(x, y)) &\rightarrow \text{par}(q_0(x), q_0(y)) & q_0(\nu a.\text{ch}(x)) &\rightarrow \nu a.\text{ch}(q_1(x)) \\
q_1(\nu a.\text{rw}(x)) &\rightarrow \nu a.\text{rw}(q_2(a)(x)) & q_2(a)(a.\text{ch}(x)) &\rightarrow a.\text{ch}(q_1(x)) \\
q_2(a)(0) &\rightarrow 0.
\end{aligned}$$

(Notice that the right hand transitions forget the channel name once the channel command has been processed; the name is no longer needed, as every channel is used only once.)

► **Remark 4.8.** In the paradigm of universal coalgebra [33], RNTAs may be viewed as coalgebras for the functor F given by

$$FX = \mathcal{P}_{\text{ufs}}(\sum_{f/n \in \Sigma} (\mathbb{A} \times X^n + [\mathbb{A}]X^n)). \quad (4.1)$$

5 Name Dropping

The key to the algorithmic tractability of name-allocating automata models in general [34, 40, 16] is to ensure that the literal language of an automaton is closed under α -equivalence, so that only boundedly many names need to be considered in inclusion checking. The problem to be overcome here is that this property does not hold in general, and needs to be enforced in a modification of the automaton that preserves the alphabetic language. Specifically, the problem comes about by extraneous names that do not occur in the remainder of a given word to be processed but do still occur in the relevant successor state, thus blocking the requisite α -renaming. As a simple example, when an automaton state q is processing $\nu a.f(t)$ for $f/1 \in \Sigma$ and we have a matching transition $q(\nu a.f(x)) \rightarrow \nu a.f(q'(x))$, then it may happen that $b \notin \text{FN}(t)$, so that a may be α -equivalently renamed into b in $\nu a.f(t)$, but $a \in \text{supp}(q')$ so that a cannot be α -equivalently renamed into b in $\nu a.f(q'(x))$. The solution to this is to extend the automaton by states that come about by dropping some of the names from the support of previous states [34]; in the example, a state q'' that has b removed from its support but otherwise behaves like q' will allow for the requisite α -renaming of the transition into $\nu a.f(q''(x))$, and will still be able to accept the remaining term t since $b \notin \text{FN}(t)$. We proceed to lay out the details of this construction, which we dub the *name-dropping modification*.

Following work on nominal Büchi automata [40], we first transform the automaton into one whose state set Q forms a *strong* nominal set; we do not need the original definition of strong nominal set [39] but instead use the equivalent description [30] of strong nominal sets as being those of the form $\sum_{i \in I} \mathbb{A}^{\#X_i}$ where the X_i are finite sets and $\mathbb{A}^{\#X_i}$ denotes the nominal set of total injective maps $X_i \rightarrow \mathbb{A}$. We generally write elements of sums like the above as pairs (i, r) , in this case consisting of $i \in I$ and $r \in \mathbb{A}^{\#X_i}$. Strong nominal sets thus materialize the intuition that the states of a nominal automaton consist of a control state (the index i in the above sum) and a store configuration assigning names to registers in a duplicate-free manner.

► **Lemma 5.1.** *For every RNTA A , there exists an RNTA A' whose states form a strong nominal set such that A and A' accept the same literal language.*

Our name-dropping modification will now come about by dropping the requirement that each register is necessarily occupied. This amounts to working with *partial* injective maps $r: X_i \rightarrow \mathbb{A}$, with undefinedness (denoted as $r(x) = \perp$) indicating that a register is currently empty. We first introduce notation for restricting such partial maps by dropping some of the names:

► **Definition 5.2.** Let X be a finite set. We write $\mathbb{A}^{\text{s}X}$ for the set of partial injective maps $X \rightarrow \mathbb{A}$. Let $r \in \mathbb{A}^{\text{s}X}$, and let $N \subseteq \text{supp}(r) = r[X]$. Then the partial injective map $r|_N \in \mathbb{A}^{\text{s}X}$ defined by $r|_N(x) = r(x)$ if $r(x) \in N$ and $r|_N(x) = \perp$ otherwise is the *restriction* of r to N (and r is an *extension* of $r|_N$).

► **Definition 5.3 (Name-dropping modification).** Let $A = (Q, \Delta, q_0)$ be an RNTA such that $Q = \sum_{i \in I} \mathbb{A}^{\#X_i}$ is a strong nominal set. For $q = (i, r) \in Q$, we write $q|_N = (i, r|_N)$. Then the *name-dropping modification* of A is the RNTA $A_{\perp} = (Q_{\perp}, \Delta_{\perp}, q_0)$ where

1. $Q_{\perp} = \sum_{i \in I} \mathbb{A}^{\otimes X_i}$;
2. for all $q, q'_1, \dots, q'_n \in Q$, $N \subseteq \text{supp}(q)$, $N_i \subseteq \text{supp}(q'_i) \cap N$ ($i = 1, \dots, n$), and $a \in N$, whenever $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q'_1(x_1), \dots, q'_n(x_n))$ in A , then $q|_N(a.f(x_1, \dots, x_n)) \rightarrow a.f(q'_1|_{N_1}(x_1), \dots, q'_n|_{N_n}(x_n))$ in A_{\perp} ; and
3. for all $q, q'_1, \dots, q'_n \in Q$, $N \subseteq \text{supp}(q)$, $a \in \mathbb{A}$, and $N_i \subseteq \text{supp}(q'_i) \cap (N \cup \{a\})$ ($i = 1, \dots, n$), whenever $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q'_1(x_1), \dots, q'_n(x_n))$ in A and $\langle a \rangle q'_i|_{N_i} = \langle b \rangle q''_i$, $i = 1, \dots, n$, then $q|_N(\nu b.f(x_1, \dots, x_n)) \rightarrow \nu b.f(q''_1(x_1), \dots, q''_n(x_n))$ in A_{\perp} .

Notice that clauses defining the transition relation on A_{\perp} are only implications: A_{\perp} inherits transitions from A as long as these are consistent with Corollary 4.3, and bound transitions in A_{\perp} are subsequently closed under α -invariance. The arising transitions are characterized as follows.

► **Lemma 5.4.** *Let $A = (Q, \Delta, i)$ be an RNTA with Q strong, and let $A_{\perp} = (Q_{\perp}, \Delta_{\perp}, i)$ be its name-dropping modification.*

1. *If $q|_N(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))$ in A_{\perp} for some $q, q_1, \dots, q_n \in Q$ and $N \subseteq \text{supp}(q)$, then each q_i has the form $q_i = q'_i|_{N_i}$ for some $q'_i \in Q$, $N_i \subseteq \text{supp}(q) \cap N$ such that $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q'_1(x_1), \dots, q'_n(x_n))$ in A .*
2. *If $q|_N(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n))$ in A_{\perp} for some $q, q_1, \dots, q_n \in Q$ and $N \subseteq \text{supp}(q)$, then for each q_i there is q'_i and $N_i \subseteq \text{supp}(q'_i) \cap (N \cup \{b\})$ such that $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu b.f(q'_1(x_1), \dots, q'_n(x_n))$ in A and $\langle b \rangle q'_i|_{N_i} = \langle a \rangle q_i$.*

In both claims, the given conditions are, by definition, also sufficient; the key point of the lemma is that all transitions of a given state $q|_N$ come from transitions of q even when one also has $q|_N = q'|_N$ for a different q' .

The degree of the name-dropping modification remains the same because the new states arise by deleting names from the support of previous states; the number of orbits increases only by a factor 2^d , where d is the degree, because there are only 2^d ways to delete names from a support of size d . Moreover, one can show that as per the intention of the construction, the name dropping modification closes an RNTA under α -equivalence; summing up:

► **Theorem 5.5.** *For each RNTA A , the name-dropping modification A_{\perp} of A is an RNTA that accepts the closure of the literal tree language of A under α -equivalence, and hence the same alphabetic tree language as A . Moreover, A_{\perp} has the same degree d as A , and the number of orbits of A_{\perp} exceeds that of A by at most a factor 2^d .*

► **Remark 5.6 (Lossiness).** It is apparent from the construction of the name-dropping modification that, in the usual correspondence between nominal automata models and register-based models [4, 34], it establishes a lossiness property saying that during any transition, letters may nondeterministically be lost from the registers. Intuitively speaking, the effect of losing a letter from a register is on the one hand that one escapes freshness requirements against that letter in successor states, but on the other hand progress may later be blocked when the lost name is required to be seen in the word; the overall consequence of this phenomenon is that distinctness of the current letter b from a letter a seen previously in the word can only be enforced if a is expected to be seen again, as already illustrated in Examples 3.6 and 4.5.

6 Inclusion Checking

We conclude by showing that language inclusion of RNTAs is decidable in elementary complexity, in sharp contrast to the typical situation in register-based models as discussed in Section 1. The algorithm is based on reducing the problem to language inclusion of classical NFTAs over finite signatures (Section 2), using the name-dropping modification

to ensure closure of literal tree languages under α -equivalence (Section 5): Using closure under α -equivalence, we can restrict to a finite set of names that is large enough to represent every relevant α -equivalence class. Using this set of names, we cut out a finite part of the RNTA in which only the specified names appear. For these restricted automata, which are just NFTAs, we can decide language inclusion in EXPTIME. A key step in this programme is thus the following:

► **Definition 6.1.** *Given a finite set $S \subseteq \mathbb{A}$, we write $\mathcal{T}_S(\Sigma) = \{t \in \mathcal{T}_{\mathbb{A}}(\Sigma) \mid \text{supp}(t) \subseteq S\}$.*

(That is, a term is in $\mathcal{T}_S(\Sigma)$ if all its free and bound names are in S .)

► **Lemma 6.2.** *Let A be an RNTA of degree d_A , and let n_{ar} be the maximal arity of symbols in Σ . Pick $S \subseteq \mathbb{A}$ such that $|S| = d_A \cdot n_{ar} + 1$. If A accepts a term t , then A also accepts some term $t' \in \mathcal{T}_S(\Sigma)$ such that $t' \equiv_{\alpha} t$.*

(Recall that initial states have empty support by our running assumption; otherwise, S would also need to contain the support of the initial state.)

► **Theorem 6.3.** *Alphabetic tree language inclusion $L_{\alpha}(A) \subseteq L_{\alpha}(B)$ of RNTAs A, B of degrees d_A, d_B , respectively, over the fixed signature Σ is decidable in doubly exponential time, and in fact in parametrized singly exponential time with the degree as the parameter, i.e. exponential in a function that depends exponentially on $d_A + d_B$ and polynomially on the size of A, B .*

Here, we understand the size of A and B in terms of standard finitary representations of orbit-finite nominal sets, which essentially just enumerate the support sizes and symmetry groups of the orbits (e.g. [40]). In the complexity analysis, we assume the signature to be fixed; if the signature is made part of the input, then the parameter includes also the maximal arity n_{ar} of symbols in Σ as in Lemma 6.2.

Proof. The proof uses reduction to a finite-alphabet [34, 40]. Again, let n_{ar} be the maximal arity of symbols in Σ , and pick $S \subseteq \mathbb{A}$ such that $|S| = d_A \cdot n_{ar} + 1$ as required in Lemma 6.2. Put $\bar{S} = S \cup \{\nu a \mid a \in S\}$, and let B_{\perp} be the name-dropping modification of B as per Theorem 5.5. Let $(Q_A, \Delta_A, q_0^A) = A$ and $(Q_B, \Delta_B, q_0^B) = B$.

1. Show that $L_{\alpha}(A) \subseteq L_{\alpha}(B)$ iff $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$.
 - “ \Rightarrow ”: By Theorem 5.5, $L(B_{\perp})$ is the closure of $L(B)$ under α -equivalence. Thus, $L(A) \subseteq L(B_{\perp})$, and hence $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$.
 - “ \Leftarrow ”: Let $[t]_{\alpha} \in L_{\alpha}(A)$; we have to show that $[t]_{\alpha} \in L_{\alpha}(B)$. By definition of $L_{\alpha}(A)$, we have $t' \in L(A)$ such that $t' \equiv_{\alpha} t$, so by Lemma 6.2 there exists $t'' \in L(A) \cap \mathcal{T}_S(\Sigma)$ such that $t'' \equiv_{\alpha} t$. Then $t'' \in L(B_{\perp})$ by hypothesis, and hence $[t]_{\alpha} \in L_{\alpha}(B_{\perp})$. By Theorem 5.5, we obtain $[t]_{\alpha} \in L_{\alpha}(B)$ as required.
2. By 1, we are left to decide whether $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$. Observe that $L(A) \cap \mathcal{T}_S(\Sigma)$ and $L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$ are effectively just tree languages over the finite signature $\bar{S} \times \Sigma$. We construct top-down NFTAs A_S and B_S over $\bar{S} \times \Sigma$ that restrict A and B_{\perp} , respectively, to S and accept $L(A) \cap \mathcal{T}_S(\Sigma)$ and $L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$, respectively: Put $A_S = (Q_{A,S}, \Delta_{A,S}, q_0^A)$ where $Q_{A,S} = \{q \in Q_A \mid \text{supp}(q) \subseteq S\}$ and $\Delta_{A,S} = \{(q(\gamma.f(x_1, \dots, x_n) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))) \in \Delta_A \mid q, q_1, \dots, q_n \in Q_{A,S}, \gamma \in \bar{S})$. The construction of $B_S = (Q_{B_{\perp},S}, \Sigma \times \bar{S}, \Delta_{B_{\perp},S}, i_{B_{\perp}})$ is analogous. The automata A_S and B_S are finite because A and B_{\perp} are orbit-finite and each orbit of a nominal set contains only finitely many elements with a given finite support.

We verify that A_S accepts $L(A) \cap \mathcal{T}_S(\Sigma)$, i.e. $L(A_S) = L(A) \cap \mathcal{T}_S(\Sigma)$; the corresponding claim for B_{\perp} is analogous. Since all states and rewrite rules of A_S are inherited from A , it is immediate that $L(A_S) \subseteq L(A) \cap \mathcal{T}_S(\Sigma)$; we show the reverse inclusion. So let

$t \in L(A) \cap \mathcal{T}_S(\Sigma)$; we have to show that A_S accepts t . We show more generally that every state q of A_S accepts all terms $t \in \mathcal{T}_S(\Sigma)$ that q accepts in A , and proceed via induction on the length of an accepting run.

For the base case, let q accept $t = \gamma.c$ in A , where $\gamma \in \bar{S}$ because $t \in \mathcal{T}_S(\Sigma)$. That is, we have $\delta = (q(\gamma.c) \rightarrow \gamma.c) \in \Delta_A$. The $\delta \in \Delta_{A,S}$ by construction, so q accepts t in A_S .

For the inductive step, let q accept $t = \gamma.f(t_1, \dots, t_n)$ in A . Again, $\gamma \in \bar{S}$ because $t \in \mathcal{T}_S(\Sigma)$. Thus, we have $\delta = (q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))) \in \Delta_A$ such that q_i accepts t_i for $i = 1, \dots, n$. We distinguish between bound and free transitions:

- $\gamma = a$: By Lemma 4.2, $\text{supp}(q_i) \subseteq \text{supp}(q) \subseteq S$, so $q_i \in Q_S$, and by induction, q_i accepts t_i in A_S for $i = 1, \dots, n$. Since, $\gamma \in S$, we thus have $\delta \in \Delta_{A,S}$; it follows that q accepts t in A_S .
 - $\gamma = \nu a$: By Lemma 4.2, $\text{supp}(q_i) \subseteq \text{supp}(q) \cup \{a\}$. Since $a \in S$ and $\text{supp}(q) \subseteq S$, this implies $\text{supp}(q_i) \subseteq S$, i.e. $q_i \in Q_S$ for $i = 1, \dots, n$. By induction, q_i accepts t_i in A_S , and again, $\delta \in \Delta_{A,S}$ by construction because $\gamma \in S$, implying that q accepts t in A_S .
3. So far, we have reduced the problem to deciding language inclusion of NFTAa, which is in EXPTIME [9]; it remains to analyse the size of the NFTAs A_S, B_S constructed in step 2., where we have first constructed the name-dropping modification B_\perp of the RNTA B and have then extracted A_S and B_S from A and B_\perp , respectively, by restricting to the finite set S of names. We assume for simplicity that the state spaces of A and B are given as strong nominal sets, so that the size of A and B is essentially the respective number of orbits. When estimating the size of A_S and B_S , it suffices to consider the number of states, since the size of the signature $\bar{S} \times \Sigma$ is linear in d_A (as Σ is assumed to be fixed) so that the number of transitions of NFTAs over $\bar{S} \times \Sigma$ is polynomial in their number of states and d_A . It thus suffices to show that the number of states in A_S and B_S , respectively, is the number of orbits of A or B , respectively, multiplied by a factor that is singly exponential in the degree. Now by Theorem 5.5, the name-dropping modification step for B increases the number of orbits by an exponential factor in the degree d_B but leaves the degree itself unchanged. Moreover, we generally have that every orbit of a given nominal set with support size m has at most $m!$ elements with a given fixed support, so the step from A, B to A_S, B_S indeed incurs only an exponential factor in the degree, which proves the claim. \blacktriangleleft

From Lemma 3.7, it is immediate that the same complexity bound as in Theorem 6.3 holds also for inclusion checking of RNTAs under global and branchwise freshness semantics, respectively (i.e. for checking whether $\mathbf{N}(L_\alpha(A)) \subseteq \mathbf{N}(L_\alpha(B))$ or $\mathbf{B}(L_\alpha(A)) \subseteq \mathbf{B}(L_\alpha(B))$, respectively). We conclude by showing that this remains true under local freshness semantics. The following observation is key:

► **Definition 6.4.** We define an ordering \leq on $\bar{\mathbb{A}}$ by $a \leq \nu a$ for all $a \in \mathbb{A}$. We then define the ordering \sqsubseteq on $\mathcal{T}_{\bar{\mathbb{A}}}(\Sigma)$ recursively by $t \sqsubseteq s$ iff t, s have the form $t = \gamma.f(t_1, \dots, t_n)$ and $s = \delta.f(s_1, \dots, s_n)$ where $\gamma \leq \delta$ and $t_i \sqsubseteq s_i$ for $i = 1, \dots, n$. For a literal language L , $\downarrow L = \{t \in \mathcal{T}_{\bar{\mathbb{A}}}(\Sigma) \mid \exists t' \in L. t \sqsubseteq t'\}$ denotes the downward closure of L with respect to \sqsubseteq .

That is, $t \sqsubseteq t'$ if t arises from t' by removing zero or more occurrences of ν ; e.g. $\nu a.f(a.f(a.f(a.k))) \sqsubseteq \nu a.f(\nu a.f(a.k))$.

► **Lemma 6.5.** For closed alphatic languages L_1, L_2 , we have $D(L_1) \subseteq D(L_2)$ iff for all $[t] \in L_1$ there exists $t' \sqsubseteq t'$ such that $[t'] \in L_2$.

► **Theorem 6.6.** *Language inclusion $D(L_\alpha(A)) \subseteq D(L_\alpha(B))$ under local freshness semantics of RNTAs A, B of degrees d_A, d_B , respectively, over the fixed signature Σ is decidable in doubly exponential time, and in fact in parametrized singly exponential time with the degree as the parameter, i.e. exponential in a function that depends exponentially on $d_A + d_B$ and polynomially on the size of A, B .*

Proof. The proof is largely analogous to that of Theorem 6.3. In step 1., one shows using Lemma 6.5 (and recalling Remark 4.4) that $D(L_\alpha(A)) \subseteq D(L_\alpha(B))$ iff $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq \downarrow(L(B_\perp) \cap \mathcal{T}_S(\Sigma))$. In step 2., the NFTA accepting $\downarrow(L(B_\perp) \cap \mathcal{T}_S(\Sigma))$ is constructed as in Theorem 6.3 and then closed downwards under \sqsubseteq by adding a transition $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))$ for every transition $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n))$. ◀

7 Conclusions

We have introduced the model of *regular nominal tree automata (RNTA)*, a species of non-deterministic top-town nominal tree automata. RNTAs can be equipped with different data tree semantics ranging from global freshness as found in session automata [6] to local freshness. Under the latter, RNTAs correspond, via the usual equivalence of nominal automata and register-based automata [4, 34], to a subclass of register tree automata [21]. As such, they are less expressive than the full register model, but in return admit inclusion checking in elementary complexity (parametrized exponential time); this in a model that allows unboundedly many registers and unrestricted nondeterminism (cf. Section 1). RNTAs feature a native notion of name allocation, allowing them to process terms in languages with name binding such as the λ - and the π -calculus.

In future research, we aim to work towards a notion of nominal automata with name allocation for infinite trees, in particular with a view to applications in reasoning over name-allocating fragments of the nominal μ -calculus [23]. Also, it will be of interest to develop the theory in coalgebraic generality [33], aiming to support automata with effects such as probabilistic or weighted branching.

References

- 1 Jiří Adámek, Horst Herrlich, and George E. Strecker. *Abstract and Concrete Categories*. John Wiley and Sons, 1990. Reprint: <http://www.tac.mta.ca/tac/reprints/articles/17/tr17abs.html>.
- 2 Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
- 3 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.
- 4 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.
- 5 Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009. doi:10.1145/1516512.1516515.
- 6 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A robust class of data languages and an application to learning. *Log. Meth. Comput. Sci.*, 10(4), 2014. doi:10.2168/LMCS-10(4:19)2014.

- 7 Thomas Colcombet. Unambiguity in automata theory. In *Descriptional Complexity of Formal Systems, DCFS 2015*, volume 9118 of *LNCS*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3.
- 8 Thomas Colcombet and Amaldev Manuel. μ -calculus on data words. *CoRR*, 2014. arXiv:1404.4827.
- 9 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. HAL Portal Inria, 2008. hal-03367725. URL: <https://hal.inria.fr/hal-03367725/document>.
- 10 Wojciech Czerwiński and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. *CoRR*, 2021. arXiv:2104.13866.
- 11 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.
- 12 Diego Figueira. Forward-XPath and extended register automata on data-trees. In Luc Segoufin, editor, *Database Theory, ICDT 2010*, pages 231–241. ACM, 2010. doi:10.1145/1804669.1804699.
- 13 Murdoch James Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Foundations of Software Science and Computation Structures, FOSSACS 2011*, volume 6604 of *LNCS*, pages 365–380. Springer, 2011. doi:10.1007/978-3-642-19805-2.
- 14 Radu Grigore, Dino Distefano, Rasmus Petersen, and Nikos Tzevelekos. Runtime verification based on register automata. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2013*, volume 7795 of *LNCS*, pages 260–276. Springer, 2013. doi:10.1007/978-3-642-36742-7_19.
- 15 Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Model checking systems and specifications with parameterized atomic propositions. In *Automated Technology for Verification and Analysis, ATVA 2012*, volume 7561 of *LNCS*, pages 122–136. Springer, 2012. doi:10.1007/978-3-642-33386-6_11.
- 16 Daniel Hausmann, Stefan Milius, and Lutz Schröder. A linear-time nominal μ -calculus with name allocation. In Filippo Bonchi and Simon J. Puglisi, editors, *Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPICs*, pages 58:1–58:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.MFCS.2021.58.
- 17 Falk Howar, Bengt Jonsson, and Frits Vaandrager. Combining black-box and white-box techniques for learning register automata. In *Computing and Software Science – State of the Art and Perspectives*, volume 10000 of *LNCS*, pages 563–588. Springer, 2019. doi:10.1007/978-3-319-91908-9_26.
- 18 Marcin Jurdziński and Ranko Lazić. Alternation-free modal μ -calculus for data trees. In *Logic in Computer Science, LICS 2007*, pages 131–140. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.11.
- 19 Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *ACM Trans. Comput. Log.*, 12(3):19:1–19:21, 2011. doi:10.1145/1929954.1929956.
- 20 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 21 Michael Kaminski and Tony Tan. Tree automata over infinite alphabets. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *LNCS*, pages 386–423. Springer, 2008. doi:10.1007/978-3-540-78127-1_21.
- 22 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760, 2010. doi:10.1142/S0129054110007532.
- 23 Bartek Klin and Mateusz Łętyk. Scalar and vectorial μ -calculus with atoms. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:5)2019.
- 24 Klaas Kürtz, Ralf Küsters, and Thomas Wilke. Selecting theories and nonce generation for recursive protocols. In *Formal methods in security engineering, FMSE 2007*, pages 61–70. ACM, 2007. doi:10.1145/1314436.1314445.

- 25 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. *CoRR*, 2021. [arXiv:2104.12695](https://arxiv.org/abs/2104.12695).
- 26 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Logic in Computer Science, LICS 2019*, pages 1–13. IEEE, 2019. [doi:10.1109/LICS.2019.8785796](https://doi.org/10.1109/LICS.2019.8785796).
- 27 Amaldev Manuel, Anca Muscholl, and Gabriele Puppis. Walking on data words. *Theory Comput. Sys.*, 59(2):180–208, 2016. [doi:10.1007/s00224-014-9603-3](https://doi.org/10.1007/s00224-014-9603-3).
- 28 Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata. In *Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPICs*, pages 53:1–53:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. [doi:10.4230/LIPICs.STACS.2019.53](https://doi.org/10.4230/LIPICs.STACS.2019.53).
- 29 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004. [doi:10.1145/1013560.1013562](https://doi.org/10.1145/1013560.1013562).
- 30 Daniela Petrişan. *Investigations into Algebra and Topology over Nominal Sets*. PhD thesis, University of Leicester, 2011.
- 31 Andrew Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.
- 32 Simon Prucker and Lutz Schröder. Nominal tree automata with name allocation. *CoRR*, abs/2405.14272, 2024. [doi:10.48550/arXiv.2405.14272](https://doi.org/10.48550/arXiv.2405.14272).
- 33 Jan Rutten. Universal coalgebra: A theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000.
- 34 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In *Foundations of Software Science and Computation Structures, FOSSACS 2017*, volume 10203 of *LNCS*, pages 124–142, 2017. [doi:10.1007/978-3-662-54458-7_8](https://doi.org/10.1007/978-3-662-54458-7_8).
- 35 Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990. [doi:10.1137/0219027](https://doi.org/10.1137/0219027).
- 36 Ryoma Senda, Yoshiaki Takata, and Hiroyuki Seki. Complexity results on register context-free grammars and register tree automata. In *Theoretical Aspects of Computing, ICTAC 2018*, volume 11187 of *LNCS*, pages 415–434. Springer, 2018. [doi:10.1007/978-3-030-02508-3_22](https://doi.org/10.1007/978-3-030-02508-3_22).
- 37 Tony Tan. Extending two-variable logic on data trees with order on data values and its automata. *ACM Trans. Comput. Log.*, 15(1):8:1–8:39, 2014. [doi:10.1145/2559945](https://doi.org/10.1145/2559945).
- 38 Szymon Torunczyk and Thomas Zeume. Register automata with extrema constraints, and an application to two-variable logic. *Log. Methods Comput. Sci.*, 18(1), 2022. [doi:10.46298/LMCS-18\(1:42\)2022](https://doi.org/10.46298/LMCS-18(1:42)2022).
- 39 Nikos Tzevelekos. *Nominal Game Semantics*. PhD thesis, University of Oxford, 2008.
- 40 Henning Urbat, Daniel Hausmann, Stefan Milius, and Lutz Schröder. Nominal büchi automata with name allocation. In Serge Haddad and Daniele Varacca, editors, *Concurrency Theory, CONCUR 2021*, volume 203 of *LIPICs*, pages 4:1–4:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. [doi:10.4230/LIPICs.CONCUR.2021.4](https://doi.org/10.4230/LIPICs.CONCUR.2021.4).
- 41 Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino, and Alexandra Silva. Tree automata as algebras: Minimisation and determinisation. In *Algebra and Coalgebra in Computer Science, CALCO 2019*, volume 139 of *LIPICs*, pages 6:1–6:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. [doi:10.4230/LIPICs.CALCO.2019.6](https://doi.org/10.4230/LIPICs.CALCO.2019.6).

Branching Bisimilarity for Processes with Time-Outs

Gaspard Reghem ✉

ENS Paris-Saclay, Université Paris-Saclay, France

Rob J. van Glabbeek ✉ 🏠 

School of Informatics, University of Edinburgh, UK

School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

Abstract

This paper provides an adaptation of branching bisimilarity to reactive systems with time-outs. Multiple equivalent definitions are procured, along with a modal characterisation and a proof of its congruence property for a standard process algebra with recursion. The last section presents a complete axiomatisation for guarded processes without infinite sequences of unobservable actions.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Reactive Systems, Time-outs, Branching Bisimilarity, Modal Characterisation, Congruence, Axiomatisation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.36

Related Version *Full Version*: <https://arxiv.org/abs/2408.10117> [20]

Funding Supported by Royal Society Wolfson Fellowship RSWF\R1\221008.

1 Introduction

Strong bisimilarity [17] is the default semantic equivalence on labelled transition systems (LTSs), modelling systems that move from state to state by performing discrete, uninterpreted actions. In [11], it has been generalised, under the name *strong reactive bisimilarity*, to LTSs that feature, besides the hidden action τ [17], an unobservable *time-out* action t [9], modelling the end of a time-consuming activity from which we abstract. This addition significantly increases the expressiveness of the model [10, 11].

Applied to the verification of realistic distributed systems, strong bisimilarity is too fine an equivalence, especially because it does not cater to abstraction from internal activity. *Branching bisimilarity* [13] is a variant that does abstract from internal activity, and lies at the basis of many verification toolsets [3, 6]. The present paper generalises branching bisimilarity to LTSs with time-outs, thereby combining the virtues of [11] and [13]. It supports the resulting notion of *branching reactive bisimilarity* through a modal characterisation, congruence results for a standard process algebra with recursion, and a complete axiomatisation.

The addition of the time-out action t aims at modelling the passage of time while staying in the realm of *untimed* process algebra. Here, “untimed” means that our framework does not facilitate measuring time, even though it models whether a system can pause in some state or not. We assume that the execution of any action is instantaneous; thus, time elapses in states only. The amount of time spent in a state is dictated by the interaction of the system with an external entity called its *environment*.

We call a system *reactive* if it interacts with an environment able to allow or disallow visible actions. The environment represents a user or other systems, running in parallel, which has no control over τ or t actions. If X is the set of visible actions currently allowed by the environment and the system can perform any transition labelled by an element of $X \cup \{\tau\}$ then it will perform one of those transitions immediately. When a visible action



© Gaspard Reghem and Rob J. van Glabbeek;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 36; pp. 36:1–36:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is performed, it triggers the environment to choose a new set of allowed actions. If the environment is allowing X and the system cannot perform any action from $X \cup \{\tau\}$, then the system is said to be *idling*. When the system idles, time-outs become executable, but the environment can also get impatient and choose a new X before any time-out occurs.

We have supposed that the environment cannot synchronise with the execution of a time-out, thus implying that, right after executing a time-out, the environment is still allowing the same set of allowed actions as before this execution. For example, the process $a.P + t.(a.Q + \tau.R)$ will never reach Q because, for the time-out to happen, the environment has to block a and so $a.Q + \tau.R$ can only be reached when the environment blocks a . In this case, the τ -transition is always executed before the environment can allow a again.

Similarly, strong and branching reactive bisimilarity satisfy the process algebraic law $\tau.P + t.Q = \tau.P$, essentially giving τ priority over t . Whereas this could have been formalised through an operational semantics in which the process $\tau.P + t.Q$ lacks an outgoing t -transition, here, and in [11], we derive an LTS for a standard process algebra with time-outs in a way that treats t just like any other action. Instead, the priority of τ over t is implemented in the reactive bisimilarity: it says that even though the transition $\tau.P + t.Q \xrightarrow{t} Q$ is present in our LTS, it will never be taken. This approach is not only simpler, it also generalises better to choices like $b.P + t.Q$, where the priority of b over t is conditional on the environment in which the system is placed, namely on whether or not this environment allows the b -action to occur.

From the system's perspective, the environment can be in two kinds of states: either allowing a specific set of actions, or being triggered to change. Our model does not stipulate how much time the environment takes to choose a new set of allowed actions once triggered, or even if it will ever make such a choice. Thus, the system could perform some transitions while the environment is triggered, especially those labelled τ . In our view, the most natural way to see the environment is as another system executed in parallel, while enforcing synchronisation on all visible actions. This implies that the environment allows a set X of actions when it idles in a state whose set of initial actions is X , and the environment is triggered when it is not idling, especially when it can perform a τ -transition. In this paradigm, while the environment is triggered, any action can be allowed for a brief amount of time. However, there is no reason to believe that it will necessarily settle down on a specific set. For instance, this can happen if the environment reaches a *divergence*: an infinite sequence of τ -transitions.

In [7], seven (or nine) forms of branching bisimilarity are classified; they differ only in the treatment of divergence. In the present paper we are chiefly interested in divergence-free processes, on grounds that in the intuition of [11] any sequence of τ -transitions could be executed in time zero; yet we do wish to allow infinite sequences of t -transitions. For divergence-free process all these forms of branching bisimilarity coincide. Nevertheless, we do not formally exclude divergences, and in their presence our branching reactive bisimilarity generalises the *stability respecting branching bisimilarity* of [7], which differs from the default version from [13] through the presence of Clause 2.e of Definition 1. There does not exist a plausible reactive generalisation of the default version.

Section 2 supplies the formal definition of branching reactive bisimilarity as well as its rooted version, which will be shown to be its congruence closure. It also provides equivalent definitions that reduce our bisimilarity to a non-reactive one and illustrate that branching reactive bisimilarity coincides with stability respecting branching bisimilarity in the absence of time-outs.

Section 3 gives a modal characterisation of branching reactive bisimilarity and its rooted version on an extension of the Hennessy-Milner logic. Section 4 introduces the process algebra CCSP_t^θ along with an alternative characterisation of branching reactive bisimilarity that will be used to prove that rooted branching reactive bisimilarity is a full congruence for CCSP_t^θ .

Section 5 displays a complete axiomatisation of our bisimilarity on different fragments of CCSP_t^0 . Most completeness proofs rely on standard techniques like equation merging, but the very last one uses a relatively new method called “canonical representatives”.

2 Branching Reactive Bisimilarity

A *labelled transition system* (LTS) is a triple $(\mathbb{P}, \text{Act}, \rightarrow)$ with \mathbb{P} a set (of *states* or *processes*), Act a set (of *actions*) and $\rightarrow \in \mathbb{P} \times \text{Act} \times \mathbb{P}$. In this paper we consider LTSs with $\text{Act} := A \uplus \{\tau, \text{t}\}$, where A is a set of *visible actions*, τ is the *hidden or invisible action*, and t the *time-out action*. Let $A_\tau := A \cup \{\tau\}$. $P \xrightarrow{\alpha} P'$ stands for $(P, \alpha, P') \in \rightarrow$ and these triplets are called *transitions*. Moreover, $P \xrightarrow{(\alpha)} P'$ denotes that either $\alpha = \tau$ and $P = P'$, or $P \xrightarrow{\alpha} P'$. Furthermore, *paths* are sequences of connected transitions and \Longrightarrow is the reflexive-transitive closure of $\xrightarrow{\tau}$. The set of *initial actions* of a process $P \in \mathbb{P}$ is $\mathcal{I}(P) := \{\alpha \in A_\tau \mid P \xrightarrow{\alpha}\}$. Here $P \xrightarrow{\alpha}$ means that there is a Q with $P \xrightarrow{\alpha} Q$.

► **Definition 1.** A *branching reactive bisimulation* is a symmetric¹ relation $\mathcal{R} \subseteq (\mathbb{P} \times \mathbb{P}) \cup (\mathbb{P} \times \mathcal{P}(A) \times \mathbb{P})$ such that, for all $P, Q \in \mathbb{P}$ and $X \subseteq A$,

1. if $\mathcal{R}(P, Q)$ then
 - a. if $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{(\alpha)} Q_2$ with $\mathcal{R}(P, Q_1)$ and $\mathcal{R}(P', Q_2)$,
 - b. for all $Y \subseteq A$, $\mathcal{R}(P, Y, Q)$;
2. if $\mathcal{R}(P, X, Q)$ then
 - a. if $P \xrightarrow{\tau} P'$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{(\tau)} Q_2$ with $\mathcal{R}(P, X, Q_1)$ and $\mathcal{R}(P', X, Q_2)$,
 - b. if $P \xrightarrow{a} P'$ with $a \in X$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{a} Q_2$ with $\mathcal{R}(P, X, Q_1)$ and $\mathcal{R}(P', Q_2)$,
 - c. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ then there is a path $Q \Longrightarrow Q_0$ with $\mathcal{R}(P, Q_0)$,
 - d. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{\text{t}} P'$ then there is a path $Q =: Q_0 \Longrightarrow Q_1 \xrightarrow{\text{t}} Q_2 \Longrightarrow Q_3 \xrightarrow{\text{t}} \dots \Longrightarrow Q_{2r-1} \xrightarrow{(\text{t})} Q_{2r}$ with $r > 0$, such that $\forall i \in [0, r-1], \mathcal{R}(P, X, Q_{2i}) \wedge \mathcal{I}(Q_{2i+1}) \cap (X \cup \{\tau\}) = \emptyset$ and $\mathcal{R}(P', X, Q_{2r})$,
 - e. if $P \not\xrightarrow{\tau}$ then there is a path $Q \Longrightarrow Q_0 \not\xrightarrow{\tau}$.

For $P, Q \in \mathbb{P}$, if there exists a branching reactive bisimulation \mathcal{R} with $\mathcal{R}(P, Q)$ (resp. $\mathcal{R}(P, X, Q)$) then P and Q are said to be *branching reactive bisimilar* (resp. *branching X -bisimilar*), which is denoted $P \leftrightarrow_{br} Q$ (resp. $P \leftrightarrow_{br}^X Q$).

To build the above definition, the definition of a strong reactive bisimulation [11] was modified in a branching manner [13]. Intuitively, a triplet $\mathcal{R}(P, X, Q)$ affirms that P and Q behave similarly when the environment allows (only) the set of actions in X to occur, whereas a couple $\mathcal{R}(P, Q)$ says that P and Q behave in the same way when the environment has been triggered to change. As said before, the environment can be seen as a system executed in parallel while enforcing the synchronisation of all visible actions.

Clause 1 captures the scenario of a triggered environment: if P can perform a visible or invisible action then Q has to be able to match it; and the environment can settle on a set Y of allowed actions at any moment. Time-outs are not considered because these can occur only when the system idles, and idling can happen only when the environment has stabilised on a set of allowed actions. One might notice that, in [11], the first clause was only required for invisible actions. However, there the case $\alpha \neq \tau$ is actually implied by the other clauses. If in our definition Clause 1.a were restricted to invisible actions then \leftrightarrow_{br} would not be a congruence for the parallel operator, as shown in Appendix A.

¹ meaning that $(P, Q) \in \mathcal{R} \Leftrightarrow (Q, P) \in \mathcal{R}$ and $(P, X, Q) \in \mathcal{R} \Leftrightarrow (Q, X, P) \in \mathcal{R}$

Clause 2 depicts the scenario of an environment allowing X . τ -transitions have to be matched since the environment cannot disallow them, and their execution does not trigger the environment to change. Visible actions have to be matched only if they are allowed, and their execution triggers the environment. Triggering the environment or not explains why Clause 2a matches Q_2 in a triplet and Clause 2b in a couple. If P idles (i.e. $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$) then the environment can be triggered, thus, Q has to be able to instantaneously reach a state Q_0 related to P in a triggered environment.² If P idles and has an outgoing time-out transition then Q has to be able to match it in a branching manner. This involves Q performing any sequence of τ and t -transitions, such that all states encountered prior to the last optional t are related to P .³ Lastly, a stability respecting clause [7] was added for practical reasons. In Appendix A, an example shows that without it \Leftrightarrow_{br} would not even be an equivalence. For the important class of *divergence-free* systems, without infinite sequences $Q_0 \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \dots$, Clause 2.e is easily seen to be redundant.

► **Lemma 2.** *Let \mathcal{R} be a branching reactive bisimulation.*

1. If $\mathcal{R}(P, X, Q)$, $P \not\xrightarrow{\tau}$ and $Q \Longrightarrow Q'$ then also $\mathcal{R}(P, X, Q')$.
2. If $\mathcal{R}(P, Q)$ or $\mathcal{R}(P, X, Q)$, $P \not\xrightarrow{\tau}$ and $Q \not\xrightarrow{\tau}$ then $\mathcal{I}(Q) = \mathcal{I}(P)$.
3. If $\mathcal{R}(P, X, Q)$, $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $Q \not\xrightarrow{\tau}$ then $\mathcal{R}(P, Q)$.
4. If $\mathcal{R}(P, X, Q)$ and $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ then there is a path $Q \Longrightarrow Q_0$ with $\mathcal{R}(P, Q_0)$, $Q_0 \not\xrightarrow{\tau}$ and $\mathcal{I}(Q_0) = \mathcal{I}(P)$.

Proof.

1. This is an immediate consequence of the symmetric counterpart of Clause 2.a (where Q takes a τ -step). When that clause yields $P \Longrightarrow P_1 \xrightarrow{(\tau)} P_2$ we have $P_2 = P$.
2. This is a direct consequence of Clause 1.a or 2.b and its symmetric counterpart.
3. By Clause 2.e there is path $Q \Longrightarrow Q_0$ with $Q_0 \not\xrightarrow{\tau}$. By Claim 1 of this lemma, $\mathcal{R}(P, X, Q_0)$. Thus, by Clause 2.c there is a path $Q_0 \Longrightarrow Q_1$ with $\mathcal{R}(P, Q_1)$, but $Q_1 = Q_0 = Q$ since $Q \not\xrightarrow{\tau}$.
4. By Clause 2.e there is path $Q \Longrightarrow Q_0$ with $Q_0 \not\xrightarrow{\tau}$. By Claim 1 of this lemma, $\mathcal{R}(P, X, Q_0)$. That $\mathcal{I}(Q_0) = \mathcal{I}(P)$ and $\mathcal{R}(P, Q_0)$ follows by Claims 2 and 3 of this lemma. ◀

Definition 1 enables us to elide some time-outs. Using the process algebra notation to be formally introduced in Section 4, the processes $a.t.b.0$ and $a.t.t.b.0$ (as well as $a.t.\tau.t.b.0$) are branching reactive bisimilar. Both require an unquantified positive but finite amount of rest between the actions a and b . To support this example, Clause 2.d of Definition 1 must allow a single time-out transition of one process to be matched by either zero or multiple time-outs of the other. An alternative definition, treating time-outs more like visible transitions, is obtained by replacing Clause 2.d by

2. d. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{t} Q_2$ with $\mathcal{R}(P', X, Q_2)$.

Requiring that the matching time-out is executable (i.e. $\mathcal{I}(Q_1) \cap (X \cup \{\tau\}) = \emptyset$) is not necessary here, as it is implied by the other clauses. Indeed, Lemma 2.3, which is not affected by changing Clause 2.d, implies the existence of a path $Q \Longrightarrow Q_1 \not\xrightarrow{\tau}$ such that $\mathcal{R}(P, Q_1)$ and $\mathcal{I}(Q_1) \cap (X \cup \{\tau\}) = \emptyset$. Since $Q_1 \not\xrightarrow{\tau}$, $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$, Clause 2d yields $Q_1 \xrightarrow{t} Q_2$ with $\mathcal{R}(P', X, Q_2)$. This version of the definition has been studied [21] and has properties similar to \Leftrightarrow_{br} , which are recapped in Appendix B.

² By Lemma 2.4 we can even choose Q_0 such that $Q_0 \not\xrightarrow{\tau}$, so that $\mathcal{I}(Q_0) = \mathcal{I}(P)$.

³ Clause 2.d requires this only for states of the form Q_{2i} with $i \in [0, r-1]$, but by Lemma 2.1 it holds for all of them. Clause 2.c further implies that in Clause 2.d we have $\mathcal{R}(P, Q_{2i+1})$ for all $i \in [0, r-1]$.

In [13], branching bisimilarity is expressed in multiple equivalent ways. For practical purposes, our definition uses the semi-branching format, which is equivalent to the branching format thanks to the following lemma.

► **Lemma 3** (Stuttering Lemma). *Let $P, P^\dagger, P^\ddagger, Q \in \mathbb{P}$, if $P \leftrightarrow_{br} Q$, $P^\ddagger \leftrightarrow_{br} Q$ (resp. $P \leftrightarrow_{br}^X Q$, $P^\ddagger \leftrightarrow_{br}^X Q$) and $P \xrightarrow{\tau} P^\dagger \xrightarrow{\tau} P^\ddagger$ then $P^\dagger \leftrightarrow_{br} Q$ (resp. $P^\dagger \leftrightarrow_{br}^X Q$).*

Proof. Let \mathcal{R} be a branching reactive bisimulation. Let's define $\mathcal{R}' := \mathcal{R} \cup \{(P^\dagger, Q), (Q, P^\dagger) \mid \exists P, P^\ddagger \in \mathbb{P}, P \Longrightarrow P^\dagger \Longrightarrow P^\ddagger \wedge \mathcal{R}(P, Q) \wedge \mathcal{R}(P^\ddagger, Q)\} \cup \{(P^\dagger, X, Q), (Q, X, P^\dagger) \mid \exists P, P^\ddagger \in \mathbb{P}, P \Longrightarrow P^\dagger \Longrightarrow P^\ddagger \wedge \mathcal{R}(P, X, Q) \wedge \mathcal{R}(P^\ddagger, X, Q)\}$. \mathcal{R}' is symmetric by definition and \mathcal{R}' is a branching reactive bisimulation, as proven in [20, Appendix E]. ◀

► **Proposition 4.** \leftrightarrow_{br} and $(\leftrightarrow_{br}^X)_{X \subseteq A}$ are equivalence relations.

Proof. Reflexivity and symmetry are trivial following the definition. For transitivity, consider two branching reactive bisimulations \mathcal{R}_1 and \mathcal{R}_2 . Let's define $\mathcal{R} := (\mathcal{R}_1 \circ \mathcal{R}_2) \cup (\mathcal{R}_2 \circ \mathcal{R}_1)$. Here $\mathcal{R}_1 \circ \mathcal{R}_2 := \{(P, Q) \mid \exists R. \mathcal{R}_1(P, R) \wedge \mathcal{R}_2(R, Q)\} \cup \{(P, X, Q) \mid \exists R. \mathcal{R}_1(P, X, R) \wedge \mathcal{R}_2(R, X, Q)\}$. \mathcal{R} is symmetric by definition and \mathcal{R} is a branching reactive bisimulation, as proven in [20, Appendix E]. ◀

2.1 Rooted Version

A well-known limitation of branching bisimilarity \leftrightarrow_b is that it fails to be a congruence for the choice operator $+$. For example, $a \leftrightarrow_b \tau.a$ but $a + b \not\leftrightarrow_b \tau.a + b$. Since the objective is to define a congruence, instead of \leftrightarrow_{br} we use the *congruence closure* of \leftrightarrow_{br} , which is the coarsest congruence included in \leftrightarrow_{br} .

► **Definition 5.** A *rooted branching reactive bisimulation* is a symmetric relation $\mathcal{R} \subseteq (\mathbb{P} \times \mathbb{P}) \cup (\mathbb{P} \times \mathcal{P}(A) \times \mathbb{P})$ such that, for all $P, Q \in \mathbb{P}$ and $X \subseteq A$,

1. if $\mathcal{R}(P, Q)$
 - a. if $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau$ then there is a transition $Q \xrightarrow{\alpha} Q'$ with $P' \leftrightarrow_{br} Q'$,
 - b. for all $Y \subseteq A$, $\mathcal{R}(P, Y, Q)$;
2. if $\mathcal{R}(P, X, Q)$
 - a. if $P \xrightarrow{\tau} P'$ then there is a transition $Q \xrightarrow{\tau} Q'$ with $P' \leftrightarrow_{br}^X Q'$,
 - b. if $P \xrightarrow{a} P'$ with $a \in X$ then there is a transition $Q \xrightarrow{a} Q'$ with $P' \leftrightarrow_{br} Q'$,
 - c. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ then $\mathcal{R}(P, Q)$,
 - d. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a transition $Q \xrightarrow{t} Q'$ with $P' \leftrightarrow_{br}^X Q'$.

For $P, Q \in \mathbb{P}$, if there exists a rooted branching reactive bisimulation \mathcal{R} with $\mathcal{R}(P, Q)$ (resp. $\mathcal{R}(P, X, Q)$) then P and Q are said to be *rooted branching reactive bisimilar* (resp. *rooted branching X -bisimilar*), which is denoted $P \leftrightarrow_{br}^r Q$ (resp. $P \leftrightarrow_{br}^{rX} Q$).

A rooted version of a bisimulation consists in enforcing a stricter matching on the first transition of a system. In the branching case, the first transition is matched in the strong manner. The stability respecting clause can be removed, as it is now implied by the other clauses. Rooting the bisimilarity is the standard technique to obtain its congruence closure; later \leftrightarrow_{br}^r will be proven to be a congruence. As any branching reactive bisimulation relating $P + b$ and $Q + b$, for a fresh action b , induces a rooted branching reactive bisimulation relating P and Q , it then follows that \leftrightarrow_{br}^r is the coarsest included in \leftrightarrow_{br} . Since \leftrightarrow_{br} is an equivalence, the proof of Proposition 4 can be adapted to \leftrightarrow_{br}^r in a straightforward way.

► **Proposition 6.** \leftrightarrow_{br}^r and $(\leftrightarrow_{br}^{rX})_{X \subseteq A}$ are equivalence relations.

2.2 Alternative Forms of Definition 1

Definition 1 can be rephrased in various ways. First of all, using Requirements 1.b and 2.c, one can move Requirement 2.d from Clause 2 (dealing with triples (P, X, Q)) to Clause 1 (dealing with pairs (P, Q)), now adding a universal quantifier over X to the requirement. Next, Requirement 2.e can be copied under Clause 1. This makes Clause 1.b unnecessary, thereby obtaining a definition in which the triples (P, X, Q) are encountered only after taking a t -transition. In this form it is obvious that branching reactive bisimilarity reduces to the classical stability respecting branching bisimilarity for systems without t -transitions. We have chosen the form of Definition 1 over the above alternatives, because we believe it comes with more natural intuitions for its plausibility.

In Appendix C a further modification of Definitions 1 and 5 is proposed, called *generalised [rooted] branching reactive bisimulation*. We show that each [rooted] branching reactive bisimulation is a generalised [rooted] branching reactive bisimulation, and two systems are [rooted] branching reactive bisimilar iff they are related by a generalised [rooted] branching reactive bisimulation. This characterisation of \Leftrightarrow_{br} and \Leftrightarrow_{br}^r will be used in the proofs of Theorem 11 and Proposition 15.

In [19], Pohlmann introduces an encoding which maps strong reactive bisimilarity to strong bisimilarity where time-outs are considered as any visible action. This encoding in essence places a given process in a most general environment, one that features environment time-out actions t_ε , as well as actions ε_X for settling in a state that allows exactly the actions in X . This proves that reactive equivalences can be expressed as non-reactive ones at the cost of increasing the processes' size. Thus, any tool set able to work on strong bisimulation could theoretically deal with its reactive counterpart.

In Appendix D, this encoding is slightly modified to yield a similar result for branching reactive bisimulation and its rooted version, for the latter result also employing actions t_X . It appears that these modifications do not impact its effect on strong reactive bisimilarity. Since our bisimilarity has some time-out eliding properties, it is not mapped to stability respecting branching bisimilarity, but to a new bisimilarity, defined below.

► **Definition 7.** A *t-branching bisimulation* is a symmetric relation $\mathcal{R} \subseteq \mathbb{P} \times \mathbb{P}$ such that, for all $P, Q \in \mathbb{P}$, if $\mathcal{R}(P, Q)$ then

1. if $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau \cup \{t_\varepsilon, \varepsilon_X \mid X \subseteq A\}$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{(\alpha)} Q_2$ with $\mathcal{R}(P, Q_1)$ and $\mathcal{R}(P', Q_2)$,
2. if $P \xrightarrow{t} P'$ then there is a path $Q = Q_0 \Longrightarrow Q_1 \xrightarrow{t} Q_2 \Longrightarrow Q_3 \xrightarrow{t} \dots \Longrightarrow Q_{2r-1} \xrightarrow{(t)} Q_{2r}$ with $r > 0$, such that $\forall i \in [0, 2r-1]$, $\mathcal{R}(P, Q_i)$ and $\mathcal{R}(P', Q_{2r})$,
3. if $P \not\xrightarrow{\tau}$ then there is a path $Q \Longrightarrow Q_0 \not\xrightarrow{\tau}$.

For $P, Q \in \mathbb{P}$, if there exists a t -branching bisimulation \mathcal{R} with $\mathcal{R}(P, Q)$ then P and Q are said to be *t-branching bisimilar*, which is denoted $P \Leftrightarrow_{tb} Q$.

The encoding also sends \Leftrightarrow_{br}^r to the rooted version of \Leftrightarrow_{tb} .

► **Definition 8.** A *rooted t-branching bisimulation* is a symmetric relation $\mathcal{R} \subseteq \mathbb{P} \times \mathbb{P}$ such that, for all $P, Q \in \mathbb{P}$, if $\mathcal{R}(P, Q)$ then

1. if $P \xrightarrow{\alpha} P'$ with $\alpha \in Act \cup \{t_\varepsilon, t_X, \varepsilon_X \mid X \subseteq A\}$ then there is a transition $Q \xrightarrow{\alpha} Q'$ with $P' \Leftrightarrow_{tb} Q'$.

For $P, Q \in \mathbb{P}$, if there exists a rooted t -branching bisimulation \mathcal{R} with $\mathcal{R}(P, Q)$ then P and Q are said to be *rooted t-branching bisimilar*, which is denoted $P \Leftrightarrow_{tb}^r Q$.

Providing a complete axiomatisation of rooted t -branching bisimilarity will be useful in the proof of completeness of the axiomatisation of rooted branching reactive bisimilarity.

3 Modal Characterisation

The Hennessy-Milner logic [15] expresses properties of the behaviour of processes in an LTS. In [11], the modality $\langle X \rangle \varphi$ was added to obtain a modal characterisation of strong reactive bisimilarity (\Leftrightarrow_r). In order to capture branching reactive bisimilarity we add another modality $X\varphi$. To avoid confusion, $\langle X \rangle \varphi$ is renamed $\langle t_X \rangle \varphi$.

► **Definition 9.** The class \mathbb{L} of *reactive Hennessy-Milner formulas* is defined as follows, where I is an index set, $\alpha \in Act$, $a \in A$ and $X \subseteq A$,

$$\varphi ::= \top \mid \bigwedge_{i \in I} \varphi_i \mid \neg \varphi \mid \langle \alpha \rangle \varphi \mid X\varphi$$

■ **Table 1** Semantics of \models and $(\models_Y)_{Y \subseteq A}$.

$P \models \top$		$P \models_Y \top$	
$P \models \bigwedge_{i \in I} \varphi_i$	iff $\forall i \in I, P \models \varphi_i$	$P \models_Y \bigwedge_{i \in I} \varphi_i$	iff $\forall i \in I, P \models_Y \varphi_i$
$P \models \neg \varphi$	iff $P \not\models \varphi$	$P \models_Y \neg \varphi$	iff $P \not\models_Y \varphi$
$P \models \langle \alpha \rangle \varphi$	iff $\exists P \xrightarrow{\alpha} P', P' \models \varphi$	$P \models_Y \langle \tau \rangle \varphi$	iff $\exists P \xrightarrow{\tau} P', P' \models_Y \varphi$
		$P \models_Y \langle t \rangle \varphi$	iff $\exists P \xrightarrow{t} P', P' \models_Y \varphi$
$P \models_Y \langle a \rangle \varphi$	iff $(a \in Y \vee \mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset) \wedge \exists P \xrightarrow{a} P', P' \models \varphi$		
$P \models X\varphi$	iff $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset \wedge P \models_X \varphi$		
$P \models_Y X\varphi$	iff $\mathcal{I}(P) \cap (X \cup Y \cup \{\tau\}) = \emptyset \wedge P \models_X \varphi$		

The satisfaction rules of \mathbb{L} are given in Table 1. $P \models \varphi$ means that P satisfies φ when the environment is triggered, and $P \models_Y \varphi$ indicates that P satisfies φ when the environment allows Y . The modality $X\varphi$ expresses that a process can idle in its current state during a period in which the environment allows the actions in X , after which it behaves according to φ .

The modality $\langle t_X \rangle \varphi$ from [11] can now be defined as $\langle t_X \rangle \varphi := X \langle t \rangle \varphi$. Write \mathbb{L}_s for the fragment of \mathbb{L} from [11], which includes $\langle t_X \rangle \varphi$ but does not feature $X\varphi$ or $\langle t \rangle \varphi$. Then the modal characterisation theorem of [11] says $P \Leftrightarrow_r Q \Leftrightarrow \forall \varphi \in \mathbb{L}_s. (P \models \varphi \Leftrightarrow Q \models \varphi)$.

Here we restrict attention to the fragment of \mathbb{L} that includes $\langle t_X \rangle \varphi$ and $X\varphi$, but not $\langle t \rangle \varphi$. On this fragment \models_Y is defined such that whenever $\mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset$ then $P \models_Y \varphi$ iff $P \models \varphi$. This is because the environment may choose to change during a period of idling.

To obtain a modal characterisation of [rooted] branching relative bisimilarity, we need a few other derived modalities. First of all, $\langle \varepsilon \rangle \varphi := \bigvee_{i \in \mathbb{N}} \langle \tau \rangle^i \varphi$. To lessen the notations, for all $\alpha \in A_\tau$, $\langle \hat{\alpha} \rangle \varphi$ denotes $\varphi \vee \langle \tau \rangle \varphi$ if $\alpha = \tau$, $\langle \alpha \rangle \varphi$ otherwise, and the modality $\langle \hat{t}_X \rangle \varphi$ denotes $\langle t_X \rangle \varphi \vee X\varphi$ or $X \langle \hat{t} \rangle \varphi$. Moreover, $\varphi \wedge \langle \hat{\alpha} \rangle \varphi'$ is shortened to $\varphi \langle \hat{\alpha} \rangle \varphi'$. Furthermore, we define $\varphi \langle \varepsilon_X \rangle \varphi' := \bigvee_{i \in \mathbb{N}} \varphi \langle \varepsilon_X \rangle^{(i)} \varphi'$, where $\varphi \langle \varepsilon_X \rangle^{(0)} \varphi' := \langle \varepsilon \rangle (\varphi \wedge \langle \hat{t}_X \rangle \varphi')$ and, for all $i > 0$, $\varphi \langle \varepsilon_X \rangle^{(i)} \varphi' := \langle \varepsilon \rangle (\varphi \wedge \langle t_X \rangle (\varphi \wedge \langle \varepsilon_X \rangle^{(i-1)} \varphi'))$. The satisfaction rules of these new modalities can be derived from the basic ones: see Table 2.

► **Definition 10.** The sub-classes \mathbb{L}_b and \mathbb{L}_b^r are defined as follows, where I is an index set, $\alpha \in A_\tau$, $X \subseteq A$, $\varphi, \varphi' \in \mathbb{L}_b$ and $\psi \in \mathbb{L}_b^r$,

$$\varphi ::= \top \mid \bigwedge_{i \in I} \varphi_i \mid \neg \varphi \mid \langle \varepsilon \rangle (\varphi \langle \hat{\alpha} \rangle \varphi') \mid \varphi \langle \varepsilon_X \rangle \varphi' \mid \langle \varepsilon \rangle \neg \langle \tau \rangle \top \quad (\mathbb{L}_b)$$

$$\psi ::= \top \mid \bigwedge_{i \in I} \psi_i \mid \neg \psi \mid \langle \alpha \rangle \varphi \mid \langle t_X \rangle \varphi \quad (\mathbb{L}_b^r)$$

■ **Table 2** Semantics of \models and $(\models_Y)_{Y \subseteq A}$ for the derived modalities.

$$\begin{array}{ll}
 P \models \langle \hat{\alpha} \rangle \varphi & \text{iff } \exists P \xrightarrow{(\alpha)} P', P' \models \varphi \quad P \models_Y \langle \hat{\tau} \rangle \varphi \quad \text{iff } \exists P \xrightarrow{(\tau)} P', P' \models_Y \varphi \\
 P \models \langle \varepsilon \rangle \varphi & \text{iff } \exists P \Longrightarrow P', P' \models \varphi \quad P \models_Y \langle \varepsilon \rangle \varphi \quad \text{iff } \exists P \Longrightarrow P', P' \models_Y \varphi \\
 P \models \varphi \langle \varepsilon_X \rangle \varphi' & \text{iff } \exists P \Longrightarrow P_1 \xrightarrow{t} P_2 \Longrightarrow P_3 \xrightarrow{t} \dots \Longrightarrow P_{2r-1} \xrightarrow{(t)} P_{2r} \text{ with } r > 0, \text{ such that} \\
 & \forall i \in [1, 2r-1] P_i \models_X \varphi \wedge P_{2r} \models_X \varphi' \text{ and} \\
 & \forall i \in [0, r-1] \mathcal{I}(P_{2i+1}) \cap (X \cup \{\tau\}) = \emptyset \\
 P \models_Y \varphi \langle \varepsilon_X \rangle \varphi' & \text{iff } \exists P \Longrightarrow P_1 \xrightarrow{t} P_2 \Longrightarrow P_3 \xrightarrow{t} \dots \Longrightarrow P_{2r-1} \xrightarrow{(t)} P_{2r} \text{ with } r > 0, \text{ such that} \\
 & \forall i \in [1, 2r-1] P_i \models_X \varphi \wedge P_{2r} \models_X \varphi' \text{ and} \\
 & \mathcal{I}(P_1) \cap (Y \cup \{\tau\}) = \emptyset \wedge \forall i \in [0, r-1] \mathcal{I}(P_{2i+1}) \cap (X \cup \{\tau\}) = \emptyset \\
 P \models \langle t_X \rangle \varphi & \text{iff } \mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset \wedge \exists P \xrightarrow{t} P', P' \models_X \varphi
 \end{array}$$

The last option for \mathbb{L}_b , inspired by [5], is used to encompass the stability respecting Clause 2.e of Definition 1.

► **Theorem 11.** *Let $P, Q \in \mathbb{P}$. For all $X \subseteq A$,*

- $P \Leftrightarrow_{br} Q$ iff $\forall \varphi \in \mathbb{L}_b, P \models \varphi \Leftrightarrow Q \models \varphi$,
- $P \Leftrightarrow_{br}^X Q$ iff $\forall \varphi \in \mathbb{L}_b, P \models_X \varphi \Leftrightarrow Q \models_X \varphi$,
- $P \Leftrightarrow_{br}^r Q$ iff $\forall \psi \in \mathbb{L}_b^r, P \models \psi \Leftrightarrow Q \models \psi$,
- $P \Leftrightarrow_{br}^{rX} Q$ iff $\forall \psi \in \mathbb{L}_b^r, P \models_X \psi \Leftrightarrow Q \models_X \psi$.

Proof. (\Rightarrow) The four propositions are proven simultaneously by structural induction on \mathbb{L}_b and \mathbb{L}_b^r in [20, Appendix F].

(\Leftarrow) Let $\equiv := \{(P, Q) \mid \forall \varphi \in \mathbb{L}_b, P \models \varphi \Leftrightarrow Q \models \varphi\} \cup \{(P, X, Q) \mid \forall \varphi \in \mathbb{L}_b, P \models_X \varphi \Leftrightarrow Q \models_X \varphi\}$, and $\equiv^r := \{(P, Q) \mid \forall \psi \in \mathbb{L}_b^r, P \models \psi \Leftrightarrow Q \models \psi\} \cup \{(P, X, Q) \mid \forall \psi \in \mathbb{L}_b^r, P \models_X \psi \Leftrightarrow Q \models_X \psi\}$. It suffices to check that \equiv [resp. \equiv^r] is a generalised [rooted] branching reactive bisimulation. This is done in [20, Appendix F]. ◀

4 Process Algebra and Congruence

The process algebra CCSP_t^θ is composed of classical operators from the well-known process algebras CCS [17], CSP [2, 18] and ACP [1, 4], as well as the time-out action t and two *environment operators* from [11], that were added in order to enable a complete axiomatisation.

► **Definition 12.** Let V be a countable set of variables, the *expressions* of CCSP_t^θ are recursively defined as follows:

$$E ::= 0 \mid x \mid \alpha.E \mid E + F \mid E \parallel_S F \mid \tau_I(E) \mid \mathcal{R}(E) \mid \theta_L^U(E) \mid \psi_X(E) \mid \langle y \mid \mathcal{S} \rangle$$

where $x \in V$, $\alpha \in \text{Act}$, $S, I, L, U, X \subseteq A$, $L \subseteq U$, $\mathcal{R} \subseteq A \times A$, \mathcal{S} is a *recursive specification*: a set of equations $\{x = \mathcal{S}_x \mid x \in V_{\mathcal{S}}\}$ with $V_{\mathcal{S}} \subseteq V$ and each \mathcal{S}_x a CCSP_t^θ expression, and $y \in V_{\mathcal{S}}$. We require that all sets $\{b \mid (a, b) \in \mathcal{R}\}$ for $a \in A$ are finite.

0 stands for a system which cannot perform any action. The expression $\alpha.E$ represents a system that first performs α and then E . The expression $E + F$ represents a choice to behave like E or F . The parallel composition $E \parallel_S F$ synchronises the execution of E and F , but only when performing actions in S . $\tau_I(E)$ represents the system E where all actions $a \in I$ are transformed into τ . The operator \mathcal{R} renames a given action $a \in A$ into a choice between all actions b with $(a, b) \in \mathcal{R}$. $\langle y \mid \mathcal{S} \rangle$ is the y -component of a solution of \mathcal{S} .

CCSP_t^θ also has two environment operators that help to develop a complete axiomatisation (like the left merge for ACP). $\theta_L^U(E)$ is the expression E plunged into an environment X such that $L \subseteq X \subseteq U$. $\theta_X^X(E)$ is denoted $\theta_X(E)$. $\psi_X(E)$ plunges E into the environment X if a

time-out occurs, but, has no effect if any other action is performed. The operational semantics of CCSP_t^θ is given in Figure 1. All operators except the environment ones follow the semantics of CCS, CSP or ACP. As $\theta_L^U(E)$ simulates the expression E plunged in an environment $L \subseteq X \subseteq U$, it has no effect on τ -transitions, which do not trigger the environment. Moreover, θ_L^U restricts the ability to perform visible actions to those allowed by the environment (i.e. included in U) and performing these actions triggers the environment. However, if the expression idles (i.e. $\mathcal{I}(E) \cap (L \cup \{\tau\}) = \emptyset$) then it might trigger the environment and $\theta_L^U(E)$ acts like E . $\psi_X(E)$ supposes that time-outs are performed while the environment allows X , thus, it has no effect on actions that are not t . However, if E can perform a time-out while the environment allows X (i.e. $\mathcal{I}(E) \cap (X \cup \{\tau\}) = \emptyset$) then $\psi_X(E)$ can perform the time-out while plunging the expression in the environment X .

$$\begin{array}{c}
\frac{}{\alpha.x \xrightarrow{\alpha} x} \quad \frac{x \xrightarrow{\alpha} x'}{x + y \xrightarrow{\alpha} x'} \quad \frac{y \xrightarrow{\alpha} y'}{x + y \xrightarrow{\alpha} y'} \\
\\
\frac{x \xrightarrow{a} x' \wedge \mathcal{R}(a, b)}{\mathcal{R}(x) \xrightarrow{b} \mathcal{R}(x')} \quad \frac{x \xrightarrow{\tau} x'}{\mathcal{R}(x) \xrightarrow{\tau} \mathcal{R}(x')} \quad \frac{x \xrightarrow{t} x'}{\mathcal{R}(x) \xrightarrow{t} \mathcal{R}(x')} \\
\\
\frac{x \xrightarrow{\alpha} x' \wedge \alpha \notin S}{x \parallel_S y \xrightarrow{\alpha} x' \parallel_S y} \quad \frac{y \xrightarrow{\alpha} y' \wedge \alpha \notin S}{x \parallel_S y \xrightarrow{\alpha} x \parallel_S y'} \quad \frac{x \xrightarrow{a} x' \wedge y \xrightarrow{a} y' \wedge a \in S}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'} \\
\\
\frac{x \xrightarrow{\alpha} x' \wedge \alpha \notin I}{\tau_I(x) \xrightarrow{\alpha} \tau_I(x')} \quad \frac{x \xrightarrow{a} x' \wedge a \in I}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')} \quad \frac{\langle \mathcal{S}_x | \mathcal{S} \rangle \xrightarrow{\alpha} x'}{\langle x | \mathcal{S} \rangle \xrightarrow{\alpha} x'} \\
\\
\frac{x \xrightarrow{\tau} x'}{\theta_L^U(x) \xrightarrow{\tau} \theta_L^U(x')} \quad \frac{x \xrightarrow{a} x' \wedge a \in U}{\theta_L^U(x) \xrightarrow{a} x'} \quad \frac{x \xrightarrow{\alpha} x' \wedge \alpha \neq t}{\psi_X(x) \xrightarrow{\alpha} x'} \\
\\
\frac{x \xrightarrow{\alpha} x' \wedge \mathcal{I}(x) \cap (L \cup \{\tau\}) = \emptyset}{\theta_L^U(x) \xrightarrow{\alpha} x'} \quad \frac{x \xrightarrow{t} x' \wedge \mathcal{I}(x) \cap (X \cup \{\tau\}) = \emptyset}{\psi_X(x) \xrightarrow{t} \theta_X(x')}
\end{array}$$

■ **Figure 1** Operational semantics of CCSP_t^θ .

All \mathcal{S}_x are considered to be sub-expressions of $\langle y | \mathcal{S} \rangle$. An occurrence of a variable x is *bound* in $E \in \text{CCSP}_t^\theta$ iff it occurs in a sub-expression $\langle y | \mathcal{S} \rangle$ of E such that $x \in V_{\mathcal{S}}$; otherwise it is *free*. An expression E is *invalid* if it has a sub-expression $\theta_L^U(F)$ or $\psi_X(F)$ such that a variable occurrence is free in F , but bound in E . An example justifying this condition can be found in [11]. The set of valid expressions of CCSP_t^θ is denoted \mathbb{E} . If an expression is valid and all of its variable occurrences are bound then it is *closed* and we call it a *process*; the set of processes is denoted \mathbb{P} .

A *substitution* is a partial function $\rho : V \rightarrow E$. The application $E[\rho]$ of a substitution ρ to an expression $E \in \mathbb{E}$ is the result of the simultaneous replacement, for all $x \in \text{dom}(\rho)$, of each free occurrence of x by the expression $\rho(x)$, while renaming bound variables to avoid name clashes. We write $\langle E | \mathcal{S} \rangle$ for the expression E where any $y \in V_{\mathcal{S}}$ is substituted by $\langle y | \mathcal{S} \rangle$.

4.1 Time-out Bisimulation

Thanks to the environment operator θ_L^U , it is possible to express our bisimilarity in a much more succinct way. Indeed, θ_X was defined so that $P \Leftrightarrow_{br}^X Q$ if and only if $\theta_X(P) \Leftrightarrow_{br} \theta_X(Q)$.

► **Definition 13.** A *branching time-out bisimulation* is a symmetric relation $\mathcal{B} \subseteq \mathbb{P} \times \mathbb{P}$ such that, for all $P, Q \in \mathbb{P}$, if $P \mathcal{B} Q$ then

1. if $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{(\alpha)} Q_2$ with $P \mathcal{B} Q_1$ and $P' \mathcal{B} Q_2$
2. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{t} Q_2 \Longrightarrow Q_3 \xrightarrow{t} \dots \Longrightarrow Q_{2r-1} \xrightarrow{t} Q_{2r}$ with $r > 0$, such that $Q_1 \not\mathcal{B} Q_2, \forall i \in [1, r-1], \theta_X(P) \mathcal{B} \theta_X(Q_{2i}) \wedge \mathcal{I}(Q_{2i+1}) \cap (X \cup \{\tau\}) = \emptyset$ and $\theta_X(P') \mathcal{B} \theta_X(Q_{2r})$
3. if $P \not\mathcal{B} Q$ then there is a path $Q \Longrightarrow Q_0 \not\mathcal{B} Q_1$.

Note that in Condition 2 above one also has $P \mathcal{B} Q_1$ and consequently $\mathcal{I}(Q_1) \cap (X \cup \{\tau\}) = \emptyset$. A rooted version of branching time-out bisimulation can be defined in the same vein.

► **Definition 14.** A *rooted branching time-out bisimulation* is a symmetric relation $\mathcal{B} \subseteq \mathbb{P} \times \mathbb{P}$ such that, for all $P, Q \in \mathbb{P}$ such that $P \mathcal{B} Q$,

1. if $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau$ then there is a step $Q \xrightarrow{\alpha} Q'$ such that $P' \Leftrightarrow_{br} Q'$
2. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a step $Q \xrightarrow{t} Q'$ such that $\theta_X(P') \Leftrightarrow_{br} \theta_X(Q')$.

► **Proposition 15.** Let $P, Q \in \mathbb{P}$,

1. $P \Leftrightarrow_{br}^X Q$ (resp. $P \Leftrightarrow_{br}^X Q$) iff there exists a branching time-out bisimulation \mathcal{B} with $P \mathcal{B} Q$ (resp. $(\theta_X(P) \mathcal{B} \theta_X(Q))$),
2. $P \Leftrightarrow_{br}^X Q$ if and only if $\theta_X(P) \Leftrightarrow_{br} \theta_X(Q)$,
3. $P \Leftrightarrow_{br}^r Q$ (resp. $P \Leftrightarrow_{br}^r Q$) iff there exists a rooted branching time-out bisimulation \mathcal{B} with $P \mathcal{B} Q$ (resp. $(\theta_X(P) \mathcal{B} \theta_X(Q))$).

Proof. Note that Proposition 15.2 is a trivial corollary of 15.1.

Let \mathcal{R} be a [generalised rooted] branching reactive bisimulation, let's define $\mathcal{B} := \{(P, Q) \mid \mathcal{R}(P, Q)\} \cup \{(\theta_X(P), \theta_X(Q)) \mid \mathcal{R}(P, X, Q)\}$. \mathcal{B} is a [rooted] branching time-out bisimulation, as proven in [20, Appendix G]. Let \mathcal{B} be a [rooted] branching time-out bisimulation, let's define $\mathcal{R} = \{(P, Q) \mid P \mathcal{B} Q\} \cup \{(P, X, Q) \mid \theta_X(P) \mathcal{B} \theta_X(Q)\}$. \mathcal{R} is a [rooted] generalised branching reactive bisimulation, as proven in [20, Appendix G]. ◀

Time-out bisimulations are very practical as there are no triplets to deal with anymore.

4.2 Congruence

Until now, bisimilarity was only defined between closed expressions, but any relation $\sim \subseteq \mathbb{P} \times \mathbb{P}$ can be extended to $\mathbb{E} \times \mathbb{E}$ in the following way: $E \sim F$ iff $\forall \rho : V \rightarrow \mathbb{P}, E[\rho] \sim F[\rho]$. It can be extended further to substitutions $\rho, \nu \in V \rightarrow \mathbb{E}$ by $\rho \sim \nu$ iff $\text{dom}(\rho) = \text{dom}(\nu)$ and $\forall x \in \text{dom}(\rho), \rho(x) \sim \nu(x)$.

► **Definition 16.** An equivalence $\sim \subseteq \mathbb{E} \times \mathbb{E}$ is a congruence for an n -ary operator f if $P_i \sim Q_i$ for all $i = 0, \dots, n-1$ implies $f(P_0, \dots, P_{n-1}) \sim f(Q_0, \dots, Q_{n-1})$. It is a *lean congruence* if, for all $E \in \mathbb{E}$ and all $\rho, \nu \in V \rightarrow \mathbb{E}$ such that $\rho \sim \nu, E[\rho] \sim E[\nu]$. It is a *full congruence* if

1. it is a congruence for all operators in the language, and
2. for all recursive specifications $\mathcal{S}, \mathcal{S}'$ with $V_{\mathcal{S}} = V_{\mathcal{S}'}$ and $x \in V_{\mathcal{S}}$ such that $\langle x \mid \mathcal{S} \rangle, \langle x \mid \mathcal{S}' \rangle \in \mathbb{P}$, if $\forall y \in V_{\mathcal{S}}, \mathcal{S}_y \sim \mathcal{S}'_y$ then $\langle x \mid \mathcal{S} \rangle \sim \langle x \mid \mathcal{S}' \rangle$.

To show that \sim is a lean congruence it suffices to restrict attention to closed substitutions $\rho, \nu \in V \rightarrow \mathbb{P}$, because the general property will then follow by composition of substitutions. A full congruence is a lean congruence, and a lean congruence is a congruence for all operators in the language, but both implications are strict, as shown in [8].

To show that \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r are full congruences, it is first necessary to prove that \Leftrightarrow_{br} and \Leftrightarrow_{tb} are congruences for some of the operators of CCSP_t^θ .

► **Proposition 17.** \Leftrightarrow_{br} and \Leftrightarrow_{tb} are congruences for action prefixing, parallel composition, abstraction, renaming and the environment operator θ_L^U , for all $L \subseteq U \subseteq A$.

Proof. Let \mathcal{B} be the smallest relation such that, for all $P, Q \in \mathbb{P}$,

- if $P \Leftrightarrow_{br} Q$ then $P \mathcal{B} Q$;
- if $P \mathcal{B} Q$ then, for all $\alpha \in \text{Act}$, $I \subseteq A$, $\mathcal{R} \in A \times A$ and $L \subseteq U \subseteq A$, $\alpha.P \mathcal{B} \alpha.Q$, $\tau_I(P) \mathcal{B} \tau_I(Q)$, $\mathcal{R}(P) \mathcal{B} \mathcal{R}(Q)$ and $\theta_L^U(P) \mathcal{B} \theta_L^U(Q)$;
- if $P_1 \mathcal{B} Q_1$, $P_2 \mathcal{B} Q_2$ and $S \subseteq A$ then $P_1 \parallel_S P_2 \mathcal{B} Q_1 \parallel_S Q_2$.

It suffices to show that \mathcal{B} is a branching time-out bisimulation up to \Leftrightarrow , which implies $\mathcal{B} \subseteq \Leftrightarrow_{br}$. A bisimulation “up to” is a notion introduced by Milner in [17]; it is commonly used when proving congruence properties. The proof uses some lemmas which were obtained in [11]. Details can be found in [20, Appendix H]. A similar proof yields the result for \Leftrightarrow_{tb} . ◀

► **Theorem 18.** \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r are full congruences.

Proof. Let $\mathcal{B} \subseteq \mathbb{P} \times \mathbb{P}$ be the smallest relation such that

- if $P \Leftrightarrow_{br}^r Q$ then $P \mathcal{B} Q$;
- if $P_1 \mathcal{B} Q_1$ and $P_2 \mathcal{B} Q_2$ then $P_1 + P_2 \mathcal{B} Q_1 + Q_2$ and $\forall S \subseteq A$, $P_1 \parallel_S P_2 \mathcal{B} Q_1 \parallel_S Q_2$;
- if $P \mathcal{B} Q$ then $\forall \alpha \in \text{Act}$, $\alpha.P \mathcal{B} \alpha.Q$, $\forall I \subseteq A$, $\tau_I(P) \mathcal{B} \tau_I(Q)$, $\forall \mathcal{R} \subseteq A \times A$, $\mathcal{R}(P) \mathcal{B} \mathcal{R}(Q)$, $\forall L \subseteq U \subseteq A$, $\theta_L^U(P) \mathcal{B} \theta_L^U(Q)$ and $\forall X \subseteq A$, $\psi_X(P) \mathcal{B} \psi_X(Q)$;
- if \mathcal{S} is a recursive specification with $z \in V_S$ and $\rho, \nu \in V \setminus V_S \rightarrow \mathbb{P}$ are substitutions such that $\forall x \in V \setminus V_S$, $\rho(x) \mathcal{B} \nu(x)$, then $\langle z | \mathcal{S} \rangle [\rho] \mathcal{B} \langle z | \mathcal{S} \rangle [\nu]$;
- if \mathcal{S} and \mathcal{S}' are recursive specifications and $x \in V_S = V_{S'}$ with $\langle x | \mathcal{S} \rangle, \langle x | \mathcal{S}' \rangle \in \mathbb{P}$ such that $\forall y \in V_S$, $\mathcal{S}_y \Leftrightarrow_{br}^r \mathcal{S}'_y$, then $\langle x | \mathcal{S} \rangle \mathcal{B} \langle x | \mathcal{S}' \rangle$.

Since $\Leftrightarrow_{br}^r \subseteq \mathcal{B}$, it suffices to prove that \mathcal{B} is a rooted branching time-out bisimulation up to \Leftrightarrow_{br} , as done in [20, Appendix I]. This implies $\mathcal{B} = \Leftrightarrow_{br}^r$ and the definition will then give us that \Leftrightarrow_{br}^r is a lean congruence. Moreover, the last condition of \mathcal{B} adds that it is a full congruence. A similar proof yields the result for \Leftrightarrow_{tb}^r . ◀

5 Axiomatisation

We will provide complete axiomatisations for \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r on various fragments of CCSP_t^θ .

5.1 Recursive Principles

The expression $\langle x | \mathcal{S} \rangle$ is intuitively defined as the x -component of the solution of \mathcal{S} . However, \mathcal{S} could perfectly well have multiple solutions that are not bisimilar to each other. For instance, take $\mathcal{S} = \{x = x\}$; any expression is an x -component of a solution of \mathcal{S} . For our complete axiomatisation, we need to restrict attention to recursive specifications which have a unique solution with respect to our notion of bisimilarity. This property can be decomposed into two principles [1, 4]: the *recursive definition principle* (RDP) states that a system of recursive equations has at least one solution and the *recursive specification principle* (RSP) that it has at most one solution. The latter holds under a condition traditionally called *guardedness*.

36:12 Branching Bisimilarity for Processes with Time-Outs

► **Definition 19.** Let \mathcal{S} be a recursive specification and $\sim \subseteq \mathbb{P} \times \mathbb{P}$, a *solution up to* \sim of \mathcal{S} is a substitution $\rho \in \mathbb{E}^{V_S}$ such that $\rho \sim \mathcal{S}[\rho]$. Here ρ and $\mathcal{S} \in \mathbb{E}^{V_S}$ are seen as V_S -tuples.

In [1, 4] RDP was proven for the classical notion of strong bisimilarity \Leftrightarrow . Since \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r are included in \Leftrightarrow , it holds for both of these relations as well.

► **Proposition 20 (RDP).** *Let \mathcal{S} be a recursive specification. The substitution $\rho : x \mapsto \langle x | \mathcal{S} \rangle$ for all $x \in V_S$ is a solution of \mathcal{S} up to \Leftrightarrow . It is called the default solution of \mathcal{S} .*

An occurrence of a variable x in an expression E is *well-guarded* if x occurs in a subexpression $a.F$ of E , with $a \in A$. Here we do not allow τ and t as guards. An expression E is *well-guarded* if no operator τ_I occurs in E and all free occurrences of variables in E are well-guarded. A recursive specification \mathcal{S} is *manifestly well-guarded* if no operator τ_I occurs in \mathcal{S} and for all $x, y \in V_S$ all occurrences of x in the expression \mathcal{S}_y are well-guarded; it is *well-guarded* if it can be made manifestly well-guarded by repeated substitution of \mathcal{S}_y for y within terms \mathcal{S}_x . A CCSP_t^θ process $P \in \mathbb{P}$ is *guarded* if each recursive specification occurring in E is well-guarded. It is *strongly guarded* if moreover there is no infinite path of τ and t -transitions $P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \xrightarrow{\alpha_3} \dots$ with $\alpha_i \in \{\tau, t\}$ for all $i > 0$, starting in a state P_0 reachable from P .

► **Proposition 21 (RSP).** *Let \mathcal{S} be a well-guarded recursive specification and $\rho, \nu \in \mathbb{E}^{V_S}$. If ρ and ν are solutions of \mathcal{S} up to \Leftrightarrow_{br}^r (or \Leftrightarrow_{tb}^r) then $\rho \Leftrightarrow_{br}^r \nu$ (resp. $\rho \Leftrightarrow_{tb}^r \nu$).*

Proof. Modifying \mathcal{S} by substituting \mathcal{S}_y for y within terms \mathcal{S}_x with $x, y \in V_S$ does not affect the set of its solutions. Hence we can restrict attention to manifestly well-guarded \mathcal{S} .

Thanks to the composition of substitutions, it suffices to prove the proposition when $\rho, \nu \in \mathbb{P}^{V_S}$ and only variables of V_S can occur in \mathcal{S}_x for $x \in V_S$. It suffices to show that the symmetric closure of $\mathcal{B} := \{(H[\mathcal{S}[\rho]], H[\mathcal{S}[\nu]]) \mid H \in \mathbb{E} \text{ is without } \tau_I \text{ operators and with free variables from } V_S \text{ only}\}$ is a rooted branching time-out bisimulation up to \Leftrightarrow_{br} . Here $\mathcal{S}[\rho] \in \mathbb{P}^{V_S}$ is seen as a substitution. Details can be found in [20, Appendix J]. An almost identical strategy can be applied to get RSP for \Leftrightarrow_{tb}^r . ◀

The following lemma, whose proof can be found in [20, Appendix K], states that, when considering strongly guarded processes, eliding a time-out is independent of the set of allowed actions.

► **Lemma 22.** *Let P be a strongly guarded CCSP_t^θ process and $X \subseteq A$. If $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$, $P \xrightarrow{t} P'$ and $\theta_X(P) \Leftrightarrow_{br} \theta_X(P')$ then $\forall Y \subseteq A$, $\mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset \Rightarrow \theta_Y(P) \Leftrightarrow_{br} \theta_Y(P')$.*

Actually, this lemma holds because our restriction of strong guardedness is too strong. Indeed, the equation $x = t.(a + \tau.x)$ has a single solution, but it is not well-guarded. The process $P = \langle x \mid \{x = t.(a + \tau.x)\} \rangle$ is not guarded, yet satisfies $P \xrightarrow{t} P' := a + \tau.P$ and $\theta_\emptyset(P) \Leftrightarrow_{br} \theta_\emptyset(P')$, while $\theta_{\{a\}}(P) \not\Leftrightarrow_{br} \theta_{\{a\}}(P')$. Even if we write P as $\tau_{\{b\}}(\langle x \mid \{x = t.(a + b.x)\} \rangle)$ it fails to be strongly guarded. This restriction was kept because being more precise is very challenging. For instance, the equation $x = t.(a + \tau.x) + t.a$ has multiple solutions: the default one, $\langle x \mid \{x = t.(a + \tau.x) + t.a + t.(a + t.b)\} \rangle$ and others. Notice that adding a branch $t.a$ to an equation with one solution can lead to it having multiple ones. Intuitively, there are situations where time-out contraction enables to hide the existence of other time-outs. Characterising these situations requires the use of semantic conditions that are difficult to verify, thus, making them undesirable. Moreover, applying the Pohlmann encoding to the processes in order to, then, use the axiomatisation of \Leftrightarrow_{tb}^r leads to the similar complications. This limitation deserves to be studied properly because it will appear for all bisimilarities authorising time-out contraction.

5.2 Axioms and Soundness

The set of axioms provided is composed of the axiomatisation of \Leftrightarrow_r [11], together with three branching axioms. The *branching axiom* is well-known since it is used in the axiomatisation of rooted branching bisimilarity [13]. The *t-branching axiom* and the *τ /t-branching axiom* are newly introduced; they are the adaptation of the branching axiom to time-out contraction.

■ **Table 3** Axiomatisation of \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r .

$x + (y + z) = (x + y) + z$	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	$\mathcal{R}(x + y) = \mathcal{R}(x) + \mathcal{R}(y)$
$x + y = y + x$	$\tau_I(\alpha.x) = \alpha.\tau_I(x)$ if $\alpha \notin I$	$\mathcal{R}(\tau.x) = \tau.\mathcal{R}(x)$
$x + x = 0$	$\tau_I(\alpha.x) = \tau.\tau_I(x)$ if $\alpha \in I$	$\mathcal{R}(t.x) = t.\mathcal{R}(x)$
$x + 0 = x$		$\mathcal{R}(a.x) = \sum_{\{b \mid \mathcal{R}(a,b)\}} b.\mathcal{R}(x)$
Expansion Theorem: if $P = \sum_{i \in I} \alpha_i.P_i$ and $Q = \sum_{j \in J} \beta_j.Q_j$ then		
$P \parallel_S Q = \sum_{i \in I, \alpha_i \notin S} (\alpha_i.P_i \parallel_S Q) + \sum_{j \in J, \beta_j \notin S} (P \parallel_S \beta_j.Q_j) + \sum_{i \in I, j \in J, \alpha_i = \beta_j \in S} \alpha_i.(P_i \parallel_S Q_j)$		
$\alpha.(\tau.(x + y) + x) = \alpha.(x + y)$ (Branching Axiom)		
$\alpha.(t.(x + \sum_{i \in I} t.y_i) + x) = \alpha.(x + \sum_{i \in I} t.y_i)$ (t-Branching Axiom)		
$\alpha.(\tau.(x + y) + t.(x + y) + x) = \alpha.(x + y)$ (τ/t-Branching Axiom)		
$\langle x \mid S \rangle = \langle S_x \mid S \rangle$ (RDP)	$S \Rightarrow x = \langle x \mid S \rangle$ with S well-guarded (RSP)	
$\theta_L^U(\sum_{i \in I} \alpha_i.x_i) = \sum_{i \in I} \alpha_i.x_i$	$(\forall i \in I, \alpha_i \notin L \cup \{\tau\})$	
$\theta_L^U(x + \alpha.y + \beta.z) = \theta_L^U(x + \alpha.y)$	$(\alpha \in L \cup \{\tau\} \wedge \beta \notin U \cup \{\tau\})$	
$\theta_L^U(x + \alpha.y + \beta.z) = \theta_L^U(x + \alpha.y) + \theta_L^U(\beta.z)$	$(\alpha \in L \cup \{\tau\} \wedge \beta \in U \cup \{\tau\})$	
$\theta_L^U(\alpha.x) = \alpha.x$	$(\alpha \neq \tau)$	
$\theta_L^U(\tau.x) = \tau.\theta_L^U(x)$		
$\psi_X(x + \alpha.y) = \psi_X(x) + \alpha.y$	$(\alpha \notin X \cup \{\tau, t\})$	
$\psi_X(x + \alpha.y + t.z) = \psi_X(x + \alpha.y)$	$(\alpha \in X \cup \{\tau\})$	
$\psi_X(x + \alpha.y + \beta.z) = \psi_X(x + \alpha.y) + \beta.z$	$(\alpha, \beta \in X \cup \{\tau\})$	
$\psi_X(\alpha.x) = \alpha.x$	$(\alpha \neq t)$	
$\psi_X(\sum_{i \in I} t.y_i) = \sum_{i \in I} t.\psi_X(y_i)$		
$(\forall X \subseteq A, \psi_X(x) = \psi_X(y)) \Rightarrow x = y$ (Reactive Approximation Axiom)		

Let Ax^∞ be the set of all axioms in the first two rectangles in Table 3 and $Ax := Ax^\infty \setminus \{\text{RDP}, \text{RSP}\}$. Let Ax_r^∞ be the set of all axioms in Table 3 except the τ /t-branching one and $Ax_r := Ax_r^\infty \setminus \{\text{RDP}, \text{RSP}\}$. The τ /t-branching axiom is removed from Ax_r^∞ because the law $L\tau: \tau.x + t.y = \tau.x$ can be derived from the reactive approximation axiom [11], and applying $L\tau$ to the branching axiom yields the τ /t-branching axiom, thus making it redundant.

► **Proposition 23.** *Let P, Q be two $CCSP_t^\theta$ processes.*

- *If $Ax^\infty \vdash P = Q$ then $P \Leftrightarrow_{tb}^r Q$.*
- *If $Ax_r^\infty \vdash P = Q$ then $P \Leftrightarrow_{br}^r Q$.*

Proof. Since \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r are congruences, it suffices to prove that each axiom is sound, meaning that replacing, in each axiom, $=$ by the desired bisimilarity and each variable by a process produces a true statement. Most of these axioms were proven to be sound for the classical notion \Leftrightarrow of strong bisimilarity [17] in [11]. Thus, since both \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r are included in \Leftrightarrow , most of them are sound for \Leftrightarrow_{br}^r and \Leftrightarrow_{tb}^r .

36:14 Branching Bisimilarity for Processes with Time-Outs

Only the branching axioms, RSP and the reactive approximation axiom remain to be proven sound. The soundness of the branching axioms is trivial and the soundness of RSP is exactly Proposition 21. For the reactive approximation axiom, it suffices to show that $\mathcal{B} := \Leftrightarrow_{br}^r \cup \{(P, Q), (Q, P) \mid \forall X \subseteq A, \psi_X(P) \Leftrightarrow_{br}^r \psi_X(Q)\}$ is a rooted branching time-out bisimulation, as done in [20, Appendix L]. ◀

5.3 Completeness

A well-known feature of most process algebras is that the standard collection of axioms allows one to bring any guarded process expression in the following normal form [1, 4].

► **Definition 24.** Let P be a guarded $CCSP_t^\theta$ process. The *head-normal form* of P is $\hat{P} := \sum_{\{(\alpha, Q) \mid P \xrightarrow{\alpha} Q\}} \alpha.Q$.

In [11], it is proven that the axiomatisation of \Leftrightarrow_r enables one to equate any guarded process with its head-normal form (using a definition of guardedness that is more liberal than the one employed here, with τ and t allowed as guards). Since the axiomatisation of \Leftrightarrow_r is included in Ax^∞ and Ax_r^∞ , this yields the property for them as well.

► **Lemma 25.** Let P be a guarded $CCSP_t^\theta$ process. Then $Ax^\infty \vdash P = \hat{P}$ and $Ax_r^\infty \vdash P = \hat{P}$. Moreover, Ax or Ax_r are sufficient if P is recursion-free.

This lemma is used extensively in the proof of the following completeness results.

► **Proposition 26.** Let P, Q be two recursion-free $CCSP_t^\theta$ processes. If $P \Leftrightarrow_{br} Q$ (resp. $P \Leftrightarrow_{tb} Q$) then, for all $\alpha \in Act$, $Ax_r \vdash \alpha.\hat{P} = \alpha.\hat{Q}$ (resp. $Ax \vdash \alpha.\hat{P} = \alpha.\hat{Q}$).

Proof. The *depth* $d(p)$ of a process P is the length of the longest path starting from P . Note that it is properly defined for recursion-free processes only. The proof proceeds by induction on $\max(d(P), d(Q))$. The technique is fairly standard and the details can be found in [20, Appendix M]. ◀

► **Theorem 27.** Let P, Q be two recursion-free $CCSP_t^\theta$ processes. If $P \Leftrightarrow_{br}^r Q$ (resp. $P \Leftrightarrow_{tb}^r Q$) then $Ax_r \vdash P = Q$ (resp. $Ax \vdash P = Q$).

Proof. It suffices to express both processes in their head-normal form and then to equate each pair of matching branches using Proposition 26. Details are in [20, Appendix M]. ◀

The following theorem lifts this result for \Leftrightarrow_{tb}^r from finite (recursion-free) processes to arbitrary (infinite) ones, subject to the restriction of strong guardedness.

► **Theorem 28.** Let P, Q be strongly guarded $CCSP_t^\theta$ processes. If $P \Leftrightarrow_{tb}^r Q$ then $Ax^\infty \vdash P = Q$.

Proof. A well-known technique called *equation merging* can be applied. Details can be found in [20, Appendix N]. ◀

5.4 Canonical Representative

Unfortunately, equation merging does not work on reactive bisimulations [11]. Thus, another technique is used [14, 16], called *canonical representatives*. The idea is to build the simplest process for each equivalence class of \simeq_{br}^r and use them as intermediary to equate processes.

Let us denote with \mathbb{P}^g the strongly guarded fragment of \mathbb{P} . For all $P \in \mathbb{P}^g$, $[P] := \{Q \in \mathbb{P}^g \mid P \simeq_{br} Q\}$ is the \simeq_{br} -equivalence class of P . $[\mathbb{P}^g]$ denotes the set of all \simeq_{br} -equivalence classes. Using the axiom of choice, a choice function $\chi : [\mathbb{P}^g] \rightarrow \mathbb{P}^g$ can be defined such that $\forall R \in [\mathbb{P}^g], \chi(R) \in R$. A transition relation can be defined between \simeq_{br} -equivalence classes:

$$\begin{aligned} \forall \alpha \in A_\tau, (R \xrightarrow{\alpha} R' \Leftrightarrow \chi(R) \Longrightarrow P_1 \xrightarrow{\alpha} P_2 \wedge P_1 \in R \wedge P_2 \in R' \wedge (\alpha \in A \vee R \neq R')) \\ R \xrightarrow{t} R' \Leftrightarrow \exists X \subseteq A, r > 0, \chi(R) \Longrightarrow P_1 \xrightarrow{t} P_2 \Longrightarrow P_3 \xrightarrow{t} \dots \Longrightarrow P_{2r-1} \xrightarrow{t} P_{2r} \\ \wedge \forall i \in [0, r-1], \theta_X(P_{2i}) \in [\theta_X(\chi(R))] \wedge \mathcal{I}(P_{2i+1}) \cap (X \cup \{\tau\}) = \emptyset \\ \wedge P_1 \in R \wedge P_{2r} \in R' \wedge [\theta_X(\chi(R))] \neq [\theta_X(\chi(R'))] \end{aligned}$$

All bisimulations can be extended to \simeq_{br} -equivalence classes. It suffices to consider the set of states $\mathbb{P}^g \uplus [\mathbb{P}^g] \uplus \{\theta_X([P]) \mid X \subseteq A \wedge P \in \mathbb{P}^g\}$.

► **Proposition 29.** *Let $P \in \mathbb{P}^g$, $P \simeq_{br} [P]$.*

Proof. It suffices to prove that $\mathcal{B} := \{(P, [P]), ([P], P) \mid P \in \mathbb{P}^g\}$ is a branching time-out bisimulation up to \simeq_{br} . Details can be found in [20, Appendix O]. ◀

► **Definition 30.** Let $P, Q \in \mathbb{P}^g$, the *canonical representative* of P and Q is a recursive specification \mathcal{S} such that $V_S := \{x_P, x_Q\} \cup \{x_R \mid R \in \bigcup_{P' \in \text{Reach}(P) \cup \text{Reach}(Q)} \text{Reach}([P'])\}$, and $\forall R \in \bigcup_{P' \in \text{Reach}(P) \cup \text{Reach}(Q)} \text{Reach}([P'])$,

$$\mathcal{S}_{x_P} := \sum_{\{(\alpha, P') \mid P \xrightarrow{\alpha} P'\}} \alpha.x_{[P']} ; \mathcal{S}_{x_Q} := \sum_{\{(\alpha, Q') \mid Q \xrightarrow{\alpha} Q'\}} \alpha.x_{[Q']} \text{ and } \mathcal{S}_{x_R} := \sum_{\{(\alpha, R') \mid R \xrightarrow{\alpha} R'\}} \alpha.x_{R'}$$

The canonical representative is well-defined since P, Q , as well as processes $[P'] \in [\mathbb{P}^g]$ are finitely branching [11]. Additionally, $\bigcup_{P' \in \text{Reach}(P) \cup \text{Reach}(Q)} \text{Reach}([P'])$ is countable. Moreover, \mathcal{S} is strongly guarded. Furthermore, by construction $\langle x_R \mid \mathcal{S} \rangle \simeq R$ for all $R \in \bigcup_{P' \in \text{Reach}(P) \cup \text{Reach}(Q)} \text{Reach}([P'])$.

► **Proposition 31.** *Let $P, Q \in \mathbb{P}^g$ and \mathcal{S} be the canonical representative of P and Q , $Ax_r^\infty \vdash P = \langle x_P \mid \mathcal{S} \rangle$.*

Proof. It suffices to show that P and $\langle x_P \mid \mathcal{S} \rangle$ are y_P -components of solutions of $\{y_{P^\dagger} = \sum_{\{(\alpha, P^\dagger) \mid P^\dagger \xrightarrow{\alpha} P^\dagger\}} \alpha.y_{P^\dagger} \mid P^\dagger \in \text{Reach}(P)\}$. Details can be found in [20, Appendix P]. ◀

► **Theorem 32.** *Let $P, Q \in \mathbb{P}^g$, if $P \simeq_{br}^r Q$ then $Ax_r^\infty \vdash P = Q$.*

Proof. It suffices to equate $\langle x_P \mid \mathcal{S} \rangle$ and $\langle x_Q \mid \mathcal{S} \rangle$ using RDP and the reactive approximation axiom. Details can be found in [20, Appendix P]. ◀

Conclusion

This paper defined a form of branching bisimilarity for processes with time-out transitions, and provided a modal characterisation, congruence results, and a complete axiomatisation for strongly guarded processes. The restriction to strongly guarded processes is rather

severe; it rules out processes that may engage in an infinite sequence of time-out transitions, interspersed with τ s. Relaxing this restriction is a suitable topic for further work. Another task is to combine this work with the ideas behind *justness* [12], a weaker form of fairness that allows the formulation and derivation of useful liveness properties. In a setting with time-outs, justness would demand that once a parallel component reaches a state in which a time-out transition is enabled, it cannot stay in that state forever after.

As an example of the use of branching reactive bisimulation, one could verify the correctness of a non-trivial system, such as Peterson’s mutual exclusion protocol, as modelled in [10]. There it was argued that a similar model without time-out transitions is not possible. The model from [10] features eight visible actions of entering or leaving the critical or non-critical section of process A or B. Abstracting from all actions pertaining to process B yields a protocol that only deals with process A, and a correctness claim could be validated by showing it branching reactive bisimilar with a simple specification of the intended behaviour of A that would apply when B were not around. Although doing such a verification is entirely feasible, for now, it can not be achieved by algebraic means, using our complete axiomatisation. The reason is that abstraction from process B yields infinite sequences of unobservable actions, which are currently not covered by our work.

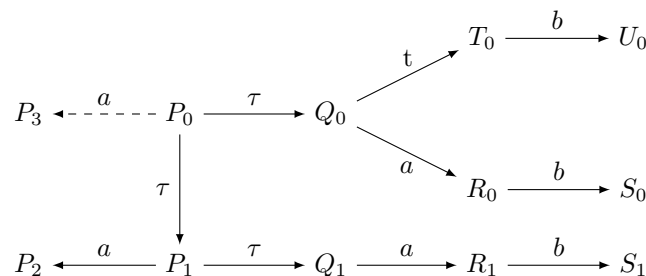
References

- 1 Jos C.M. Baeten and W. Peter Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990. doi:10.1017/CB09780511624193.
- 2 Stephen D. Brookes, Tony (C.A.R.) Hoare, and Bill (A.W.) Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984. doi:10.1145/828.833.
- 3 Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mCRL2 toolset for analysing concurrent systems—improvements in expressivity and usability. In Tomáš Vojnar and Lijun Zhang, editors, Proc. 25th International Conference on *Tools and Algorithms for the Construction and Analysis of Systems*, TACAS’19, held as part of the *European Joint Conferences on Theory and Practice of Software*, ETAPS’19, Prague, Czech Republic, volume 11428 of LNCS, pages 21–39. Springer, 2019. doi:10.1007/978-3-030-17465-1_2.
- 4 Wan J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series. Springer, 2000. doi:10.1007/978-3-662-04293-9.
- 5 Wan J. Fokkink, Rob J. van Glabbeek, and Bas Luttik. Divide and congruence III: From decomposition of modal formulas to preservation of stability and divergence. *Information and Computation*, 268:104435, 2019. doi:10.1016/j.ic.2019.104435.
- 6 Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2010: A toolbox for the construction and analysis of distributed processes. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, Proceedings *Tools and Algorithms for the Construction and Analysis of Systems*, TACAS ’11, volume 6605 of LNCS, pages 372–387. Springer, 2011. doi:10.1007/978-3-642-19835-9_33.
- 7 Rob J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract). In E. Best, editor, Proceedings *CONCUR’93*, 4th International Conference on *Concurrency Theory*, Hildesheim, Germany, August 1993, volume 715 of LNCS, pages 66–81. Springer, 1993. doi:10.1007/3-540-57208-2_6.
- 8 Rob J. van Glabbeek. Lean and full congruence formats for recursion. In Proceedings 32nd Annual ACM/IEEE Symposium on *Logic in Computer Science*, LICS’17, Reykjavik, Iceland, June 2017. IEEE Computer Society Press, 2017. doi:10.1109/LICS.2017.8005142.
- 9 Rob J. van Glabbeek. Failure trace semantics for a process algebra with time-outs. *Logical Methods in Computer Science*, 17(2), 2021. doi:10.23638/LMCS-17(2:11)2021.
- 10 Rob J. van Glabbeek. Modelling mutual exclusion in a process algebra with time-outs. *Information and Computation*, 294, 2023. doi:10.1016/j.ic.2023.105079.

- 11 Rob J. van Glabbeek. Reactive bisimulation semantics for a process algebra with timeouts. *Acta Informatica*, 60(1):11–57, 2023. doi:10.1007/s00236-022-00417-1.
- 12 Rob J. van Glabbeek and Peter Höfner. Progress, justness and fairness. *ACM Computing Surveys*, 52(4), August 2019. doi:10.1145/3329125.
- 13 Rob J. van Glabbeek and W. Peter Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556.
- 14 Clemens Grabmayer and Wan J. Fokkink. A complete proof system for 1-free regular expressions modulo bisimilarity. In H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller, editors, Proc. 35th Annual ACM/IEEE Symposium on *Logic in Computer Science*, LICS’20, pages 465–478. ACM, 2020. doi:10.1145/3373718.3394744.
- 15 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985. doi:10.1145/2455.2460.
- 16 Xinxin Liu and Tingting Yu. Canonical solutions to recursive equations and completeness of equational axiomatisations. In I. Konnov and L. Kovacs, editors, Proceedings 31st International Conference on *Concurrency Theory* (CONCUR 2020), volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.35.
- 17 Robin Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 19, pages 1201–1242. Elsevier Science Publishers B.V. (North-Holland), 1990. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, doi:10.1007/3-540-10235-3.
- 18 Ernst-Ruediger Olderog and Tony (C.A.R.) Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986. doi:10.1007/BF00268075.
- 19 Maximilian Pohlmann. Reducing strong reactive bisimilarity to strong bisimilarity. Bachelor’s thesis, Technische Universität Berlin, 2021. URL: <https://maxpohlmann.github.io/Reducing-Reactive-to-Strong-Bisimilarity/thesis.pdf>.
- 20 Gaspard Reghem and Rob J. van Glabbeek. Branching bisimilarity for processes with time-outs. technical report, full version of the present paper, 2024. arXiv:2408.10117.
- 21 Gaspard Reghem and Rob J. van Glabbeek. Concrete branching bisimilarity for processes with time-outs, 2024. URL: <https://theory.stanford.edu/~rvg/abstracts.html#167>.

A Examples

Scope of the First Clause of Definition 1



■ **Figure 2** Counter-Example to a Naive Clause 1.a.

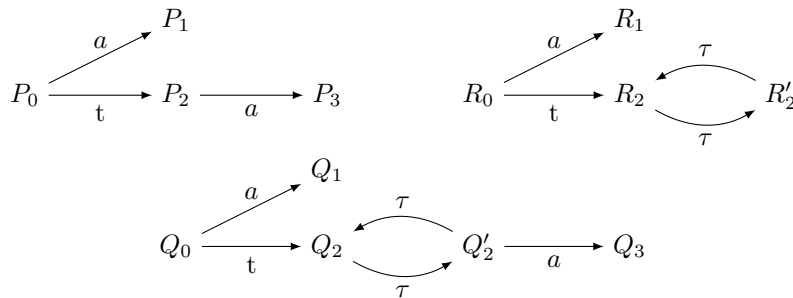
In Figure 2, the process $a.0 + \tau.(t.b.0 + a.b.0) + \tau.(\tau.a.b.0 + a.0)$ is represented as an LTS. Let $A := \{a, b\}$. Removing the dashed a -transition generates the process $\tau.(t.b.0 + a.b.0) + \tau.(\tau.a.b.0 + a.0)$.

First, we are going to show that these two processes are not branching reactive bisimilar. Let's try to build a branching reactive bisimulation between them. The only way to match the dashed a -transition of $a.0 + \tau.(t.0 + a.b.0) + \tau.(\tau.a.b.0 + a.0)$ is by the a -transition between P_1 and P_2 , because all other a -transitions are followed by a b -transition. This requires to elide the τ -transition between P_0 and P_1 , who must be branching reactive bisimilar. Since $P_0 \stackrel{\text{br}}{\leftrightarrow} P_1$, when considering the τ -transition between P_0 and Q_0 , Q_0 has to be branching reactive bisimilar to P_1 or Q_1 . Now, the a -transition between Q_0 and R_0 has to be matched by the a -transition between Q_1 and R_1 because of the following b -transition. This implies $Q_0 \stackrel{\text{br}}{\leftrightarrow} Q_1$, thus, $Q_0 \stackrel{\emptyset}{\leftrightarrow}_{\text{br}} Q_1$. One has $\mathcal{I}(Q_0) \cap (\emptyset \cup \{\tau\}) = \emptyset$ and $Q_0 \xrightarrow{t} T_0$, i.e., when the environment temporary allows no visible actions, Q_0 can time-out into a state in which b is possible. This behaviour cannot be matched by Q_1 – a contradiction.

Now, consider the alternative to Definition 1 in which the first clause has been changed to 1. a. if $P \xrightarrow{\tau} P'$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{(\tau)} Q_2$ with $\mathcal{R}(P, Q_1)$ and $\mathcal{R}(P', Q_2)$. In other words, the scope of the first clause is restricted to τ -transitions. This modification enables building a bisimulation between the two processes. Indeed, the dashed a -transition is only considered when the environment allows a . Thus, it is sufficient to get $P_0 \stackrel{A}{\leftrightarrow}_{\text{br}} P_1$ and $P_0 \stackrel{\{a\}}{\leftrightarrow}_{\text{br}} P_1$ and not $P_0 \stackrel{\text{br}}{\leftrightarrow} P_1$ anymore. Therefore, it is sufficient to match Q_0 and Q_1 in environments allowing a . As a result, the outgoing time-out transition of Q_0 is never considered when matching Q_0 with Q_1 , solving our previous issue. Once this observation is made, building the bisimulation is trivial.

Finally, place both processes in the context $_ \parallel_{\{a\}} (\tau.0 + a.0)$. It behaves like a one-way switch enabling to block all a -transitions forever as soon as the τ -transition is performed. Let's try to build a branching reactive bisimulation between the two processes. Following the same reasoning as before, it is necessary to get $P_0 \parallel_{\{a\}} (\tau.0 + a.0) \stackrel{A}{\leftrightarrow}_{\text{br}} P_1 \parallel_{\{a\}} (\tau.0 + a.0)$ because of the dashed a -transition, and then $Q_0 \parallel_{\{a\}} (\tau.0 + a.0) \stackrel{A}{\leftrightarrow}_{\text{br}} Q_1 \parallel_{\{a\}} (\tau.0 + a.0)$ because of the a -transition between Q_0 and R_0 . Note that $Q_0 \parallel_{\{a\}} (\tau.0 + a.0) \xrightarrow{\tau} Q_0 \parallel_{\{a\}} 0 \xrightarrow{t} T_0 \parallel_{\{a\}} 0 \xrightarrow{b} U_0 \parallel_{\{a\}} 0$ and $\mathcal{I}(Q_0 \parallel_{\{a\}} 0) \cap (A \cup \{\tau\}) = \emptyset$. As before, $Q_0 \parallel_{\{a\}} (\tau.0 + a.0)$ can time-out into a state in which b is executable, whereas this behaviour is impossible in $Q_1 \parallel_{\{a\}} (\tau.0 + a.0)$. As a result, restricting the scope of the first clause of Definition 1 to τ -transitions prevents $\stackrel{\text{br}}{\leftrightarrow}$ from being a congruence for parallel composition.

Necessity of the Stability Respecting Clause



■ **Figure 3** Counter-Example to the Absence of a Stability Respecting Clause.

In Figure 3, three processes are represented as LTSs. Take $A := \{a\}$. According to Definition 1, $\neg(P_0 \stackrel{\text{br}}{\leftrightarrow} Q_0)$ and $Q_0 \stackrel{A}{\leftrightarrow}_{\text{br}} R_0$.

Let's try to build a branching reactive bisimulation between the top-left and bottom processes. Matching the time-out between Q_0 and Q_2 implies that $Q_2 \stackrel{\emptyset}{\leftrightarrow}_{br} P_0$ or $Q_2 \stackrel{\emptyset}{\leftrightarrow}_{br} P_2$. However, $P_0 \not\stackrel{\tau}{\rightarrow}$ and $P_2 \not\stackrel{\tau}{\rightarrow}$, thus, there should be a path $Q_2 \Longrightarrow Q'_2 \stackrel{\tau}{\rightarrow}$, but this is not the case.

The symmetric closure of

$$\mathcal{R} := \{(Q_0, R_0), (Q_1, R_1), (Q_2, \emptyset, R_2), (Q'_2, \emptyset, R'_2)\} \cup \{(Q_0, X, R_0), (Q_1, X, R_1) \mid X \subseteq A\}$$

is a branching reactive bisimulation. The a -transition between Q'_2 and Q_3 does not have to be matched since Q'_2 is considered only when the environment disallows a .

Now, suppose that the stability respecting condition is removed from Definition 1. As a result, a branching reactive bisimulation can be built between the top-left and bottom processes. The symmetric closure of

$$\begin{aligned} \mathcal{R}' := & \{(P_0, Q_0), (P_1, Q_1), (P_2, Q_2), (P_2, Q'_2), (P_3, Q_3)\} \\ & \cup \{(P_0, X, Q_0), (P_1, X, Q_1), (P_2, X, Q_2), (P_2, X, Q'_2), (P_3, X, Q_3) \mid X \subseteq A\} \end{aligned}$$

would be a branching reactive bisimulation. Moreover, \mathcal{R} would still be a branching reactive bisimulation, since Definition 1 has merely been weakened. Therefore, according to the modified Definition 1, $P_0 \stackrel{\emptyset}{\leftrightarrow}_{br} Q_0$ and $Q_0 \stackrel{\emptyset}{\leftrightarrow}_{br} R_0$. However, when trying to construct a branching reactive bisimulation between P_0 and R_0 , because of the time-out transition, R_2 has to be matched to P_0 or P_2 and no a -transition is reachable from R_2 ; therefore, $\neg(P_0 \stackrel{\emptyset}{\leftrightarrow}_{br} R_0)$. As a result, removing the stability respecting clause from Definition 1 prevents $\stackrel{\emptyset}{\leftrightarrow}_{br}$ from being an equivalence relation.

B Concrete Time-out Version

Before studying $\stackrel{\emptyset}{\leftrightarrow}_{br}$, we looked at another version which is not eliding any time-out transitions. More formally, it is defined by replacing Clause 2.d of Definition 1 by

2. d. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \stackrel{\tau}{\rightarrow} P'$ then there exists a path $Q \Longrightarrow Q_1 \stackrel{\tau}{\rightarrow} Q_2$ with $\mathcal{R}(P', X, Q_2)$.

It is not necessary to require to match with an executable time-out (i.e. $\mathcal{I}(Q_1) \cap (X \cup \{\tau\}) = \emptyset$) since this is implied by the other clauses. It is also implied that $\mathcal{R}(P, X, Q_1)$ in the above clause. This bisimilarity has properties similar to $\stackrel{\emptyset}{\leftrightarrow}_{br}$, to be recapped below. No proof will be provided here since they rely on the same strategies and are actually simpler because of the absence of time-out omission. However, a technical report [21] is available. In the remainder of this appendix, $\stackrel{c}{\leftrightarrow}_{br}$ stands for the concrete time-out version.

The stuttering lemma (Lemma 3) still holds and $\stackrel{c}{\leftrightarrow}_{br}$ and $(\stackrel{c}{\leftrightarrow}_{br}^X)_{X \subseteq A}$ are still equivalence relations (Proposition 4). The rooted version of $\stackrel{c}{\leftrightarrow}_{br}$ is exactly Definition 5 and $\stackrel{r}{\leftrightarrow}_{br}$ and $(\stackrel{r}{\leftrightarrow}_{br}^X)_{X \subseteq A}$ are still equivalence relations (Proposition 6). The Pohlmann encoding (Table 4) is simplified as the rooted variants are no longer needed: $P \stackrel{cr}{\leftrightarrow}_{br} Q \Leftrightarrow \vartheta(P) \stackrel{c}{\leftrightarrow}_b \vartheta(Q)$. If $\stackrel{c}{\leftrightarrow}_b$ stands for the classical stability respecting branching bisimulation [13, 7], $P \stackrel{c}{\leftrightarrow}_{br} Q \Leftrightarrow \vartheta(P) \stackrel{c}{\leftrightarrow}_b \vartheta(Q)$; $P \stackrel{cX}{\leftrightarrow}_{br} Q \Leftrightarrow \vartheta_X(P) \stackrel{c}{\leftrightarrow}_b \vartheta_X(Q)$; $P \stackrel{cr}{\leftrightarrow}_{br} Q \Leftrightarrow \vartheta(P) \stackrel{c}{\leftrightarrow}_b \vartheta(Q)$ and $P \stackrel{crX}{\leftrightarrow}_{br} Q \Leftrightarrow \vartheta_X(P) \stackrel{c}{\leftrightarrow}_b \vartheta_X(Q)$.

The generalised definition of $\stackrel{c}{\leftrightarrow}_{br}$ can be obtained by replacing Clause 1.b. and 2.c. in Definition 33 by

1. b. If $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \stackrel{\tau}{\rightarrow} P'$ then there exists a path $Q \Longrightarrow Q_1 \stackrel{\tau}{\rightarrow} Q_2$ with $\mathcal{R}(P', X, Q_2)$
2. c. If $\mathcal{I}(P) \cap (X \cup Y \cup \{\tau\}) = \emptyset$ and $P \stackrel{\tau}{\rightarrow} P'$ then there exists a path $Q \Longrightarrow Q_1 \stackrel{\tau}{\rightarrow} Q_2$ with $\mathcal{R}(P', Y, Q_2)$

The rooted generalised version is exactly Definition 34 and they induce the same bisimilarities as the previous definitions (Proposition 35). In the modal characterisation, $X\varphi$ is not useful anymore, nor $\varphi\langle\varepsilon_X\rangle\varphi'$. Replacing the fifth induction rule of \mathbb{L}_b by $\langle\varepsilon\rangle\langle t_X\rangle\varphi$ yields the counterpart of Theorem 11.

The corresponding time-out bisimulation can be obtained by replacing Clause 2. of Definition 13 by

2. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there exists a path $P \Longrightarrow P_1 \xrightarrow{t} P_2$ with $\theta_X(P') \mathcal{B} \theta_X(Q_2)$.

The rooted time-out bisimulation is exactly Definition 14 and they agree with the previous definitions (Proposition 15). \Leftrightarrow_{br}^c is a congruence for prefixing, parallel composition, abstraction, renaming and the operator θ_L^U (Proposition 17). $\Leftrightarrow_{br}^{cr}$ is a full congruence (Theorem 18).

As $\Leftrightarrow_{br}^{cr} \subseteq \Leftrightarrow$, RDP holds for $\Leftrightarrow_{br}^{cr}$. The definition of well-guarded recursion can be weakened by allowing t as a guard and RSP holds for $\Leftrightarrow_{br}^{cr}$ on processes that are guarded in this sense. Lemma 22 is not useful anymore since time-out omissions are not considered. The set of all axioms of Table 3 except the t -branching and τ/t -branching ones is a complete axiomatisation of \Leftrightarrow_{br}^c (Theorem 32). Moreover, to obtain the complete axiomatisation of \Leftrightarrow_{br}^c on recursion-free processes, it suffices to remove RDP and RSP.

C Generalised branching reactive bisimulation

The second clause of Definition 1 is quite tedious to check; thus, an equivalent definition of the bisimilarity would be useful. Actually, it is possible to define the exact same notion in a more general way at the cost of some clear motivations.

► **Definition 33.** A *generalised branching reactive bisimulation* is a symmetric relation $\mathcal{R} \subseteq (\mathbb{P} \times \mathbb{P}) \cup (\mathbb{P} \times \mathcal{P}(A) \times \mathbb{P})$ such that, for all $P, Q \in \mathbb{P}$ and $X \subseteq A$,

1. if $\mathcal{R}(P, Q)$
 - a. if $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{(\alpha)} Q_2$ with $\mathcal{R}(P, Q_1)$ and $\mathcal{R}(P', Q_2)$,
 - b. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a path $Q = Q_0 \Longrightarrow Q_1 \xrightarrow{t} Q_2 \Longrightarrow Q_3 \xrightarrow{t} \dots \Longrightarrow Q_{2r-1} \xrightarrow{(t)} Q_{2r}$ with $r > 0$, such that $Q_1 \not\xrightarrow{\tau}, \forall i \in [1, r-1]$, $\mathcal{R}(P, X, Q_{2i}) \wedge \mathcal{I}(Q_{2i+1}) \cap (X \cup \{\tau\}) = \emptyset$ and $\mathcal{R}(P', X, Q_{2r})$,
 - c. if $P \not\xrightarrow{\tau}$ then there exists a path $Q \Longrightarrow Q_0 \not\xrightarrow{\tau}$;
2. if $\mathcal{R}(P, X, Q)$
 - a. if $P \xrightarrow{\tau} P'$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{(\tau)} Q_2$ with $\mathcal{R}(P, X, Q_1)$ and $\mathcal{R}(P', X, Q_2)$,
 - b. if $P \xrightarrow{a} P'$ with $a \in X \vee \mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ then there is a path $Q \Longrightarrow Q_1 \xrightarrow{a} Q_2$ with $\mathcal{R}(P, X, Q_1)$ and $\mathcal{R}(P', Q_2)$,
 - c. if $\mathcal{I}(P) \cap ((X \cup Y) \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a path $Q = Q_0 \Longrightarrow Q_1 \xrightarrow{t} Q_2 \Longrightarrow Q_3 \xrightarrow{t} \dots \Longrightarrow Q_{2r-1} \xrightarrow{(t)} Q_{2r}$ with $r > 0$, such that $Q_1 \not\xrightarrow{\tau}, \forall i \in [1, r-1]$, $\mathcal{R}(P, Y, Q_{2i}) \wedge \mathcal{I}(Q_{2i+1}) \cap (Y \cup \{\tau\}) = \emptyset$ and $\mathcal{R}(P', Y, Q_{2r})$,
 - d. if $P \not\xrightarrow{\tau}$ then there is a path $Q \Longrightarrow Q_0 \not\xrightarrow{\tau}$.

The strong point of the generalised definitions is the restriction on the use of triplets, making use of them only after performing a time-out. A generalised version of rooted branching reactive bisimulation can be defined in a similar fashion.

► **Definition 34.** A *generalised rooted branching reactive bisimulation* is a symmetric relation $\mathcal{R} \subseteq (\mathbb{P} \times \mathbb{P}) \cup (\mathbb{P} \times \mathcal{P}(A) \times \mathbb{P})$ such that, for all $P, Q \in \mathbb{P}$ and $X \subseteq A$,

1. if $\mathcal{R}(P, Q)$
 - a. if $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau$ then there is a transition $Q \xrightarrow{\alpha} Q'$ such that $P' \leftrightarrow_{br} Q'$,
 - b. if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a transition $Q \xrightarrow{t} Q'$ with $P' \leftrightarrow_{br}^X Q'$,
2. if $\mathcal{R}(P, X, Q)$
 - a. if $P \xrightarrow{\tau} P'$ then there is a transition $Q \xrightarrow{\tau} Q'$ such that $P' \leftrightarrow_{br}^X Q'$,
 - b. if $P \xrightarrow{a} P'$ with $a \in X \vee \mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ then there is a transition $Q \xrightarrow{a} Q'$ such that $P' \leftrightarrow_{br} Q'$,
 - c. if $\mathcal{I}(P) \cap ((X \cup Y) \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ then there is a transition $Q \xrightarrow{t} Q'$ such that $P' \leftrightarrow_{br}^Y Q'$.

Note that if a system has no time-out, then a generalised [rooted] branching reactive bisimulation is a stability respecting [rooted] branching bisimulation, thus proving that [rooted] branching reactive bisimilarity is indeed an extension of stability respecting [rooted] branching bisimilarity to reactive systems with time-outs.

► **Proposition 35.** Let $P, Q \in \mathbb{P}$ and $X \subseteq A$,

- $P \leftrightarrow_{br} Q$ (resp. $P \leftrightarrow_{br}^X Q$) iff there exists a generalised branching reactive bisimulation \mathcal{R} with $\mathcal{R}(P, Q)$ (resp. $\mathcal{R}(P, X, Q)$),
- $P \leftrightarrow_{br}^r Q$ (resp. $P \leftrightarrow_{br}^{rX} Q$) iff there exists a rooted generalised branching reactive bisimulation \mathcal{R} with $\mathcal{R}(P, Q)$ (resp. $\mathcal{R}(P, X, Q)$).

Proof. It suffices to verify that any [rooted] branching reactive bisimulation is a [rooted] generalised branching reactive bisimulation and that, for any [rooted] generalised branching reactive bisimulation \mathcal{R} , $\mathcal{R}' := \mathcal{R} \cup \{(P, X, Q) \mid \mathcal{R}(P, Q) \wedge X \subseteq A\} \cup \{(P, Y, Q), (P, Q) \mid \exists X \subseteq A, \mathcal{R}(P, X, Q) \wedge (\mathcal{I}(P) \cup \mathcal{I}(Q)) \cap (X \cup \{\tau\}) = \emptyset \wedge Y \subseteq A\}$ is a [rooted] branching reactive bisimulation. Details can be found in [20, Appendix C]. ◀

D Pohlmann Encoding

Reactive bisimulations are sometimes complicated to check because of the large number of potential sets of allowed actions. In [19], Pohlmann introduces an encoding which reduces strong reactive bisimilarity to strong bisimilarity. To this end he introduces unary operators ϑ and ϑ_X for $X \subseteq A$ that model placing their argument process in an environment that is triggered to change, or allows exactly the actions in X , respectively. The actions $t_\varepsilon \notin A$ and $\varepsilon_X \notin A$ for $X \subseteq A$ are generated by the new operators, but may not be used by processes substituted for their arguments P . They model a time-out action taken by the environment, and the stabilisation of an environment into one that allows exactly the set of actions X , respectively. After a slight modification of the encoding, a similar result can be obtained for branching reactive bisimilarity. We also introduce variants ϑ^r and ϑ_X^r of these operators that are targeting rooted branching reactive bisimilarity.

In [19], the first rule only applies to τ -transitions; this echoes the previous remark about applying the first clause of Definition 1 only to invisible actions. As the intermediary actions t_ε and $(\varepsilon_X)_{X \subseteq A}$ interfere with rootedness, the actions $(t_X)_{X \subseteq A}$ are added when rootedness has to be preserved. One can think of these as doing the actions ε_X and t in one (instead of two) steps. Note that the encoding rules mirror the clauses of Definition 1. The encoding transforms \leftrightarrow_{br} into \leftrightarrow_{tb} (see Definition 7), and \leftrightarrow_{br}^r in \leftrightarrow_{tb}^r (Definition 8).

36:22 Branching Bisimilarity for Processes with Time-Outs

■ **Table 4** Operational semantics of ϑ , ϑ^r , $(\vartheta_X)_{X \subseteq A}$ and $(\vartheta_X^r)_{X \subseteq A}$.

$$\begin{array}{l}
\vartheta(P) \xrightarrow{\alpha} \vartheta(P') \quad \wedge \quad \vartheta^r(P) \xrightarrow{\alpha} \vartheta^r(P') \quad \Leftrightarrow \quad P \xrightarrow{\alpha} P' \wedge \alpha \in A_\tau \\
\vartheta^r(P) \xrightarrow{t_X} \vartheta_X(P') \quad \Leftrightarrow \quad \mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset \wedge P \xrightarrow{t} P' \\
\vartheta(P) \xrightarrow{\varepsilon_X} \vartheta_X(P) \quad \wedge \quad \vartheta^r(P) \xrightarrow{\varepsilon_X} \vartheta_X(P) \\
\vartheta_X(P) \xrightarrow{\tau} \vartheta_X(P') \quad \wedge \quad \vartheta_X^r(P) \xrightarrow{\tau} \vartheta_X(P') \quad \Leftrightarrow \quad P \xrightarrow{\tau} P' \\
\vartheta_X(P) \xrightarrow{a} \vartheta(P') \quad \wedge \quad \vartheta_X^r(P) \xrightarrow{a} \vartheta(P') \quad \Leftrightarrow \quad P \xrightarrow{a} P' \wedge a \in X \\
\vartheta_X(P) \xrightarrow{t_\varepsilon} \vartheta(P) \quad \wedge \quad \vartheta_X^r(P) \xrightarrow{t_\varepsilon} \vartheta^r(P) \quad \Leftrightarrow \quad \mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset \\
\vartheta_X(P) \xrightarrow{t} \vartheta_X(P') \quad \wedge \quad \vartheta_X^r(P) \xrightarrow{t} \vartheta_X(P') \quad \Leftrightarrow \quad \mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset \wedge P \xrightarrow{t} P'
\end{array}$$

► **Proposition 36.** *Let $P, Q \in \mathbb{P}$.*

$$\begin{array}{ll}
\blacksquare P \Leftrightarrow_{br} Q \Leftrightarrow \vartheta(P) \Leftrightarrow_{tb} \vartheta(Q) & \blacksquare P \Leftrightarrow_{br}^X Q \Leftrightarrow \vartheta_X(P) \Leftrightarrow_{tb} \vartheta_X(Q) \\
\blacksquare P \Leftrightarrow_{br}^r Q \Leftrightarrow \vartheta^r(P) \Leftrightarrow_{tb}^r \vartheta^r(Q) & \blacksquare P \Leftrightarrow_{br}^{rX} Q \Leftrightarrow \vartheta_X^r(P) \Leftrightarrow_{tb}^r \vartheta_X^r(Q)
\end{array}$$

Proof. It suffices to prove that: if R is a branching reactive bisimulation then $R' := \{(\vartheta(P), \vartheta(Q)) \mid R(P, Q)\} \cup \{(\vartheta_X(P), \vartheta_X(Q)) \mid R(P, X, Q)\}$ is a t-branching bisimulation; and if R is a t-branching bisimulation then $R' := \{(P, Q), (P, X, Q) \mid R(\vartheta(P), \vartheta(Q)) \wedge X \subseteq A\} \cup \{(P, X, Q) \mid R(\vartheta_X(P), \vartheta_X(Q))\}$ is a branching reactive bisimulation. The rooted case is very similar. Details can be found in [20, Appendix D]. ◀

It would have been possible to define the t-branching bisimilarity differently while preserving the same result. The encoded processes are part of a sub-class with specific properties. For instance, an encoded process cannot have an outgoing τ -transition and an outgoing time-out by definition of ϑ and $(\vartheta_X)_{X \subseteq A}$, i.e., for any encoded process P , $P \xrightarrow{t} \Rightarrow P \not\xrightarrow{\tau}$. Thus, adding the condition $\forall i \in [0, r-1], Q_{2i+1} \not\xrightarrow{\tau}$ in clause 2 of Definition 7 does not interfere with our result even though it obviously defines a different bisimilarity. We settled on Definition 7 because it is the one that yields the simplest proofs.



A Spectrum of Approximate Probabilistic Bisimulations

Timm Spork  


Technische Universität Dresden, Dresden, Germany

Christel Baier  

Technische Universität Dresden, Dresden, Germany



Joost-Pieter Katoen  

RWTH Aachen University, Aachen, Germany

Jakob Piribauer  

Technische Universität Dresden, Dresden, Germany

Universität Leipzig, Leipzig, Germany

Tim Quatmann  

RWTH Aachen University, Aachen, Germany

Abstract

This paper studies various notions of approximate probabilistic bisimulation on labeled Markov chains (LMCs). We introduce approximate versions of weak and branching bisimulation, as well as a notion of ε -perturbed bisimulation that relates LMCs that can be made (exactly) probabilistically bisimilar by small perturbations of their transition probabilities. We explore how the notions interrelate and establish their connections to other well-known notions like ε -bisimulation.

2012 ACM Subject Classification Theory of computation \rightarrow Logic and verification; Theory of computation \rightarrow Random walks and Markov chains

Keywords and phrases Markov chains, Approximate bisimulation, Abstraction, Model checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.37

Related Version *Full Version:* <http://arxiv.org/abs/2407.07584>

Funding *Christel Baier, Jakob Piribauer and Timm Spork:* This work was partly funded by the DFG Grant 389792660 as part of TRR 248 (Foundations of Perspicuous Software Systems) and the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy).

Tim Quatmann: This research was funded by a KI-Starter grant from the Ministerium für Kultur und Wissenschaft NRW.

Acknowledgements We thank the reviewers for their helpful feedback, comments and suggestions. In particular, we thank the reviewer who pointed out the work in [31, 32] on approximate simulation relations which we were previously not aware of.

1 Introduction

Probabilistic model checking is widely used for the automatic verification of probabilistic models, like labeled Markov chains (LMC), against properties specified in (temporal) logics like PCTL* [11]. In practice, a big obstacle is the *state space explosion problem*: the number of states required to model a system can make its verification intractable [37, 11, 36].

To circumvent this issue, a well-established approach is the use of abstractions. For a given LMC \mathcal{M} , an abstraction \mathcal{A} is a model derived from \mathcal{M} that is (oftentimes) smaller than \mathcal{M} and preserves some properties of interest. Instead of verifying a formula on \mathcal{M} , one does so on \mathcal{A} and afterwards transfers the result back to the original model [11, 27, 38].



© Timm Spork, Christel Baier, Joost-Pieter Katoen, Jakob Piribauer, and Tim Quatmann; licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 37; pp. 37:1–37:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A prominent type of abstraction are probabilistic bisimulation quotients. They are constructed w.r.t. probabilistic bisimulations, a class of behavioral equivalences introduced by Larsen and Skou [41] as an extension of Milner’s bisimulation [43] to probabilistic models. A probabilistic bisimulation is an equivalence R on the state space of an LMC \mathcal{M} that only relates states that behave exactly the same, i.e., that have the same local properties, and transition to R -equivalence classes with equal probability. The coarsest probabilistic bisimulation \sim , called (probabilistic) bisimilarity, is the union of all probabilistic bisimulations in \mathcal{M} [11]. The bisimilarity relation can be computed efficiently [9, 19, 48] and preserves PCTL* state formulas [8, 33]. Since verifying PCTL* on bisimulation quotients can significantly speed up the verification process [37], their use is a vital part of probabilistic model checkers such as, e.g., STORM [34].

Other notions of behavioral equivalence are *weak* and *branching* probabilistic bisimulations [42, 51, 10, 12, 18, 50], which were introduced with the intention to abstract from sequences of *internal actions* or *stutter steps* a model can perform. Intuitively, these notions can abstract from the possibility of a state to, for some time, only visit equally labeled states (weak) or stay in its own equivalence class (branching) [35]. It is well-known that weak and branching probabilistic bisimilarity, denoted \approx^w and \approx^b , respectively, coincide for LMCs [10], and that they characterize satisfaction equivalence for a variant of PCTL* [24].

A problem with all of the above notions lies, however, in their lack of robustness against errors in the transition probabilities. The requirement of related states to have *exactly* the same transition probabilities to equivalence classes implies that even an infinitesimally small perturbation of any of these probabilities can cause two bisimilar states to become non-bisimilar, resulting in larger quotients [20, 52, 27]. This disadvantage was first observed in [30], where the use of *approximate* notions of bisimulation is suggested for its mitigation.

The literature proposes various types of approximate bisimilarity, the most well-known and well-studied one being ε -bisimilarity (\sim_ε) [25]. Other notions include approximate probabilistic bisimilarity with precision ε (\equiv_ε), or ε -APB for short [27, 1, 2], up-to- (n, ε) -bisimilarity (\sim_ε^n) [25, 13], or ε -lumpability of a given LMC [17, 29, 28]. Here, we propose definitions for approximate versions of weak (\approx_ε^w) and branching probabilistic bisimilarity (\approx_ε^b). Similar notions have, to the best of our knowledge, only been discussed sporadically in the context of noninterference under the term “weak bisimulation with precision ε ” [4, 7, 5, 6, 26, 3]. Moreover, we introduce ε -perturbed bisimilarity (\simeq_ε) which relates two LMCs if they can be made bisimilar by small perturbations of their transition probabilities. Implicitly, this relation arises in the work [38] on a type of abstraction called ε -quotients. With our definition, two LMCs are ε -perturbed bisimilar iff they have bisimilar ε -quotients.

All of the approximate notions have in common that they allow a small tolerance, say $\varepsilon > 0$, in the transition probabilities of related states, but differ in the specifics of where and how this tolerance is put to use. Broadly speaking, we can distinguish two groups of relations: while \sim_ε , \equiv_ε , \sim_ε^n and \approx_ε^w are additive in their tolerances and are closer to classic process relations, the notions underlying \sim_ε^* and \equiv_ε^* , denoting *transitive* ε -bisimilarity and *transitive* ε -APB, respectively, as well as \simeq_ε and \approx_ε^b are better suited for the construction of abstractions since they are required to be equivalences. Collapsing the equivalence classes of such a relation into single states yields quotient models, which in some cases are such that formulas given in specific (fragments of) logics are (approximately) preserved between the original LMC and its quotient. However, it turns out that requiring transitivity can cause some unnatural behavior, like the possibility to distinguish probabilistically bisimilar LMCs and a lack of additivity. Furthermore, the induced bisimilarity relations, which are again defined as the union of all corresponding relations in the model \mathcal{M} (e.g., \approx_ε^b is the union of

■ **Table 1** Overview of the notions of approximate bisimulation we consider and some of their properties. Being suitable for “quotienting” is meant w.r.t. the underlying bisimulation relation.

Notion	Symbol for Union	Additive	Quotienting
ε -Bisimulation [25, 14]	\sim_ε	}	✓
ε -APB [27, 1, 2]	\equiv_ε		
Up-To- (n, ε) -Bisimulation [25, 13]	\sim_ε^n		
Weak ε -Bisimulation	\approx_ε^w		
Transitive ε -Bisimulation	\sim_ε^*	}	×
Transitive ε -APB	\equiv_ε^*		
Branching ε -Bisimulation	\approx_ε^b		
ε -Perturbed Bisimulation [38]	\simeq_ε		

all branching ε -bisimulations in \mathcal{M}) might themselves not be of the respective type anymore (e.g., \approx_ε^b is not necessarily a branching ε -bisimulation). This contrasts the non-transitive case, where the induced bisimilarity relations are always of the respective type. We summarize the relations we consider, together with some of their properties, in Table 1.

Main Contributions. The main contributions are as follows:

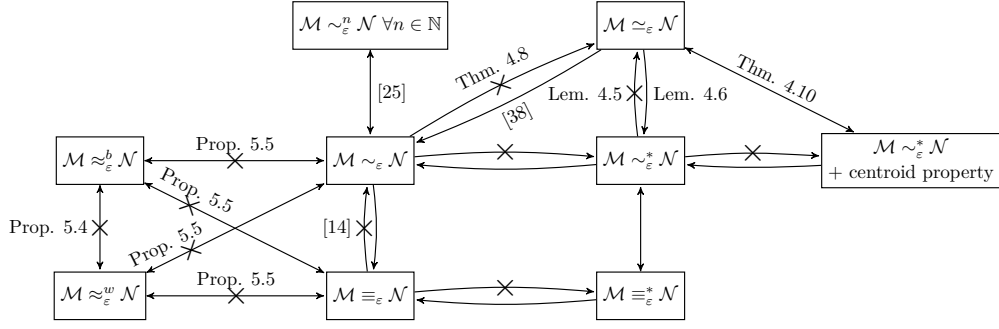
1. Starting with the classic notion of ε -bisimilarity, we show tightness of a bound from [32] on the absolute difference of unbounded reachability probabilities in ε -bisimilar states (Example 3.13).
2. We introduce ε -perturbed bisimilarity (\simeq_ε), a notion that relates two LMCs if they have bisimilar ε -quotients á la [38], i.e., if they can be made probabilistically bisimilar by small perturbations of their transition probabilities. We show that \simeq_ε is strictly finer than (transitive) ε -bisimilarity \sim_ε^* (Lemma 4.6 and Theorem 4.7) and that deciding both \simeq_ε and \sim_ε^* is NP-complete (Theorem 4.12). Furthermore, we characterize \simeq_ε in terms of transitive ε -bisimulations satisfying a *centroid property* (Theorem 4.10) and discuss some anomalies of \simeq_ε : the relation is not always an ε -perturbed bisimulation itself, it is not additive in ε and it can distinguish bisimilar LMCs (Proposition 4.4).
3. We define approximate versions of weak (\approx_ε^w) and branching probabilistic bisimilarity (\approx_ε^b). Our definitions can be evaluated locally and coincide with the exact notions \approx^w and \approx^b , respectively, if $\varepsilon = 0$. We discuss how \approx_ε^w and \approx_ε^b are related to one another, as well as to ε -bisimilarity (Propositions 5.4 and 5.5). Moreover, we extend the bounds for reachability probabilities of Theorem 3.11 to states related by \approx_ε^w and \approx_ε^b (Corollary 5.9 and Proposition 5.10), and prove that deciding \approx_ε^b is NP-complete (Theorem 5.11).

Together with various known results from the literature and some easy observations, our results complete the relation between several notions of approximate probabilistic bisimulation, as summarized in Figure 1.

Structure. Section 2 presents preliminaries. Section 3 considers ε -bisimulations, ε -APBs and up-to- (n, ε) -bisimulations. Section 4 introduces and analyzes ε -perturbed bisimulations. Section 5 introduces weak and branching ε -bisimulations and establishes how they relate to ε -bisimulations. Section 6 summarizes our results and points out future work.

2 Preliminaries

Distributions. $Distr(S) = \{\mu: S \rightarrow [0, 1] \mid \sum_{s \in S} \mu(s) = 1\}$ is the set of *distributions* over countable $S \neq \emptyset$. $\mu \in Distr(S)$ has *support* $supp(\mu) = \{s \in S \mid \mu(s) > 0\}$, and for $A \subseteq S$ we set $\mu(A) = \sum_{s \in A} \mu(s)$. The L_1 -distance of $\mu, \nu \in Distr(S)$ is $\|\mu - \nu\|_1 = \sum_{s \in S} |\mu(s) - \nu(s)|$.



■ **Figure 1** The relationship of different approximate probabilistic bisimulations.

Labeled Markov chains. Fix a countable set AP of *atomic propositions*. A *labeled Markov chain* (LMC) $\mathcal{M} = (S, P, s_{init}, l)$ has a countable set of *states* $S \neq \emptyset$, a *transition distribution function* $P: S \rightarrow \text{Distr}(S)$, a unique *initial state* s_{init} , and a *labeling function* $l: S \rightarrow 2^{AP}$. We use \mathcal{M} and \mathcal{N} to range over LMCs. For $s \in S$, let $L(s) = \{t \in S \mid l(s) = l(t)\}$. \mathcal{M} is *finitely branching* if $|\text{supp}(P(s))| < \infty$ for all $s \in S$, and \mathcal{M} is *finite* if $|S| < \infty$. The *direct sum* $\mathcal{M} \oplus \mathcal{N}$ is the LMC obtained from the disjoint union of \mathcal{M} and \mathcal{N} . The initial state of $\mathcal{M} \oplus \mathcal{N}$ is not relevant for our purposes.

For $s, t \in S$, $P(s)(t)$ denotes the probability to move from s to t in a single step. We write $\text{Succ}(s) = \text{supp}(P(s))$ for the set of direct successors of s . $\pi = s_0 s_1 \dots \in S^\omega$ is an (*infinite*) *path* of \mathcal{M} if $s_{i+1} \in \text{Succ}(s_i)$ for all $i \in \mathbb{N}$. $\pi[i] = s_i$ is the state at position i of π , and $\text{trace}(\pi) = l(s_0)l(s_1)\dots \in (2^{AP})^\omega$ is the *trace* of π . The set of infinite paths is $\text{Paths}(\mathcal{M})$. *Finite* paths $\pi = s_0 s_1 \dots s_k \in S^{k+1}$ for some $k \in \mathbb{N}$ and their traces are defined analogously.

Let $s \in S$. We consider the standard probability measure $\text{Pr}_s^{\mathcal{M}}$ on sets of infinite paths of LMCs, defined via *cylinder sets* $\text{Cyl}(\rho) = \{\pi \in \text{Paths}(\mathcal{M}) \mid \rho \text{ is a prefix of } \pi\}$ of finite paths $\rho \in S^*$. See [11] for details. For $\rho = s_0 s_1 \dots s_n$, we abbreviate $\text{Pr}_s^{\mathcal{M}}(\text{Cyl}(\rho))$ by $\text{Pr}_s^{\mathcal{M}}(\rho)$ and the measure yields $\text{Pr}_s^{\mathcal{M}}(\rho) = 0$ if $s_0 \neq s$ and $\text{Pr}_s^{\mathcal{M}}(\rho) = \prod_{j=0}^{n-1} P(s_j)(s_{j+1})$ otherwise. We write $\text{Pr}^{\mathcal{M}}$ for $\text{Pr}_{s_{init}}^{\mathcal{M}}$ and drop the superscript if \mathcal{M} is clear from the context. Given a set of finite traces $T \subseteq (2^{AP})^{k+1}$ for some $k \in \mathbb{N}$, $\text{Pr}_s^{\mathcal{M}}(T)$ denotes the probability to follow, when starting in s , a finite path $\pi = s s_1 \dots s_k$ with $\text{trace}(\pi) \in T$. $\mathbb{E}_s^{\mathcal{M}}(X)$ or simply $\mathbb{E}_s(X)$ denotes the *expected value* of a random variable X on $\text{Paths}(\mathcal{M})$ w.r.t. $\text{Pr}_s^{\mathcal{M}}$.

LTL. A popular logic for the specification of desired properties of LMCs is the *linear temporal logic* (LTL) which can be used to, e.g., specify properties such as reachability, safety or liveness [44, 11]. For $a \in AP$, LTL formulas are formed w.r.t. the grammar

$$\varphi ::= \text{true} \mid a \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathbf{U}\varphi_2.$$

Here, \bigcirc is the *next* operator, so $\pi \in \text{Paths}(\mathcal{M})$ satisfies $\bigcirc\varphi$ iff φ is true in $\pi[1]$. For the *until* operator \mathbf{U} , π satisfies $\varphi_1 \mathbf{U}\varphi_2$ iff, alongside π , φ_1 holds until φ_2 is true. As syntactic sugar we define the *reachability* operator $\diamond\varphi \equiv \text{true}\mathbf{U}\varphi$ and the *always* operator $\square\varphi \equiv \neg\diamond\neg\varphi$.

For $B, C \subseteq S$ and $s \in S$, $\text{Pr}_s(\text{BUC})$ is the probability to reach a state in C via a (finite) path from s that only consists of states in B . Moreover, $\text{Pr}_s(\diamond^{\leq n}\varphi)$ denotes the probability to reach a state satisfying φ from s in at most $n \in \mathbb{N}$ steps. For details on LTL, see [11].

Relations. Given a relation $R \subseteq S \times S$ and an $A \subseteq S$, $R(A) = \{t \in S \mid \exists s \in A: (s, t) \in R\}$ is the *image of A under R*. If R is reflexive then $A \subseteq R(A)$, and A is called *R-closed* if $R(A) \subseteq A$. When R is an equivalence, i.e., when it is reflexive, symmetric and transitive, the *equivalence class* of $s \in S$ is $[s]_R = R(\{s\}) = \{t \in S \mid (s, t) \in R\}$, and we set $S/R = \{[s]_R \mid s \in S\}$. For an equivalence R , the *R-closed* sets are precisely the (unions of) *R* equivalence classes.

Bisimulation. An equivalence $R \subseteq S \times S$ is a (*probabilistic*) *bisimulation* on \mathcal{M} if for all $(s, t) \in R$ and all R -equivalence classes C it holds that $l(s) = l(t)$ and $P(s)(C) = P(t)(C)$. States $s, t \in S$ are (*probabilistically*) *bisimilar*, written $s \sim^{\mathcal{M}} t$ or simply $s \sim t$, if there is a bisimulation R on \mathcal{M} with $(s, t) \in R$. We call two LMCs \mathcal{M}, \mathcal{N} *bisimilar*, written $\mathcal{M} \sim \mathcal{N}$, if $s_{init}^{\mathcal{M}} \sim s_{init}^{\mathcal{N}}$ in $\mathcal{M} \oplus \mathcal{N}$. An alternative characterization of bisimulations can be found in, e.g., [23, 25, 27, 14]: an equivalence R is a bisimulation iff for all $(s, t) \in R$ and all R -closed sets $A \subseteq S$ it holds that $l(s) = l(t)$ and $P(s)(A) = P(t)(A)$. The (*probabilistic bisimulation*) *quotient* of \mathcal{M} is the LMC $\mathcal{M}/\sim = (S/\sim, P_\sim, [s_{init}]_\sim, l_\sim)$ with $l_\sim([s]_\sim) = l(s)$, and $P_\sim([s]_\sim)([t]_\sim) = \sum_{q \in [t]_\sim} P(s)(q)$ for all $[s]_\sim, [t]_\sim \in S/\sim$. It holds that $\mathcal{M} \sim \mathcal{M}/\sim$. An important result is that *bisimilarity* \sim preserves the satisfaction of PCTL* state formulas [8].

We also consider *weak* and *branching probabilistic bisimulations* [43, 51, 10, 35]. An equivalence R is a *weak probabilistic bisimulation* if, for all $(s, t) \in R$ and all R -equivalence classes $C \neq [s]_R = [t]_R$, it holds that $l(s) = l(t)$ and $\Pr_s(L(s)UC) = \Pr_t(L(t)UC)$. R is a *branching probabilistic bisimulation* if, instead of the second condition in the previous definition, $\Pr_s([s]_R UC) = \Pr_t([t]_R UC)$ holds. *Weak probabilistic bisimilarity* \approx^w and *branching probabilistic bisimilarity* \approx^b are defined like \sim , and lifted to LMCs in the same way.

3 ε -Bisimulation, ε -APB and Up-To- (n, ε) -Bisimulation

If not specified otherwise, we always assume $\varepsilon \in [0, 1]$ and $\mathcal{M} = (S, P, s_{init}, l)$ to be finitely branching. This section summarizes various notions of approximate probabilistic bisimulation from the literature. We first provide their formal definitions and discuss how the notions interrelate. Afterwards, in Section 3.2, we present some logical preservation results.

3.1 Definitions and Interrelation

We start with the seminal notion of ε -*bisimulations* of Desharnais *et al.* [25]. While originally introduced for *labeled Markov processes* [21, 22], ε -bisimulations were later adapted to other models like LMCs [14, 38] or Segala's *probabilistic automata* [45, 47].

► **Definition 3.1** ([25, 14]). *A reflexive¹ and symmetric relation $R \subseteq S \times S$ is an ε -bisimulation if for all $(s, t) \in R$ and any $A \subseteq S$ it holds that*

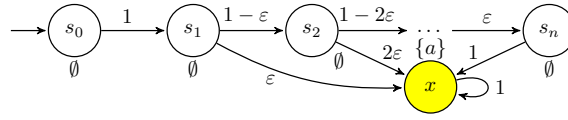
$$(i) \ l(s) = l(t) \quad \text{and} \quad (ii) \ P(s)(A) \leq P(t)(R(A)) + \varepsilon.$$

States s, t are ε -bisimilar, denoted $s \sim_\varepsilon t$, if there is an ε -bisimulation R with $(s, t) \in R$. LMCs \mathcal{M}, \mathcal{N} are ε -bisimilar, denoted $\mathcal{M} \sim_\varepsilon \mathcal{N}$, if $s_{init}^{\mathcal{M}} \sim_\varepsilon s_{init}^{\mathcal{N}}$ in $\mathcal{M} \oplus \mathcal{N}$.

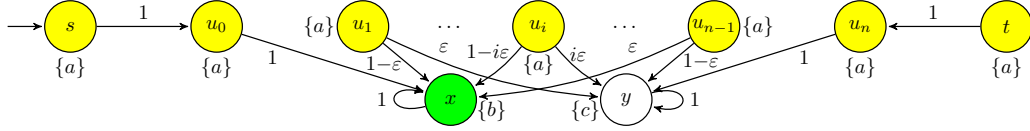
Intuitively, $s \sim_\varepsilon t$ if both states can mimic the other's transition probabilities to any $A \subseteq S$ by transitioning to the (potentially bigger) set $\sim_\varepsilon(A)$ with a probability that is smaller by at most ε than the original one. The parameter ε describes how much the behavior of related states may differ: for ε close to 1 more states can be related, while for $\varepsilon \approx 0$ related states behave almost equivalently. In the extreme case of $\varepsilon = 0$, we have $\sim_0 = \sim$ [25, 14].

Instead of being transitive, ε -bisimulations are *additive* in their tolerances: $s \sim_{\varepsilon_1} t$ and $t \sim_{\varepsilon_2} u$ implies $s \sim_{\varepsilon'} u$ for some $0 \leq \varepsilon' \leq \min\{1, \varepsilon_1 + \varepsilon_2\}$ [25]. As the next example suggests, transitivity is not always desirable for ε -bisimulations if $\varepsilon > 0$.

¹ In contrast to [25, 14] we require reflexivity of ε -bisimulations. This is a rather natural assumption (a state should always simulate itself) that does not affect \sim_ε .



■ **Figure 2** The LMC used in Example 3.2.



■ **Figure 3** The LMC used in Example 3.5, adapted from [14].

► **Example 3.2.** Let $\varepsilon = \frac{1}{n}$ for $n \geq 1$ and consider the LMC of Figure 2. There, the reflexive and symmetric closure of $R = \{(s_i, s_{i+1}) \mid 0 \leq i \leq n-1\}$ is an ε -bisimulation. Hence, s_0 and s_n are related by a chain of ε -bisimilar states, even though they behave completely different: s_0 transitions to the $\{a\}$ -labeled state x with probability 0, s_n does so with probability 1.

Desharnais *et al.* [25] describe how to check condition (ii) of Definition 3.1 in terms of the values of maximum flows in specific flow networks á la [46, 9]. Their result is well-suited for algorithmic purposes, but is restricted to finite models. Equivalently, one can characterize ε -bisimulations by the existence of *weight functions* $\Delta: S \rightarrow \text{Distr}(S)$ that describe how to split the successor probabilities of related states. This formulation is used in, e.g., [47, 38, 39, 31]. The following lemma provides one such characterization that is proved using ideas from [14, 15] and a technical measure-theoretic statement from [16].

► **Lemma 3.3.** *A reflexive and symmetric relation $R \subseteq S \times S$ that only relates states with the same label is an ε -bisimulation iff for all $(s, t) \in R$ there is a map $\Delta: \text{Succ}(s) \rightarrow \text{Distr}(\text{Succ}(t))$ such that*

1. *for all $t' \in \text{Succ}(t)$ we have $P(t)(t') = \sum_{s' \in \text{Succ}(s)} P(s)(s') \cdot \Delta(s')(t')$, and*
2. *$\sum_{s' \in \text{Succ}(s)} P(s)(s') \cdot \Delta(s')(R(s') \cap \text{Succ}(t)) \geq 1 - \varepsilon$.*

Intuitively, Lemma 3.3 tells us that, if $s \sim_\varepsilon t$, the successors s' of s can be mapped to distributions $\Delta(s')$, i.e., convex combinations, of successors of t . More precisely, it shows that (i) if we move from s to a successor s' with probability $P(s)(s')$ and, afterwards, from s' to a successor t' of t with probability $\Delta(s')(t')$, then we reach t' with probability $P(t)(t')$, and that (ii) the overall probability that the states s' and t' are ε -bisimilar is at least $1 - \varepsilon$.

A second notion of approximate probabilistic bisimulation are ε -APBs, which stands short for *approximate probabilistic bisimulations with precision ε* [27, 1, 2]. In contrast to ε -bisimulations, where the differences in transition probabilities of related states are bounded w.r.t. all subsets $A \subseteq S$, an ε -APB R only requires a difference of at most ε for the probabilities of related states to transition to R -closed subsets of S .

► **Definition 3.4** ([27]). *A reflexive and symmetric relation $R \subseteq S \times S$ is an ε -APB if for all $(s, t) \in R$ and any R -closed set $A \subseteq S$ it holds that*

$$(i) \ l(s) = l(t) \quad \text{and} \quad (ii) \ |P(s)(A) - P(t)(A)| \leq \varepsilon.$$

We write $s \equiv_\varepsilon t$ if s and t are related by any ε -APB, and $\mathcal{M} \equiv_\varepsilon \mathcal{N}$ if $s_{init}^{\mathcal{M}} \equiv_\varepsilon s_{init}^{\mathcal{N}}$ in $\mathcal{M} \oplus \mathcal{N}$.

Like \sim_ε , ε -APBs are additive in their tolerances, and we have $\equiv_0 = \sim = \sim_0$ [25, 27].

► **Example 3.5** ([14]). Let $\varepsilon \in (0, 1]$ and $n = \lceil \frac{1}{\varepsilon} \rceil \in \mathbb{N}$. Consider \mathcal{M} as in Figure 3, and let R be the reflexive and symmetric closure of $\{(s, t), (x, x), (y, y)\} \cup \{(u_i, u_{i+1}) \mid 0 \leq i \leq n-1\}$. The R -closed sets in \mathcal{M} are $\{s, t\}, \{x\}, \{y\}, \{u_i \mid 0 \leq i \leq n\}$ and their unions. For all $(p, q) \in R$ and R -closed sets A it holds that $|P(p)(A) - P(q)(A)| \leq \varepsilon$, so R is an ε -APB.

Example 3.5 illustrates that the use of ε -APBs as a notion that relates states with almost equivalent behavior is questionable: even though states s and t in Figure 3 are related by \equiv_ε , they behave completely different. This is caused by the set $\{u_0, \dots, u_n\}$ of (unreachable) states being R -closed, which in turn allows to relate s and t by the relation R from the example. Such an anomaly cannot occur for \sim_ε , and we in fact have $s \sim_\varepsilon t$ in Figure 3 for every $\varepsilon \in (0, 1)$. In particular, this shows that \sim_ε can be strictly finer than \equiv_ε for $\varepsilon \in (0, 1)$.

Lastly, we introduce *up-to- (n, ε) -bisimulations* [25, 13], which are relations that require the behaviors of related states to differ by at most ε for at least n steps.

► **Definition 3.6** ([25, 13]). *The up-to- (n, ε) -bisimulation $\sim_\varepsilon^n \subseteq S \times S$ is inductively defined on n via $s \sim_\varepsilon^0 t$ for all $s, t \in S$ and, for $n \geq 0$, $s \sim_\varepsilon^{n+1} t$ iff for all $A \subseteq S$*

$$(i) \ l(s) = l(t), \quad (ii) \ P(s)(A) \leq P(t)(\sim_\varepsilon^n(A)) + \varepsilon \quad \text{and} \quad (iii) \ P(t)(A) \leq P(s)(\sim_\varepsilon^n(A)) + \varepsilon.$$

States s, t are *(n, ε) -bisimilar* if $s \sim_\varepsilon^n t$, and the notion is lifted to LMCs as usual. Similar to \sim_ε and \equiv_ε , \sim_ε^n is reflexive and symmetric, but not transitive. Instead, it is additive in the tolerances and monotonic in n and ε , i.e., for $n \geq n'$ and $\varepsilon \leq \varepsilon'$, $s \sim_\varepsilon^n t$ implies $s \sim_{\varepsilon'}^{n'} t$ [13].

It is clear that $s \sim_\varepsilon^n t$ for a fixed n does not necessarily imply $s \sim_\varepsilon t$ or $s \equiv_\varepsilon t$, as (n, ε) -bisimilarity only restricts the behavior of related states for n steps. However, considering the limit $n \rightarrow \infty$ makes \sim_ε and \sim_ε^n coincide, i.e., $s \sim_\varepsilon t$ iff $s \sim_\varepsilon^n t$ for all $n \in \mathbb{N}$ [25].

We now make precise the relationship between ε -APBs and up-to- (n, ε) -bisimulations.

► **Proposition 3.7.** *If $\varepsilon \in (0, 1)$, $s \equiv_\varepsilon t$ implies $s \sim_\varepsilon^n t$ if $n \leq 2$, but not necessarily if $n \geq 3$.*

3.2 Preservation of Logical Properties

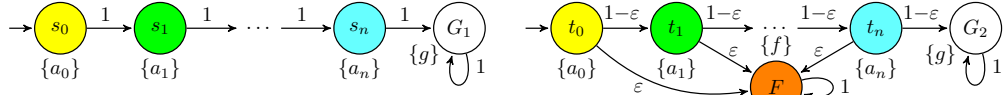
A key application of exact probabilistic bisimilarity \sim is the use of quotients $\mathcal{Q} = \mathcal{M}/\sim$ to speed up PCTL* model checking [37, 36]. As abstractions built by grouping states related by approximate probabilistic bisimulations can be smaller than \mathcal{Q} [27], these notions might prove useful to combat the *state space explosion problem* of model checking [37, 11, 36]. It is hence of interest to see which logical properties these relations preserve.

We start by considering \sim_ε . As shown by Bian and Abate [14], ε -bisimilarity induces bounds on the absolute difference of satisfaction probabilities of *finite horizon* properties, i.e., of properties that only depend on traces of finite length, in related states.

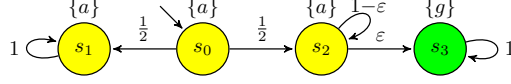
► **Theorem 3.8** ([14]). *Let $s \sim_\varepsilon t$, $k \in \mathbb{N}$ and $T \subseteq (2^{AP})^{k+1}$ a set of traces of length $k+1$. Then $|\Pr_s(T) - \Pr_t(T)| \leq 1 - (1 - \varepsilon)^k$.*

Since any finite horizon LTL formula coincides with a set of traces of finite length, Theorem 3.8 in particular bounds the satisfaction probabilities of such formulas in ε -bisimilar states. Furthermore, as argued in [14] and the next example, this bound is tight.

► **Example 3.9.** Consider Figure 4. For $i \in \{0, \dots, n\}$, let $l(s_i) = l(t_i) = a_i$ for pairwise distinct a_i , $l(G_1) = g = l(G_2)$ and $l(F) = f$ for some $f \neq g$. Then $s_0 \sim_\varepsilon t_0$ and the upper bound of Theorem 3.8 is met exactly: $|\Pr_{s_0}(\diamond^{\leq n+1} g) - \Pr_{t_0}(\diamond^{\leq n+1} g)| = 1 - (1 - \varepsilon)^{n+1}$.



■ **Figure 4** An LMC in which $s_0 \sim_\varepsilon t_0$ and $|\Pr_{s_0}(\diamond^{\leq n+1}g) - \Pr_{t_0}(\diamond^{\leq n+1}g)| = 1 - (1 - \varepsilon)^{n+1}$.



■ **Figure 5** The LMC used in Example 3.10.

A disadvantage of the bound provided in Theorem 3.8 is, however, that it rapidly converges to 1 for increasing k and is thus not suitable when reasoning about long (or infinite) time horizons. In fact, it is the case that – without further assumptions – even simple unbounded reachability probabilities in ε -bisimilar states can strongly deviate.

► **Example 3.10.** Let $\varepsilon \geq 0$. The states s_0 , s_1 , and s_2 in Figure 5 are pairwise ε -bisimilar. However, if $\varepsilon > 0$, we have $\Pr_{s_0}(\diamond g) = \frac{1}{2}$, $\Pr_{s_1}(\diamond g) = 0$, and $\Pr_{s_2}(\diamond g) = 1$.

The difference in reachability probabilities observed in the last example is caused by \sim_ε relating states that are able to reach a goal state g with positive probability to those that can not reach g at all. One way to avoid this issue is to require that states from which g is not reachable are labeled with a distinct label f . The existence of such a label f is a rather natural assumption, as a typical preprocessing step when computing reachability probabilities is to identify the states from which no goal state is reachable, i.e., to identify the states we assume to be labeled with f [11]. A result in the spirit of Theorem 3.8 that deals with *unbounded* reachability properties can then be obtained as follows.

► **Theorem 3.11** ([31, 32]). *Let some states in \mathcal{M} be labeled with g , and let exactly the states that cannot reach a g -labeled state be labeled with f . Further, let $s \sim_\varepsilon t$, and let N be the random variable that counts the number of steps until reaching a g - or f -labeled state. Then,*

$$|\Pr_s(\diamond g) - \Pr_t(\diamond g)| \leq \varepsilon \cdot \mathbb{E}_s(N).$$

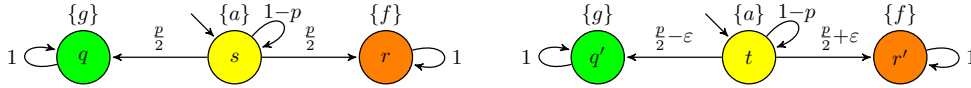
► **Remark 3.12.** A result similar to Theorem 3.11 is derived by Haesaert *et al.* in [31, 32] in the context of policy synthesis in control theory. In fact, their result is more general, as it considers all properties that can be described as the language of a deterministic finite automaton. These properties include, among others, the *syntactically co-safe* LTL formulas [40], which form a fragment of LTL built according to the grammar

$$\varphi ::= \text{true} \mid a \mid \neg a \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathbf{U} \varphi_2,$$

where $a \in AP$. As unbounded reachability $\diamond g$ is a syntactically co-safe LTL formula, the results of [31, 32] extend the bound in Theorem 3.11 to a broader class of properties.

Next, we show that the bound described in Theorem 3.11 is actually tight.

► **Example 3.13.** Let $p \in (0, 1)$, $\varepsilon < \frac{p}{2}$ and consider Figure 6, where $s \sim_\varepsilon t$. There, $\Pr_s(\diamond g) = \frac{1}{2}$, $\Pr_t(\diamond g) = \frac{1}{2} - \frac{\varepsilon}{p}$ and $\mathbb{E}_s(N) = \mathbb{E}_t(N) = \frac{1}{p}$. Hence, the bound in Theorem 3.11 is met exactly: $|\Pr_s(\diamond g) - \Pr_t(\diamond g)| = \frac{\varepsilon}{p} = \varepsilon \cdot \mathbb{E}_s(N) = \varepsilon \cdot \mathbb{E}_t(N)$.



■ **Figure 6** The LMC used in Example 3.13. The states s and t are ε -bisimilar.

Regarding ε -APBs and up-to- (n, ε) -bisimilarity, some preservation results w.r.t. the (approximate or robust) satisfaction of PCTL state-formulas can be found in the literature. Since, as we have seen in Section 3.1, $s \sim_\varepsilon t$ implies both $s \equiv_\varepsilon t$ and $s \sim_\varepsilon^n t$ for any $n \in \mathbb{N}$ and any two states s, t , the following results also hold for ε -bisimilar states.

An important property of ε -APBs is that related states satisfy the same ε -robust PCTL state formulas Φ_{robust} [27], i.e., that $s \equiv_\varepsilon t$ implies that $s \models \Phi_{\text{robust}}$ iff $t \models \Phi_{\text{robust}}$, where \models is the usual PCTL *satisfaction relation* [11]. Intuitively, Φ_{robust} is ε -robust if for all subformulas ϕ of Φ_{robust} and all $s \in S$ either a strengthened version of ϕ , obtained by making ϕ 's probability thresholds harder to meet, holds in s , or even relaxing ϕ 's probability thresholds is not sufficient to ensure that s satisfies ϕ . For details, see [27].

Furthermore, it was shown in [13] that (n, ε) -bisimilar states approximately satisfy the same bounded PCTL state formulas. The fragment of PCTL considered does not allow unbounded until, and requires all until operator appearing in a formula to have the same time bound $k \in \mathbb{N}$. Under these assumptions, the precision of the approximation of satisfaction probabilities between (n, ε) -bisimilar states is proved to depend linearly on the parameters n and ε , as well as the common step bound k of the until operators. For details, see [13].

4 ε -Perturbed Bisimulation

In this section we consider finite LMCs. In [38], Kiefer and Tang define the notion of ε -quotients for $\varepsilon \geq 0$. Their goal is to construct, from a given *perturbed* LMC \mathcal{M}' , an abstraction that is as close as possible to the exact bisimulation quotient of an unknown, unperturbed LMC \mathcal{M} corresponding to \mathcal{M}' . This inspires us to introduce ε -perturbed bisimulations, which relate two LMCs iff they can be made probabilistically bisimilar by small perturbations of their transition probabilities. Since we require ε -perturbed bisimulations to be equivalences, these relations are well-suited for the construction of quotients of a given model.

Like the ε -quotients of [38], we base our definition on ε -perturbations of LMCs.

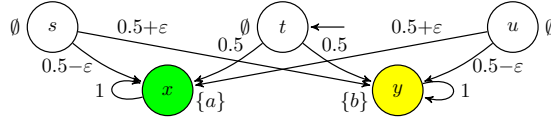
► **Definition 4.1** ([38]). $\mathcal{M}' = (S, P', s_{\text{init}}, l)$ is an ε -perturbation of $\mathcal{M} = (S, P, s_{\text{init}}, l)$ if $\|P(s) - P'(s)\|_1 \leq \varepsilon$ for all $s \in S$.

\mathcal{M} and any of its ε -perturbations \mathcal{M}' have the same state space and labeling, and we often write $S' = \{s' \mid s \in S\}$ for the state space of \mathcal{M}' . Hence, \mathcal{M} and \mathcal{M}' only differ in their transition distribution functions. However, \mathcal{M}' does not need to preserve the structure of \mathcal{M} , i.e., there can be transitions in \mathcal{M} that have probability 0 in \mathcal{M}' and vice versa. As the next lemma shows, the total probability mass of these transitions cannot exceed $\frac{\varepsilon}{2}$.

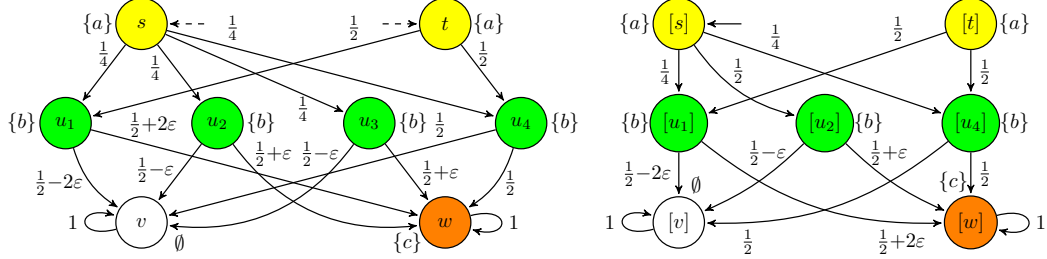
► **Lemma 4.2.** For all $s \in S$ and $A \subseteq S$ it holds that $|P(s)(A) - P'(s')(A')| \leq \frac{\varepsilon}{2}$.

We now define the novel notion of ε -perturbed bisimulation.

► **Definition 4.3.** An equivalence $R \subseteq S \times S$ is called an ε -perturbed bisimulation on \mathcal{M} if there is an ε -perturbation \mathcal{M}' of \mathcal{M} such that R is a bisimulation on \mathcal{M}' . Two states $s, t \in S$ are ε -perturbed bisimilar, denoted $s \simeq_\varepsilon t$, if they are related by some ε -perturbed bisimulation. Given LMCs \mathcal{M} and \mathcal{N} , then $\mathcal{M} \simeq_\varepsilon \mathcal{N}$ if $s_{\text{init}}^{\mathcal{M}} \simeq_\varepsilon s_{\text{init}}^{\mathcal{N}}$ in $\mathcal{M} \oplus \mathcal{N}$.



■ **Figure 7** An LMC in which there is no unique maximal transitive ε -bisimulation.



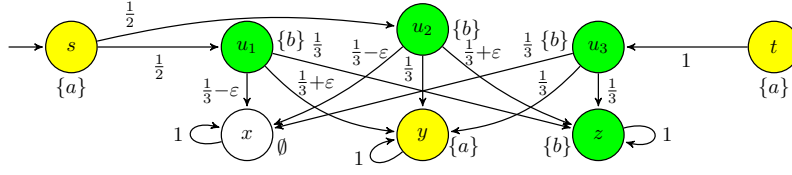
■ **Figure 8** Two LMCs \mathcal{M}_s and \mathcal{M}_t (left) with initial states s and t , respectively, and $\mathcal{Q} = \mathcal{M}_s/\sim$ (right), demonstrating that \simeq_ε and \sim_ε^* can differentiate bisimilar models and are not additive.

In the terminology of [38], $\mathcal{M} \simeq_\varepsilon \mathcal{N}$ iff \mathcal{M} and \mathcal{N} have bisimilar ε -perturbations iff there are bisimilar ε -quotients of \mathcal{M} and \mathcal{N} . If all states of \mathcal{M} and \mathcal{N} are reachable, even the stronger characterization $\mathcal{M} \simeq_\varepsilon \mathcal{N}$ iff \mathcal{M} and \mathcal{N} have *isomorphic* ε -perturbations iff there are *isomorphic* ε -quotients of \mathcal{M} and \mathcal{N} holds. Since the unique 0-perturbation of any LMC is the LMC itself, $\mathcal{M} \simeq_0 \mathcal{N}$ iff $\mathcal{M} \sim \mathcal{N}$. Moreover, \simeq_ε is symmetric and reflexive, but not always transitive, which implies that \simeq_ε is not necessarily an ε -perturbed bisimulation itself.

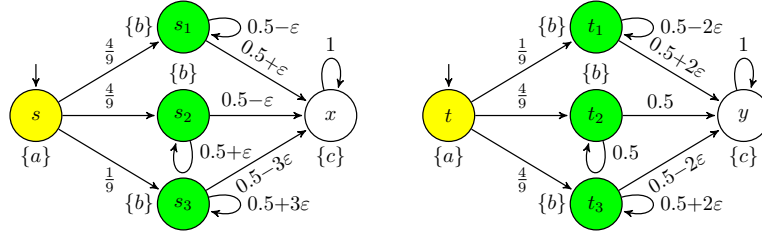
Let $s \sim_\varepsilon^* t$ denote that states s and t are related by a *transitive* ε -bisimulation. We remark that both \simeq_ε and \sim_ε^* are definitions in the spirit of a notion called *ε -lumpability* (or *quasi-lumpability*), which describes that a LMC can be made exactly lumpable w.r.t. a given equivalence by slight changes (up to ε in each value) of its transition probabilities [17, 29, 28]. In contrast to the non-transitive case, any transitive ε -APB is also an ε -bisimulation.

The requirement of transitivity comes with the downside that there is not always a *unique* largest transitive ε -bisimulation: in Figure 7, no transitive ε -bisimulation R can contain both (s, t) and (t, u) , as otherwise also $(s, u) \in R$ must hold. However, $s \sim_\varepsilon^* t$ and $t \sim_\varepsilon^* u$ as $R_1 = \{\{s, t\}, \{u\}, \{x\}, \{y\}\}$ and $R_2 = \{\{s\}, \{t, u\}, \{x\}, \{y\}\}$ are transitive ε -bisimulations. Hence, the union of all transitive ε -bisimulations in a given model is thus not always a transitive ε -bisimulation itself. This is different than in the non-transitive case, where \sim_ε is always an ε -bisimulation [25]. Since $s \simeq_\varepsilon t$ and $t \simeq_\varepsilon u$ but $s \not\simeq_\varepsilon u$ in Figure 7, it follows that there is also not always a unique largest ε -perturbed bisimulation.

Now consider, for $\varepsilon < \frac{1}{4}$, the LMCs \mathcal{M}_s and \mathcal{M}_t on the left of Figure 8, with initial states s and t , respectively. In both models, \sim is the finest equivalence that contains (u_2, u_3) . Let R_1 be the finest equivalence that contains $(s, t), (u_1, u_2), (u_3, u_4)$, and let R_2 be the one that contains $(s, t), (u_1, u_3), (u_2, u_4)$. Both R_1 and R_2 are transitive ε -bisimulations, and since $u_1 \not\sim_\varepsilon u_4$ no other transitive ε -bisimulation can contain (s, t) . Hence, no such relation contains (u_2, u_3) . Let $\mathcal{Q} = \mathcal{M}_s/\sim$ be as on the right of the figure. Then $\mathcal{M}_s \sim \mathcal{Q}$ and $\mathcal{M}_s \simeq_\varepsilon \mathcal{M}_t$ as, e.g., the ε -perturbations \mathcal{M}'_s and \mathcal{M}'_t that enforce $u'_1 \sim u'_2$ and $u'_3 \sim u'_4$ and are otherwise unchanged are bisimilar. However, there are no bisimilar ε -perturbations of \mathcal{M}_t and \mathcal{Q} , i.e., $\mathcal{M}_t \not\simeq_\varepsilon \mathcal{Q}$. Since $\simeq_0 = \sim$ this observation additionally yields that \simeq_ε cannot be additive, as otherwise $\mathcal{M}_s \sim \mathcal{Q}$ and $\mathcal{M}_s \simeq_\varepsilon \mathcal{M}_t$ would have to imply $\mathcal{M}_t \simeq_\varepsilon \mathcal{Q}$. All in all, this leads to the following result.



■ **Figure 9** An LMC that demonstrates that \simeq_ε is strictly finer than \sim_ε^* .



■ **Figure 10** The LMCs \mathcal{M}_2 (left) and \mathcal{N}_2 (right), as used in the proof of Theorem 4.8.

► **Proposition 4.4.** *The relation \simeq_ε is not additive in the tolerances and can distinguish bisimilar LMCs in the following sense: there are LMCs $\mathcal{M}_1, \mathcal{M}_2$ and \mathcal{N} such that $\mathcal{M}_1 \sim \mathcal{M}_2$ and $\mathcal{M}_1 \simeq_\varepsilon \mathcal{N}$, but $\mathcal{M}_2 \not\simeq_\varepsilon \mathcal{N}$.*

This behavior of \simeq_ε is in contrast to, e.g., \sim_ε , as $s_1 \sim s_2$ and $s_1 \sim_\varepsilon t$ always implies $s_2 \sim_\varepsilon t$. In particular, the non-additivity does not hinge on the existence of bisimilar states in the model. To see this consider, e.g., slight perturbations \mathcal{M}'_s and \mathcal{M}'_t of the LMCs on the left of Figure 8, where for some $\delta < \varepsilon$ we set $P(u_2)(v) = \frac{1}{2} - \varepsilon - \delta$ and $P(u_2)(w) = \frac{1}{2} + \varepsilon + \delta$, and leave the rest of the models unchanged. Then $u_2 \sim u_3$ in \mathcal{M}'_s and \mathcal{M}'_t , but still $\mathcal{M}'_s \simeq_\delta \mathcal{Q}$ and $\mathcal{M}'_s \simeq_\varepsilon \mathcal{M}'_t$ while $\mathcal{M}'_t \not\simeq_{\varepsilon+\delta} \mathcal{Q}$, where \mathcal{Q} is again the (unperturbed) LMC on the right of the figure. Similar results hold for \sim_ε^* , as $\mathcal{M}_s \sim_\varepsilon^* \mathcal{M}_t$ and $\mathcal{M}_s \sim \mathcal{Q}$, but $\mathcal{M}_t \not\sim_\varepsilon^* \mathcal{Q}$.

We now discuss how \simeq_ε relates to \sim_ε and \sim_ε^* , starting with the direction from left to right. From [39] it follows directly that $\mathcal{M} \simeq_\varepsilon \mathcal{N}$ implies $\mathcal{M} \sim_\varepsilon \mathcal{N}$. As we show next, the claim also holds when considering the stronger requirement of *transitive* ε -bisimilarity.

► **Lemma 4.5.** *$\mathcal{M} \simeq_\varepsilon \mathcal{N}$ implies $\mathcal{M} \sim_\varepsilon^* \mathcal{N}$.*

It is thus possible to transfer known results for \sim_ε like, e.g., the preservation of approximate satisfaction of bounded PCTL state formulas [13], the exact preservation of ε -robust PCTL [27], or the bounds on finite horizon [14] and syntactically co-safe [31, 32] LTL satisfaction probabilities to ε -perturbed bisimilar LMCs.

Regarding the reverse implication, consider Figure 9. There, the finest equivalence that relates u_1, u_2 and u_3 and contains (s, t) is a transitive ε -bisimulation. However, there is no ε -perturbation of the LMC in which s and t are bisimilar. Hence, $s \sim_\varepsilon^* t$, but $s \not\simeq_\varepsilon t$.

► **Lemma 4.6.** *\simeq_ε is strictly finer than \sim_ε^* .*

In fact, ε -bisimilarity is not even guaranteed to imply δ -perturbed bisimilarity if $\varepsilon \ll \delta$, or if the Markov chains in question are graph-isomorphic.

► **Theorem 4.7.** *Let $\varepsilon \in (0, \frac{1}{4}]$. There are LMCs \mathcal{M} and \mathcal{N} with $\mathcal{M} \sim_\varepsilon \mathcal{N}$ but $\mathcal{M} \not\simeq_{\frac{1}{4}} \mathcal{N}$.*

► **Theorem 4.8.** *There is a family $\mathcal{F} = \{(\mathcal{M}_n, \mathcal{N}_n) \mid n \in \mathbb{N}_{\geq 1}\}$ of pairs of finite LMCs such that, for all $n \in \mathbb{N}_{\geq 1}$ and $\varepsilon \in \left(0, \frac{1}{n \cdot (n+1)^2}\right]$, \mathcal{M}_n and \mathcal{N}_n are graph-isomorphic and ε -bisimilar, but $\mathcal{M}_n \not\approx_{\delta} \mathcal{N}_n$ for any $\delta < n\varepsilon$.*

Proof sketch. We sketch the case $n = 2$, with \mathcal{M}_2 and \mathcal{N}_2 as in Figure 10, $\varepsilon \in \left(0, \frac{1}{18}\right]$ and \sim_{ε} the symmetric and reflexive closure of $\{(s, t), (s_1, t_1), (s_1, t_2), (s_2, t_2), (s_2, t_3), (s_3, t_3), (x, y)\}$. Any bisimilar perturbations \mathcal{M}'_2 and \mathcal{N}'_2 must ensure $s' \sim t'$. The smallest (w.r.t. the required tolerances) perturbations that achieve this make s'_2, s'_3 and t'_3 , as well as s'_1, t'_1 and t'_2 , bisimilar, and set the total probability mass from s' (resp. t') to reach these (sets of) state(s) to $\frac{1}{2}$ each. But this requires a perturbation by at least $\delta = \frac{1}{9} \geq 2\varepsilon$. ◀

► **Remark 4.9.** Theorems 4.7 and 4.8 seem to resemble results of [38, 39]. There, an LMC is presented in which a specific order of merging ε -bisimilar states results in an approximate quotient that requires tolerance $\geq \frac{1}{4}$, and a family of LMCs is provided [39, Thm. 12] in which merging ε -bisimilar states yields an approximate quotient that requires tolerance $\geq n\varepsilon$. Our results differ in that we consider the existence of bisimilar ε -perturbations of two LMCs, and in that we show that no suitable *smaller* tolerance exists.

The observation that \simeq_{ε} is strictly finer than \sim_{ε} (and even \sim_{ε}^*) raises the question whether there are logical properties which are preserved under \simeq_{ε} , but not necessarily under \sim_{ε}^* . It is future work to make this precise. Here, we note that the bound for reachability probabilities from Theorem 3.11 remains tight under \simeq_{ε} : the LMCs \mathcal{M} and \mathcal{N} in Figure 6 satisfy $\mathcal{M} \simeq_{\varepsilon} \mathcal{N}$, but the bounds are tight by Example 3.13.

The following theorem characterizes \simeq_{ε} in terms of transitive ε -bisimulations that satisfy an additional *centroid property* specified as in Equation (1) below.

► **Theorem 4.10.** *The following statements are equivalent:*

- (i) $\mathcal{M} \simeq_{\varepsilon} \mathcal{N}$.
- (ii) *There is an ε -perturbation of $\mathcal{M} \oplus \mathcal{N}$ in which $s_{init}^{\mathcal{M}} \sim s_{init}^{\mathcal{N}}$.*
- (iii) *There is a transitive ε -bisimulation R on $\mathcal{M} \oplus \mathcal{N}$ with $(s_{init}^{\mathcal{M}}, s_{init}^{\mathcal{N}}) \in R$ such that for each $A \in S/R$, where S is the disjoint union of $S^{\mathcal{M}}$ and $S^{\mathcal{N}}$, there is a $P_A^* \in \text{Distr}(S/R)$ with*

$$|P(s)(C) - P_A^*(C)| \leq \frac{\varepsilon}{2} \text{ for all } s \in A \text{ and all } R\text{-closed sets } C. \quad (1)$$

From the next lemma it follows immediately that, for a given equivalence $R \subseteq S \times S$, the centroid property in Equation (1) can be checked efficiently.

► **Lemma 4.11.** *For a finite set X and $\mu_1, \dots, \mu_k \in \text{Distr}(X)$, the following are equivalent:*

- (i) *There exists $\mu^* \in \text{Distr}(X)$ with $|\mu_l(B) - \mu^*(B)| \leq \frac{\varepsilon}{2}$ for all $l \in \{1, \dots, k\}$ and $B \subseteq X$.*
- (ii) *There exists $\mu \in \text{Distr}(X)$ with $\|\mu_l - \mu\|_1 \leq \varepsilon$ for all $l \in \{1, \dots, k\}$.*
- (iii) *The following linear constraint system over non-negative variables $\delta_{l,i}$ and x_i for $l \in \{1, \dots, k\}$ and $i \in X$ is solvable:*

$$\sum_{i \in X} x_i = 1 \quad \text{and} \quad x_i - \mu_l(i) \leq \delta_{l,i} \quad \text{and} \quad \mu_l(i) - x_i \leq \delta_{l,i} \quad \text{and} \quad \sum_{i \in X} \delta_{l,i} \leq \varepsilon.$$

The equivalence to (iii) further implies that $\mu^ = \mu$ can be computed in polynomial time.*

However, as we show next, for given \mathcal{M}, \mathcal{N} and ε it is NP-complete to decide if $\mathcal{M} \simeq_{\varepsilon} \mathcal{N}$ and if $\mathcal{M} \sim_{\varepsilon}^* \mathcal{N}$. This stands in contrast to the polynomial time computability of \sim_{ε} [25], which is possible in $\mathcal{O}(|S|^7)$ by iteratively solving maximum flow problems. Our proofs are inspired by [39, Thm. 1], which proves that deciding if a LMC has an ε -quotient with a fixed number of states is NP-complete.

► **Theorem 4.12.** *For given finite LMCs \mathcal{M} and \mathcal{N} and given $\varepsilon \in (0, 1]$, it is NP-complete to decide if (i) $\mathcal{M} \simeq_\varepsilon \mathcal{N}$ and to decide if (ii) $\mathcal{M} \sim_\varepsilon^* \mathcal{N}$.*

Nevertheless, one can check in polynomial time if a *given* equivalence R is a transitive ε -bisimulation or an ε -perturbed bisimulation. Since constructing quotients w.r.t. these relations by collapsing equivalence classes into single states can be done efficiently as well, the notions are therefore still suitable for constructing abstractions in practical applications.

► **Proposition 4.13.** *Given an equivalence R , one can decide in polynomial time if (i) R is a transitive ε -bisimulation and if (ii) R is an ε -perturbed bisimulation.*

5 Branching and Weak ε -Bisimulation

We now introduce approximate versions of branching and weak probabilistic bisimulation. A similar approach has been discussed sporadically in the context of noninterference under the term “weak bisimulation with precision ε ” [4, 7, 5, 6, 26, 3]. While our notion of branching ε -bisimilarity is a branching variant of transitive ε -bisimilarity \sim_ε^* , the weak ε -bisimilarity we propose is a weak variant of \sim_ε . Hence, the former is tailored to the construction of quotients of a given model, while the latter is closer to classic process relations.

► **Definition 5.1.** *An equivalence $R \subseteq S \times S$ is a branching ε -bisimulation if for all $(s, t) \in R$ and all R -closed sets $A \subseteq S$ it holds that*

$$(i) \ l(s) = l(t) \quad \text{and} \quad (ii) \ |\Pr_s([s]_R \mathbf{U}A) - \Pr_t([t]_R \mathbf{U}A)| \leq \varepsilon.$$

We call $s, t \in S$ branching ε -bisimilar, written $s \approx_\varepsilon^b t$, if they are related by a branching ε -bisimulation. LMCs \mathcal{M} and \mathcal{N} are branching ε -bisimilar, written $\mathcal{M} \approx_\varepsilon^b \mathcal{N}$, if $s_{init}^{\mathcal{M}} \approx_\varepsilon^b s_{init}^{\mathcal{N}}$ in $\mathcal{M} \oplus \mathcal{N}$.

We require branching ε -bisimulations to be equivalences, as their goal is to abstract from stutter steps inside a state’s equivalence class. Because of transitivity, Definition 5.1 can also be formulated in the style of Definition 3.1 and should thus not be understood as an explicit extension of Definition 3.4. With the same arguments as for \sim_ε^* and \simeq_ε , transitivity causes that there may not be a unique maximal branching ε -bisimulation, that \approx_ε^b is not additive in the tolerances, and that it can differentiate bisimilar models: the first claim follows from $s \approx_\varepsilon^b t$ and $t \approx_\varepsilon^b u$ but $s \not\approx_\varepsilon^b u$ in Figure 7, the others from $\sim_\varepsilon^* = \approx_\varepsilon^b$ in Figure 8.

► **Definition 5.2.** *A reflexive and symmetric relation $R \subseteq S \times S$ is a weak ε -bisimulation if for all $(s, t) \in R$ and all $A \subseteq S$ it holds that*

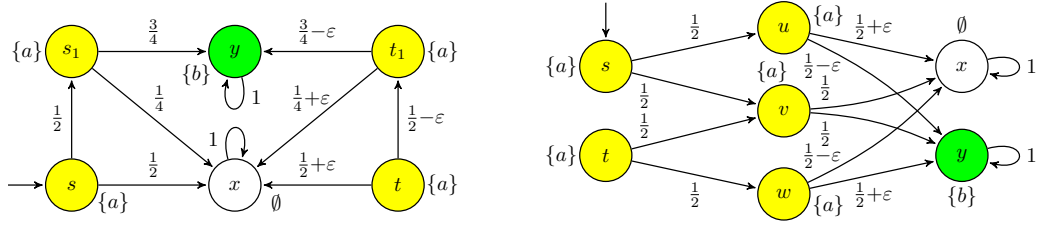
$$(i) \ l(s) = l(t) \quad \text{and} \quad (ii) \ \Pr_s(L(s)\mathbf{U}A) \leq \Pr_t(L(t)\mathbf{U}R(A)) + \varepsilon.$$

We call $s, t \in S$ weakly ε -bisimilar, written $s \approx_\varepsilon^w t$, if they are related by a weak ε -bisimulation. LMCs \mathcal{M} and \mathcal{N} are weakly ε -bisimilar, written $\mathcal{M} \approx_\varepsilon^w \mathcal{N}$, if $s_{init}^{\mathcal{M}} \approx_\varepsilon^w s_{init}^{\mathcal{N}}$ in $\mathcal{M} \oplus \mathcal{N}$.

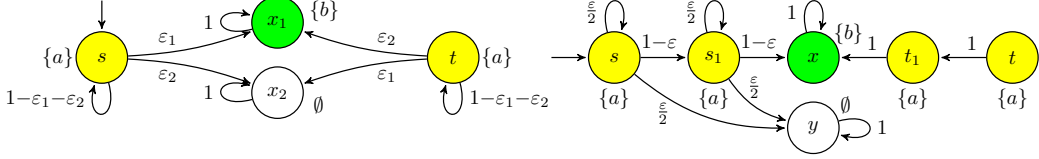
In contrast to branching ε -bisimulations, we do not require transitivity for weak ε -bisimulations. As it turns out, \approx_ε^w is instead additive in the tolerances.

► **Lemma 5.3.** *$s \approx_\varepsilon^w t$ and $t \approx_\delta^w u$ implies $s \approx_{\varepsilon+\delta}^w u$.*

Further, \approx_0^w and \approx_0^b coincide with \approx^w and \approx^b , respectively, so our notions are conservative extensions of their exact counterparts. In particular, as $\approx^w = \approx^b$ for LMCs [10], it follows that $\approx_0^w = \approx_0^b$. For $\varepsilon > 0$ the notions can, however, become incomparable. This is different compared to the nonprobabilistic case, where \approx^b is strictly finer than \approx^w [51].



■ **Figure 11** The LMCs used in the proof of Proposition 5.4.



■ **Figure 12** The LMCs used in the proof of (i) of Proposition 5.5.

► **Proposition 5.4.** For $0 < \varepsilon < \frac{1}{4}$, $s \approx_\varepsilon^b t \not\approx s \approx_\varepsilon^w t$ and $s \approx_\varepsilon^w t \not\approx s \approx_\varepsilon^b t$.

Proof. Let $\varepsilon \in (0, \frac{1}{4})$ and consider Figure 11. In the left LMC, $s \approx_\varepsilon^b t$, as the largest branching ε -bisimulation is induced by the equivalence classes $\{\{s, t\}, \{s_1, t_1\}, \{x\}, \{y\}\}$ and, in particular, $s \not\approx_\varepsilon^b s_1$ and $t \not\approx_\varepsilon^b t_1$. However, $s \not\approx_\varepsilon^w t$ as $\Pr_t(L(t) \cup \{x\}) = \frac{5}{8} + \frac{5}{4}\varepsilon - \varepsilon^2 > \frac{5}{8} + \varepsilon = \Pr_s(L(s) \cup \{x\}) + \varepsilon$. Furthermore, in the right LMC, $s \approx_\varepsilon^w t$ while $s \not\approx_\varepsilon^b t$ since any branching ε -bisimulation R that contains (s, t) must also contain (u, w) due to transitivity, which is not possible as, e.g., $|\Pr_u([u]_R \cup [x]_R) - \Pr_w([w]_R \cup [x]_R)| > \varepsilon$. ◀

The major difference between \approx_ε^w , \approx_ε^b and $\sim_\varepsilon, \equiv_\varepsilon$ is that the former can abstract from (some) stutter steps. Consequently, if no stuttering is possible, i.e., when $P(s)(L(s)) = 0$ for all $s \in S$, we have $\sim_\varepsilon^* = \approx_\varepsilon^b$ and $\sim_\varepsilon = \approx_\varepsilon^w$. Otherwise, the notions become incomparable.

► **Proposition 5.5.** Let $\approx_\varepsilon \in \{\approx_\varepsilon^b, \approx_\varepsilon^w\}$. Then there are LMCs with states $s, t \in S$ such that (i) $s \sim_\varepsilon t$ and $s \equiv_\varepsilon t$ but $s \not\approx_\varepsilon t$, and (ii) $s \approx_\varepsilon t$ but $s \not\sim_\varepsilon t$ and $s \not\equiv_\varepsilon t$. Hence, \approx_ε and $\sim_\varepsilon, \equiv_\varepsilon$ are incomparable. Furthermore, (i) and (ii) also hold for \sim_ε^* and \equiv_ε^* instead of \sim_ε and \equiv_ε .

Proof. To show (i) we do a case distinction on \approx_ε . If $\approx_\varepsilon = \approx_\varepsilon^b$, consider the LMC on the left of Figure 12 where $\varepsilon_1, \varepsilon_2 \in (0, 1)$, $\varepsilon_1 \neq \varepsilon_2$, $\varepsilon_1 + \varepsilon_2 < 1$, and $\varepsilon = |\varepsilon_1 - \varepsilon_2|$. In this model, both $s \sim_\varepsilon t$ and $s \equiv_\varepsilon t$. However, for any equivalence R that only relates states with the same label, $|\Pr_s([s]_R \cup \{x_1\}) - \Pr_t([t]_R \cup \{x_1\})| = \frac{|\varepsilon_1 - \varepsilon_2|}{\varepsilon_1 + \varepsilon_2} \varepsilon^{1 + \varepsilon_2} < |\varepsilon_1 - \varepsilon_2| = \varepsilon$, so $s \not\approx_\varepsilon^b t$.

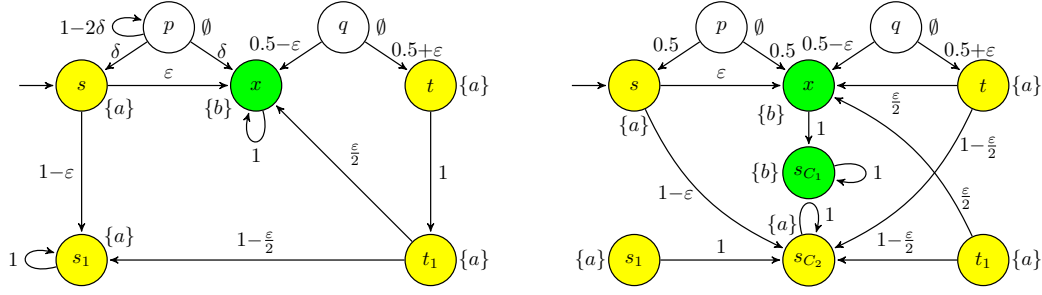
If $\approx_\varepsilon = \approx_\varepsilon^w$, consider the right of Figure 12 with $\varepsilon \in (0, 1)$. There, $s \sim_\varepsilon t$ and $s \equiv_\varepsilon t$. However, $\Pr_t(L(t) \cup \{x\}) = 1 > \frac{4(1-\varepsilon)^2}{(2-\varepsilon)^2} = \Pr_s(L(s) \cup \{x\})$ for all $\varepsilon \in (0, 1)$, so $s \not\approx_\varepsilon^w t$.

The second claim follows when considering an LMC with three states, say s, t and x , with initial state s and $l(s) = l(t) \neq l(x)$ as well as $P(s)(t) = P(t)(x) = P(x)(x) = 1$. There, $s \approx_\varepsilon^b t$ and $s \approx_\varepsilon^w t$ for any ε , but neither $s \equiv_\varepsilon t$ nor $s \sim_\varepsilon t$.

The claims are shown analogously when replacing \sim_ε and \equiv_ε with \sim_ε^* resp. \equiv_ε^* . ◀

Note that the anomaly of \equiv_ε described in Example 3.5 does not occur for branching ε -bisimilarity, as here transitivity would enforce $u_i \approx_\varepsilon^b u_j$ for all i, j in Figure 3 if $s \approx_\varepsilon^b t$.

The next lemma bounds the probabilities of states related by \approx_ε^b or \approx_ε^w to stutter forever.



■ **Figure 13** An LMC \mathcal{M} (left) and its transformation \mathcal{M}_R (right) w.r.t. the branching ε -bisimulation R with equivalence classes $\{s, t, s_1, t_1\}, \{p, q\}, \{x\}$ for $0 < \varepsilon < 1 - 2\delta$ and $div_R = \{\{s, t, s_1, t_1\}, \{x\}\}$.

► **Lemma 5.6.** *Let $\varepsilon \in [0, 1]$ and let R be a branching ε -bisimulation.*

1. *If $(s, t) \in R$ and $C = [s]_R = [t]_R$ then $|\Pr_s(\Box C) - \Pr_t(\Box C)| \leq \varepsilon$.*
2. *If \mathcal{M} is finite and $(s, t) \in R$ then, for any $C \in S/R$, either (i) $\Pr_s(\Box C) = 0$ or (ii) $\Pr_s(\Box C) \geq 1 - \varepsilon$ for all $s \in C$.*
3. *If $s \approx_\varepsilon^w t$ and $b = l(s) = l(t)$ then $|\Pr_s(\Box b) - \Pr_t(\Box b)| \leq \varepsilon$.*

Since \approx_ε^b and \approx_ε^w cannot differentiate single steps from steps after an arbitrary (but finite) amount of stuttering, they do not preserve any next-step probabilities. Furthermore, in Figure 4 both $s \approx_\varepsilon^b t$ and $s \approx_\varepsilon^w t$, so by Example 3.9 we cannot expect a better bound for finite horizon satisfaction probabilities in related states than the one from [14] stated in Theorem 3.8. We can, however, extend Theorem 3.11 to states related by \approx_ε^b and \approx_ε^w .

Given an equivalence R on a finite LMC \mathcal{M} , let $div_R \subseteq S/R$ be the set of *divergent* R -equivalence classes, i.e., $C \in div_R$ iff $\Pr_s(\Box C) \geq 1 - \varepsilon$ for all $s \in C$. We construct from \mathcal{M} an LMC \mathcal{M}_R and an equivalence R^b on \mathcal{M}_R with $R \subseteq R^b$. Intuitively, \mathcal{M}_R is obtained from \mathcal{M} by redirecting the probabilities $\Pr_s(\Box C)$ for $C = [s]_R$ to fresh “divergence states” s_C .

► **Definition 5.7.** *Given a finite LMC \mathcal{M} and an equivalence R that only relates states with the same label, let $\mathcal{M}_R = (S_R, P_R, s_{init}, l_R)$ with*

- $S_R = S \cup \{s_C \mid C \in div_R\}$ where the s_C are fresh, pairwise different states
- $l_R(s) = l(s)$ if $s \in S$ and $l(s_C) = l(s)$ for some $s \in C$
- for $s \in S$ and $C = [s]_R$, the values of the distribution $P_R(s)$ are defined by

$$P_R(s)(t) = \begin{cases} \Pr_s(CUt), & \text{if } t \in S \setminus C \\ \Pr_s(\Box C), & \text{if } s \neq s_C \text{ and } t = s_C \\ 1, & \text{if } s = t = s_C \\ 0, & \text{otherwise} \end{cases}.$$

An example for the transformation from \mathcal{M} to \mathcal{M}_R can be found in Figure 13. We now show the connection between branching ε -bisimulations R on finite LMCs \mathcal{M} and transitive ε -bisimulations on their transformations \mathcal{M}_R .

► **Lemma 5.8.** *Let \mathcal{M} be finite, R an equivalence relating only equally labeled states, and R^b the finest equivalence on S_R with $R \subseteq R^b$ and $(s, s_C) \in R^b$ for all $C \in div_R$ and $s \in C$. Then R is a branching ε -bisimulation on \mathcal{M} iff R^b is a transitive ε -bisimulation on \mathcal{M}_R .*

It is clear from the definition of \mathcal{M}_R that for every $C \in S/R$ and all $s \in S$ with $s \notin C$ we have $\Pr_s^{\mathcal{M}}([s]_R UC) = P_R(s)(C)$. Hence, Lemma 5.8 allows us to transfer Theorem 3.11 to states $s \approx_\varepsilon^b t$, since they are ε -bisimilar in \mathcal{M}_R . As in \mathcal{M}_R any transition from s to a $u \in S$ represents an equivalence class change in \mathcal{M} , the random variable N^b now has to count the number of *equivalence class changes* on paths to a g - or f -labeled state.

► **Corollary 5.9.** *Let \mathcal{M} be finite, let some states in \mathcal{M} be labeled with g , and let exactly the states that cannot reach a g -labeled state be labeled with f . Further, let $s \approx_\varepsilon^b t$, and let N^b denote the random variable that counts the number of equivalence class changes until a g - or f -labeled state is reached. Then $|\Pr_s(\diamond g) - \Pr_t(\diamond g)| \leq \varepsilon \cdot \mathbb{E}_s(N^b)$.*

Furthermore, it is possible to extend Theorem 3.11 to weakly ε -bisimilar states.

► **Proposition 5.10.** *Let \mathcal{M} , f and g be as in Corollary 5.9, let $s \approx_\varepsilon^w t$ and let N^w denote the random variable that counts the number of label changes until a g - or f -labeled state is reached. Then $|\Pr_s(\diamond g) - \Pr_t(\diamond g)| \leq \varepsilon \cdot \mathbb{E}_s(N^w)$.*

Proof sketch. Let $\mathcal{L} = \{b \in 2^{AP} \mid \exists s \in S: \Pr_s(\square b) > 0\}$. From \mathcal{M} we construct an LMC \mathcal{M}^w , almost similar to \mathcal{M}_R in Definition 5.7. The main differences are that we introduce fresh states s_b for all $b \in \mathcal{L}$, and that we set $P^w(s)(t) = \Pr_s(L(s)Ut)$ for all $s, t \in S$ with $l(s) \neq l(t)$ as well as $P^w(s)(s_b) = \Pr_s(\square b)$ if $l(s) = b \in \mathcal{L}$. Because for any weak ε -bisimulation R the finest reflexive and symmetric relation R^w on \mathcal{M}^w with $R \subseteq R^w$ and $(s, s_b) \in R^w$ iff $b = l(s)$ and $\Pr_s(\square b) \geq 1 - \varepsilon$ is an ε -bisimulation on \mathcal{M}^w , the result follows from Theorem 3.11. ◀

As the LMCs in Figure 6 are both branching $\frac{\varepsilon}{p}$ -bisimilar and weak $\frac{\varepsilon}{p}$ -bisimilar, and since in these models $\mathbb{E}_s(N^b) = \mathbb{E}_s(N^w) = 1$, the bounds are again tight by Example 3.13.

We finish this section by analyzing the complexity of deciding if two given states s, t are branching ε -bisimilar, i.e., if $s \approx_\varepsilon^b t$. The analogous problem for \approx_ε^w is left open.

► **Theorem 5.11.** *Given a finite \mathcal{M} , $s, t \in S$, and $\varepsilon \in (0, 1]$, deciding if $s \approx_\varepsilon^b t$ is NP-complete.*

6 Conclusion and Future Work

We investigated several new types of approximate probabilistic bisimulation and showed how they interrelate, as well as how they are connected to notions from the literature like, e.g., \sim_ε and \equiv_ε (see Figure 1). These connections in turn allowed the transfer of known preservation results for logical formulas between the different notions, which we extended by tight bounds on the absolute difference of unbounded reachability probabilities in weak and branching ε -bisimilar states. Additionally, we established complexity results for most of our relations.

The results of Section 4 indicate that ε -perturbed bisimilarity \simeq_ε and transitive ε -bisimilarity \sim_ε^* show some anomalies (lack of additivity, the possibility to differentiate bisimilar models and the fact that they themselves are not necessarily an ε -perturbed resp. a transitive ε -bisimulation) when viewed as process relations. However, both relations can be interesting for algorithmic purposes as they permit efficient quotienting techniques: given a transitive ε -bisimulation R (with or without the centroid property) on an LMC \mathcal{M} , one can build in polynomial time a quotient LMC that arises from \mathcal{M} by collapsing all R -equivalence classes into single states. The quotient under an ε -perturbed bisimulation R_1 enjoys the property that every state s and its R_1 -equivalence class $[s]_{R_1}$ are $\frac{\varepsilon}{2}$ -bisimilar [38], while for the quotients under a transitive ε -bisimulation R_2 that lacks the centroid property we can only guarantee $s \sim_\varepsilon [s]_{R_2}$. On the other hand, transitive ε -bisimulations can identify more states and hence can induce smaller quotients.

Similarly, the transitivity of branching ε -bisimulations causes the same anomalies as for \simeq_ε and \sim_ε^* . However, checking if a given equivalence is a branching ε -bisimulation and constructing a corresponding quotient is again possible in polynomial time. Hence, investigating the potential of transitive (or branching) ε -bisimulations as abstraction techniques for an approximate analysis of LMCs in practice is an interesting future research direction.

Other open questions include the search for a characterization of logical formulas that distinguish $\sim_\varepsilon, \sim_\varepsilon^*, \approx_\varepsilon^w, \approx_\varepsilon^b$ and \simeq_ε , and how our results relate to bisimilarity distances [49].


References

- 1 Alessandro Abate. Approximation metrics based on probabilistic bisimulations for general state-space Markov processes: A survey. *Electronic Notes in Theoretical Computer Science*, 297:3–25, 2013. Proceedings of the first workshop on Hybrid Autonomous Systems.
- 2 Alessandro Abate, Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic model checking of labelled Markov processes via finite approximate bisimulations. In Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten, editors, *Horizons of the Mind. A Tribute to Prakash Panangaden: Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, volume 8464 of *Lecture Notes in Computer Science (LNCS)*, pages 40–58. Springer International Publishing, Cham, 2014.
- 3 Alessandro Aldini. A note on the approximation of weak probabilistic bisimulation, 2009.
- 4 Alessandro Aldini, Mario Bravetti, Alessandra Di Pierro, Roberto Gorrieri, Chris Hankin, and Herbert Wiklicky. Two formal approaches for approximating noninterference properties. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design II (FOSAD 2001)*, volume 2946 of *Lecture Notes in Computer Science (LNCS)*, pages 1–43. Springer, Berlin, Heidelberg, 2004.
- 5 Alessandro Aldini, Mario Bravetti, and Roberto Gorrieri. A process-algebraic approach for the analysis of probabilistic noninterference. *J. Comput. Secur.*, 12(2):191–245, April 2004.
- 6 Alessandro Aldini and Alessandra Di Pierro. A quantitative approach to noninterference for probabilistic systems. *Electronic Notes in Theoretical Computer Science*, 99:155–182, 2004. Proceedings of the MEFISTO Project 2003, Formal Methods for Security and Time.
- 7 Alessandro Aldini and Roberto Gorrieri. Security analysis of a probabilistic non-repudiation protocol. In Holger Hermanns and Roberto Segala, editors, *Process Algebra and Probabilistic Methods: Performance Modeling and Verification*, volume 2399 of *Lecture Notes in Computer Science (LNCS)*, pages 17–36. Springer, Berlin, Heidelberg, 2002.
- 8 Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In Pierre Wolper, editor, *Computer Aided Verification (CAV 1995)*, volume 939 of *Lecture Notes in Computer Science (LNCS)*, pages 155–165, Berlin, Heidelberg, 1995. Springer, Berlin, Heidelberg.
- 9 Christel Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science (LNCS)*, pages 50–61. Springer, Berlin, Heidelberg, 1996.
- 10 Christel Baier and Holger Hermanns. Weak bisimulation for fully probabilistic processes. In Orna Grumberg, editor, *Computer Aided Verification (CAV 1997)*, volume 1254 of *Lecture Notes in Computer Science (LNCS)*, pages 119–130. Springer, Berlin, Heidelberg, 1997.
- 11 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- 12 Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Verena Wolf. Comparative branching-time semantics for Markov chains. *Information and Computation*, 200(2):149–214, 2005.
- 13 Massimo Bartoletti, Maurizio Murgia, and Roberto Zunino. Sound approximate and asymptotic probabilistic bisimulations for PCTL. *Logical Methods in Computer Science*, 19(1), 2023.
- 14 Gaoang Bian and Alessandro Abate. On the relationship between bisimulation and trace equivalence in an approximate probabilistic context. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 10203 of *Lecture Notes in Computer Science (LNCS)*, pages 321–337. Springer, Berlin, Heidelberg, 2017.
- 15 Gaoang Bian and Alessandro Abate. On the relationship between bisimulation and trace equivalence in an approximate probabilistic context (extended version). *CoRR*, abs/1701.04547, 2017. Full version of [14]. [arXiv:1701.04547](https://arxiv.org/abs/1701.04547).
- 16 Béla Bollobás and Nicolas Th. Varopoulos. Representation of systems of measurable sets. *Mathematical Proceedings of the Cambridge Philosophical Society*, 78(2):323–325, 1975.
- 17 Peter Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, January 1995.

- 18 David de Frutos-Escrig, Jeroen J. A. Keiren, and Tim A. C. Willemse. Games for bisimulations and abstraction. *CoRR*, abs/1611.00401, 2016. [arXiv:1611.00401](https://arxiv.org/abs/1611.00401).
- 19 Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal state-space lumping in Markov chains. *Information Processing Letters*, 87(6):309–315, 2003.
- 20 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled Markov systems. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR'99 Concurrency Theory (CONCUR 1999)*, volume 1664 of *Lecture Notes in Computer Science (LNCS)*, pages 258–273. Springer, Berlin, Heidelberg, 1999.
- 21 Josée Desharnais, Abbas Edalat, and Prakash Panangaden. A logical characterization of bisimulation for labeled Markov processes. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science (LiCS)*, pages 478–487, 1998.
- 22 Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179(2):163–193, 2002.
- 23 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Approximating labelled Markov processes. *Information and Computation*, 184(1):160–200, 2003.
- 24 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Weak bisimulation is sound and complete for pCTL*. *Information and Computation*, 208(2):203–219, 2010.
- 25 Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate analysis of probabilistic processes: Logic, simulation and games. In *Fifth International Conference on Quantitative Evaluation of Systems (QEST 2008)*, pages 264–273, 2008.
- 26 Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Measuring the confinement of probabilistic systems. *Theoretical Computer Science*, 340(1):3–56, 2005. Theoretical Foundations of Security Analysis and Design II.
- 27 Alessandro D’Innocenzo, Alessandro Abate, and Joost-Pieter Katoen. Robust PCTL model checking. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2012)*, pages 275–286, New York, NY, USA, 2012. Association for Computing Machinery.
- 28 András Faragó. On the convergence rate of quasi lumpable Markov chains. In András Horváth and Miklós Telek, editors, *Formal Methods and Stochastic Models for Performance Evaluation*, volume 4054 of *Lecture Notes in Computer Science (LNCS)*, pages 138–147. Springer, Berlin, Heidelberg, 2006.
- 29 Giuliana Franceschinis and Richard R. Muntz. Bounds for quasi-lumpable markow chains. *Perform. Evaluation*, 20:223–243, 1994.
- 30 Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In Manfred Broy and Cliff B. Jones, editors, *Proceedings of the IFIP TC2 Working Conference on Programming Concepts and Methods*, pages 443–458, 1990.
- 31 Sofie Haesaert, Petter Nilsson, and Sadegh Soudjani. Formal multi-objective synthesis of continuous-state mdps. In *2021 American Control Conference (ACC)*, pages 3428–3433, 2021.
- 32 Sofie Haesaert and Sadegh Soudjani. Robust dynamic programming for temporal logic control of stochastic systems. *IEEE Transactions on Automatic Control*, 66(6):2496–2511, 2021.
- 33 Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
- 34 Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The probabilistic model checker storm. *International Journal on Software Tools for Technology Transfer (STTT)*, 24:589–610, 2022.
- 35 David N. Jansen, Jan Friso Groote, Ferry Timmers, and Pengfei Yang. A Near-Linear-Time Algorithm for Weak Bisimilarity on Markov Chains. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 36 Joost-Pieter Katoen. The probabilistic model checking landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LiCS’16)*, pages 31–45, New York, NY, USA, 2016. Association for Computing Machinery.

- 37 Joost-Pieter Katoen, Tim Kemna, Ivan Zapreev, and David N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007)*, volume 4424 of *Lecture Notes in Computer Science (LNCS)*, pages 87–101. Springer, Berlin, Heidelberg, 2007.
- 38 Stefan Kiefer and Qiyi Tang. Approximate Bisimulation Minimisation. In Mikołaj Bojańczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 39 Stefan Kiefer and Qiyi Tang. Approximate bisimulation minimisation. *CoRR*, abs/2110.00326, 2021. Full version of [38]. [arXiv:2110.00326](https://arxiv.org/abs/2110.00326).
- 40 Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification (CAV 1999)*, volume 1633 of *Lecture Notes in Computer Science (LNCS)*, pages 172–183, Berlin, Heidelberg, 1999. Springer, Berlin, Heidelberg.
- 41 Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
- 42 R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., USA, 1989.
- 43 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin, Heidelberg, 1982.
- 44 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 46–57, 1977.
- 45 Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
- 46 Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR '94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science (LNCS)*, pages 481–496. Springer, Berlin, Heidelberg, 1994.
- 47 Mathieu Tracol, Josée Desharnais, and Abir Zhioua. Computing distances between probabilistic automata. In Mieke Massink and Gethin Norman, editors, *Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL 2011), Saarbrücken, Germany, April 1-3, 2011*, volume 57 of *EPTCS*, pages 148–162, 2011.
- 48 Antti Valmari and Giuliana Franceschinis. Simple $O(m \log n)$ time Markov chain lumping. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science (LNCS)*, pages 38–52, Berlin, Heidelberg, 2010. Springer, Berlin, Heidelberg.
- 49 Franck van Breugel. Probabilistic bisimilarity distances. *ACM SIGLOG News*, 4(4):33–51, 2017.
- 50 Rob J. van Glabbeek, Jan Friso Groote, and Erik P. de Vink. A complete axiomatization of branching bisimilarity for a simple process language with probabilistic choice. In Mário S. Alvim, Kostas Chatzikokolakis, Carlos Olarte, and Frank Valencia, editors, *The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy: Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday*, volume 11760 of *Lecture Notes in Computer Science (LNCS)*, pages 139–162, Cham, 2019. Springer International Publishing.
- 51 Rob J. van Glabbeek and W. Peter Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, May 1996.
- 52 Ralf Wimmer and Bernd Becker. Correctness issues of symbolic bisimulation computation for Markov chains. In Bruno Müller-Clostermann, Klaus Eichtler, and Erwin P. Rathgeb, editors, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB&DFT 2010)*, volume 5987 of *Lecture Notes in Computer Science (LNCS)*, pages 287–301. Springer, Berlin, Heidelberg, 2010.

Progress, Justness and Fairness in Modal μ -Calculus Formulae

Myrthe S. C. Spronck ✉ 

Eindhoven University of Technology, The Netherlands

Bas Luttik ✉ 

Eindhoven University of Technology, The Netherlands

Tim A. C. Willemse ✉ 

Eindhoven University of Technology, The Netherlands

Abstract

When verifying liveness properties on a transition system, it is often necessary to discard spurious violating paths by making assumptions on which paths represent realistic executions. Capturing that some property holds under such an assumption in a logical formula is challenging and error-prone, particularly in the modal μ -calculus. In this paper, we present template formulae in the modal μ -calculus that can be instantiated to a broad range of liveness properties. We consider the following assumptions: progress, justness, weak fairness, strong fairness, and hyperfairness, each with respect to actions. The correctness of these formulae has been proven.

2012 ACM Subject Classification Theory of computation \rightarrow Modal and temporal logics

Keywords and phrases Modal μ -calculus, Property specification, Completeness criteria, Progress, Justness, Fairness, Liveness properties

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.38

Related Version *Full Version:* <https://arxiv.org/abs/2407.08060>

Acknowledgements We thank an anonymous reviewer for the observation that weak and strong hyperfairness must be distinguished.

1 Introduction

Formal verification through model checking requires a formalisation of the properties of the modelled system as formulae in some logic, such as LTL [32], CTL [17] or the modal μ -calculus [29]. In this paper, we focus on the modal μ -calculus, a highly expressive logic used in established model checkers such as mCLR2 [10] and CADP [19].

A frequently encountered problem when checking liveness properties is that spurious violations are found, such as paths on which some components never make progress. Often, such paths do not represent realistic executions of the system. It is then a challenge to restrict verification to those paths that do represent realistic system executions. For this, we use completeness criteria [21, 22]: predicates on paths that say which paths are to be regarded as realistic runs of the system. These runs are called complete runs. Examples of completeness criteria are progress, justness and fairness.

It turns out that writing a modal μ -calculus formula for a property being satisfied under a completeness criterion is non-trivial. Since the μ -calculus is a branching-time logic, we cannot separately formalise when a path is complete and when it satisfies the property, and then combine the two formalisations with an implication. Instead, a more intricate integration of both aspects of a path is needed. Our aim is to achieve such an integration for a broad spectrum of liveness properties and establish the correctness of the resulting formulae. To this end, we shall consider a template property that can be instantiated to a plethora of



© Myrthe S. C. Spronck, Bas Luttik, and Tim A. C. Willemse;
licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 38; pp. 38:1–38:22



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

liveness properties and, in particular, covers all liveness property patterns of [16]. Then, we present modal μ -calculus formulae integrating the completeness criteria of progress, justness, weak fairness, strong fairness, and hyperfairness with this template property.

As discussed in [23], for the formulation of realistic completeness criteria it is sometimes necessary to give special treatment to a set of blocking actions, i.e., actions that require cooperation of the environment in which the modelled system operates. Our template formulae are therefore parameterised with a set of blocking actions. We shall see that, given a set of blocking actions, there are two different interpretations of hyperfairness; we call these weak and strong hyperfairness.

Regarding our presented formulae, the progress formula is similar to those commonly used for liveness properties even when completeness is not explicitly considered. Our formulae for justness, weak fairness and weak hyperfairness only subtly differ from each other. We characterise the similarities these three share and give a generic formula that can be adapted to represent all completeness criteria that meet these conditions. Lastly, we observe that strong fairness and strong hyperfairness do not meet these conditions. We give alternative formulae that are significantly more complex. Whether more efficient formulae for these completeness criteria exist remains an open problem.

Modal μ -calculus formulae are often hard to interpret. Accordingly, it is not trivial to see that our formulae indeed express the integration of liveness properties with completeness criteria. We therefore include elaborate correctness proofs in the full version of this paper.

Our work is essentially a generalisation along two dimensions (viz., the completeness criterion and the liveness property) of the works of [34] and [6, 36]. In [34], the tool PASS is presented for automatically translating common property patterns into modal μ -calculus formulae. Some of those patterns integrate an assumption that excludes paths deemed unrealistic, but since the exact assumption is not stated separately, we cannot make a formal comparison with our approach. In [6], a formula for justness is presented, covering one of the properties we cover. This formula forms the basis for our justness, weak fairness and weak hyperfairness formulae. Our formulae for strong fairness and strong hyperfairness are in part inspired by the formula for termination under strong fairness presented in [36].

The organisation of this paper is as follows. In Section 2 we recap the relevant definitions on labelled transition systems, as well as the syntax and semantics of the modal μ -calculus. In Section 3, we motivate our work with an example, and in Section 4 we give the completeness criteria we cover in this paper. In Section 5, we formally identify the class of liveness properties we study and relate it to a popular class of properties. Our template formulae are presented in Section 6, combining the completeness criteria from Section 4 with the property template from Section 5. We give a small application example in Section 7 and discuss the scope of our work in Section 8. Finally, we give our conclusions in Section 9.

2 Preliminaries

We represent models as labelled transition systems (LTSs). In this section, we briefly introduce the relevant definitions on LTSs, as well as the modal μ -calculus.

2.1 Labelled Transition Systems

► **Definition 1.** *An LTS is a tuple $M = (\mathcal{S}, s_{init}, Act, Trans)$ where*

- \mathcal{S} is a set of states,
- $s_{init} \in \mathcal{S}$ is the initial state,
- Act is a set of action labels, also referred to as the alphabet of the LTS, and
- $Trans \subseteq \mathcal{S} \times Act \times \mathcal{S}$ is a transition relation.

In this paper, we only consider finite LTSs, such as the kind used in finite-state model checking. In particular, our formulae are proven correct under the assumption that Act is finite. We write $s \xrightarrow{a} s'$ as shorthand for $(s, a, s') \in Trans$, and for a given transition $t = (s, a, s')$ we write $src(t) = s$, $act(t) = a$ and $trgt(t) = s'$.

For the definitions below, we fix an LTS $M = (\mathcal{S}, s_{init}, Act, Trans)$.

► **Definition 2.** A path is an (alternating) sequence $\pi = s_0 t_1 s_1 t_2 \dots$ of states $s_0, s_1, \dots \in \mathcal{S}$ and transitions $t_1, t_2, \dots \in Trans$. A path must start with a state, and must be either infinite, or end in a state. In the latter case, the end of the path is referred to as the final state. For all $i \geq 0$, t_{i+1} must satisfy $src(t_{i+1}) = s_i$ and $trgt(t_{i+1}) = s_{i+1}$.

We sometimes refer to transitions on a path as steps. We say an action occurs on a path if a transition labelled with that action is on the path. We call a path on which no action in some set α occurs an α -free path. One path can be appended to another: let $\pi' = s'_0 t'_1 s'_1 \dots t'_n s'_n$ and $\pi'' = s''_0 t''_1 s''_1 \dots$, where π' must be finite and π'' may be finite or infinite. Then the path π defined as π'' appended to π' is written as $\pi = \pi' \cdot \pi'' = s'_0 t'_1 s'_1 \dots t'_n s'_n t''_1 s''_1 \dots$. This is only allowed when $s'_n = s''_0$.

► **Definition 3.** We say that:

- A transition $t \in Trans$ is enabled in a state $s \in \mathcal{S}$ if, and only if, $src(t) = s$.
- An action $a \in Act$ is enabled in a state $s \in \mathcal{S}$ if, and only if, there exists a transition $t \in Trans$ with $act(t) = a$ that is enabled in s .
- An action $a \in Act$ is perpetually enabled on a path π if a is enabled in every state of π .
- An action $a \in Act$ is relentlessly enabled on a path π if every suffix of π contains a state in which a is enabled.
- A state without enabled actions is called a deadlock state.

Every action that is perpetually enabled on a path is also relentlessly enabled on that path.

2.2 Modal μ -Calculus

The modal μ -calculus is given in [29]. Our presentation of the logic is based on [7, 8, 9, 26].

The syntax of the modal μ -calculus is described by the following grammar, in which a ranges over the set of actions Act , and X ranges over a set of formal variables Var .

$$\phi, \psi ::= \text{ff} \mid X \mid \neg\phi \mid \phi \vee \psi \mid \langle a \rangle \phi \mid \mu X. \phi$$

Here ff is false; \neg represents negation; \vee is disjunction; $\langle \rangle$ is the diamond operator; and μ is the least fixpoint operator. We say that $\mu X. \phi$ binds X in ϕ . Variables that are unbound in a formula are free, and a formula without free variables is closed.

A modal μ -calculus formula ϕ must both adhere to this grammar and be *syntactically monotonic*, meaning that for every occurrence of $\mu X. \psi$ in ϕ , every free occurrence of X in ψ must always be preceded by an even number of negations.

We give the semantics of a modal μ -calculus formula ϕ with respect to an arbitrary LTS $M = (\mathcal{S}, s_{init}, Act, Trans)$ and environment $e : Var \rightarrow 2^{\mathcal{S}}$.

$$\begin{aligned} \llbracket \text{ff} \rrbracket_e^M &= \emptyset & \llbracket \phi \vee \psi \rrbracket_e^M &= \llbracket \phi \rrbracket_e^M \cup \llbracket \psi \rrbracket_e^M \\ \llbracket X \rrbracket_e^M &= e(X) & \llbracket \langle a \rangle \phi \rrbracket_e^M &= \left\{ s \in \mathcal{S} \mid \exists s' \in \mathcal{S}. s \xrightarrow{a} s' \wedge s' \in \llbracket \phi \rrbracket_e^M \right\} \\ \llbracket \neg\phi \rrbracket_e^M &= \mathcal{S} \setminus \llbracket \phi \rrbracket_e^M & \llbracket \mu X. \phi \rrbracket_e^M &= \bigcap \left\{ \mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq \llbracket \phi \rrbracket_{e[X:=\mathcal{S}']}^M \right\} \end{aligned}$$

In contexts where the model is fixed, we drop the M from $\llbracket \phi \rrbracket_e^M$. Additionally, we drop e when the environment does not affect the semantics of the formula, e.g. with closed formulae.

We use conjunction, \wedge , and implication, \Rightarrow , as the usual abbreviations. We also add several abbreviations: $tt = \neg ff$ for true; $[a]\phi = \neg\langle a\rangle\neg\phi$ for the box operator; and $\nu X.\phi = \neg\mu X.(\neg\phi[X := \neg X])$ for the greatest fixpoint.

To express formulae more compactly, we extend our syntax to allow regular expressions over finite sets of actions to be used in the box and diamond operators. Since we limit this to finite sets of actions, the syntactical extension does not increase the expressivity of the logic, it merely simplifies the presentation. This is a common extension of the μ -calculus syntax, for instance shown in [26], based on the operators defined for PDL [18]. We overload the symbol for a single action to also represent the singleton set containing that action. We use union, intersection, set difference, and set complement to describe sets of actions as usual. Regular expressions over sets of actions, henceforth referred to as *regular formulae*, are defined by the following grammar:

$$R, Q ::= \varepsilon \mid \alpha \mid R \cdot Q \mid R + Q \mid R^*$$

The empty sequence is represented by ε , and α ranges over sets of actions. The symbol \cdot represents concatenation, $+$ the union of formulae, and * is closure under repetition.

We define the meaning of the diamond operator over the new regular formulae as abbreviations of standard modal μ -calculus formulae:

$$\begin{aligned} \langle \varepsilon \rangle \phi &= \phi & \langle \alpha \rangle \phi &= \bigvee_{a \in \alpha} \langle a \rangle \phi & \langle R \cdot Q \rangle \phi &= \langle R \rangle \langle Q \rangle \phi \\ \langle R + Q \rangle \phi &= \langle R \rangle \phi \vee \langle Q \rangle \phi & \langle R^* \rangle \phi &= \mu X.(\langle R \rangle X \vee \phi) \end{aligned}$$

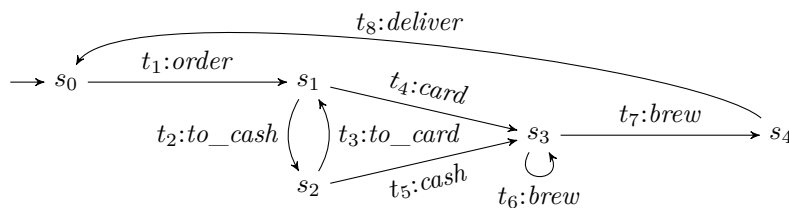
The box operator is defined dually. We say a path π *matches* a regular formula R if the sequence of actions on π is in the language of R .

3 Motivation

When analysing algorithms and systems, there are many different properties which may need to be checked. For instance, when model checking mutual exclusion algorithms we want to check linear properties such as mutual exclusion and starvation freedom, but also branching properties such as invariant reachability of the critical section. The modal μ -calculus, which subsumes even CTL^{*}, is able to express all these properties and more, and is therefore used in toolsets such as mCLR2 [10] and CADP [19].

An issue that is frequently encountered when checking liveness properties in particular, is that the model admits executions that violate the property but do not represent realistic executions of the real system. For example, models of algorithms that contain a busy waiting loop usually admit executions where processes do nothing except wait. Infinite loops can also be introduced by abstractions of reality, such as modelling a loop to represent an event that occurs an arbitrary, but finite, number of times. Counterexamples that are due to such modelling artefacts obscure whether the property is satisfied on all realistic executions. The problem we address in this paper is how to avoid such counterexamples and check properties only on realistic executions. We illustrate the problem with an example, which we also employ as a running example throughout this paper.

► **Example 4.** Consider the coffee machine modelled in Figure 1. When a user places an *order* for one or more cups of coffee, they are required to scan their payment *card*. If the user prefers using coinage, they switch the machine to its alternate mode (*to_cash*), and then pay in *cash*. In the alternate mode, the machine can be switched back using *to_card*. After



■ **Figure 1** The LTS for the running example.

payment, the machine will *brew* the cup(s) of coffee. This is modelled as a non-deterministic choice between a looping and a final *brew* action, since at least one cup was ordered. Finally, the coffee is *delivered* and the machine awaits the next order.

We consider three example properties.

1. *Single order*: whenever an *order* is made, there may not be a second *order* until a *deliver* has taken place, $[Act^* \cdot order \cdot \overline{deliver}^* \cdot order]ff$.
2. *Inevitable delivery*: whenever an *order* is made, there will inevitably be an occurrence of *deliver*, $[Act^* \cdot order] \mu X. ((Act)tt \wedge [\overline{deliver}]X)$.
3. *Possible delivery*: it is invariantly possible to eventually execute the *deliver* action, $[Act^*] \langle Act^* \cdot deliver \rangle tt$.

The described problem occurs with *inevitable delivery*: $s_0 t_1 s_1 t_4 (s_3 t_6)^\omega$ is a violating path, on which infinitely many cups are part of the same order. Similarly, $s_0 t_1 (s_1 t_2 s_2 t_3)^\omega$ violates the property because the user never decides on a payment method. The first counterexample represents an impossible scenario, and the second gives information on problematic user behaviour but tells us little about the machine itself.

The kind of spurious counterexamples discussed in the example above primarily occur when checking liveness properties. We therefore focus on liveness properties, such as *inevitable delivery*, in this paper. We will briefly discuss safety properties in Section 8.

There are ad-hoc solutions to exclude unrealistic counterexamples, e.g. altering the model to remove the unrealistic executions, or tailoring the formula to exclude specific problematic counterexamples [25]. Such ad-hoc solutions are undesirable because they clutter the model or the formula, and are therefore error-prone. We aim for a more generic solution, of which the correctness can be established once and for all. Such a generic solution requires, on the one hand, a general method to distinguish between realistic and unrealistic executions, and, on the other hand, a general class of liveness properties.

A general method to distinguish between realistic and unrealistic executions is provided by *completeness criteria* [21, 22], i.e., predicates on paths that label some as complete and all others as incomplete. If a property is satisfied on all complete paths, it is satisfied under the given completeness criterion. Completeness criteria give us a model-independent way to determine which paths are unrealistic, and therefore a generic solution to the stated problem. Depending on the property and the model, we may prefer a different completeness criterion. We therefore consider several criteria instead of fixing one specific criterion. These completeness criteria are discussed in Section 4.

To find a general class of liveness properties, we take the *property specification patterns* (PSP) of [16] as a starting point. Since the modal μ -calculus as presented in Section 2.2 supports references to action occurrences but not state information, we specifically interpret these patterns on action occurrences. Our first contribution, in Section 5, will be to characterise a class of liveness properties that subsumes all liveness properties expressible in

PSP. Our second and main contribution is then presented in Section 6, where we combine the identified completeness criteria with our class of liveness properties, yielding template formulae for each combination.

4 Completeness Criteria

It is often assumed, sometimes implicitly, that as long as a system is capable of executing actions, it will continue to do so [24]. One could consider this the “default” completeness criterion, also known as *progress* [21]; it says that only paths that are infinite or end in a deadlock state model complete runs and are hence complete paths. We first present a modified version of the progress assumption that allows some actions to be blocked by the environment. We then define the other completeness criteria considered in this paper. As already remarked in the previous section, the modal μ -calculus is most suited to reasoning about action occurrences. Hence, we focus on completeness criteria defined on action labels. For more general definitions on sets of transitions, see [24].

4.1 Progress with Blocking Actions

In [23], it is argued that it is useful to consider some actions of an LTS as blocking. A blocking action is an action that depends on participation by the environment of the modelled system. Consequently, even when such an action is enabled in a state because the system is willing to perform it, it may not be possible for the action to occur because the environment is uncooperative. In this paper, we refer to the set of blocking actions as $\mathcal{B} \subseteq Act$, and the set of non-blocking actions as $\overline{\mathcal{B}} = Act \setminus \mathcal{B}$. Which actions are in \mathcal{B} is a modelling choice.

The default progress assumption can be adapted to account for blocking actions [20, 24].

► **Definition 5.** *A state $s \in \mathcal{S}$ is a \mathcal{B} -locked state if, and only if, all actions enabled in s are in \mathcal{B} . A path π is \mathcal{B} -progressing if, and only if, it is infinite or ends in a \mathcal{B} -locked state.*

We refer to the assumption that only \mathcal{B} -progressing paths represent complete executions as \mathcal{B} -progress. The “default” completeness criterion is equivalent to \emptyset -progress.

► **Example 6.** Consider Figure 1. Here, *order* is an environment action, since it involves the user. If we do not assume that there will always be a next user, we should add *order* to \mathcal{B} . In some cases, we may want to consider the possibility that the machine is broken and not capable of producing coffee. In those cases, we should add *brew* to \mathcal{B} . Our choice of \mathcal{B} affects which paths are progressing: $s_0t_1s_1t_4s_3$ is not \emptyset -progressing, but it is $\{brew\}$ -progressing.

All completeness criteria we discuss in this paper are parameterised with a set of blocking actions. The justness and fairness assumptions discussed in the remainder of this section label paths as incomplete if certain actions do not occur. Since it can never be assumed that the environment supports the occurrence of blocking actions, we do not want justness and fairness to label paths as incomplete due to the non-occurrence of blocking actions.

For readability the prefix \mathcal{B} - will sometimes be dropped from the names of the completeness criteria and their acronyms. From this point, we will always discuss completeness criteria with respect to a set of blocking actions.

4.2 Justness

Justness [20, 24] is a natural extension of progress to exclude infinite paths instead of finite paths. The idea is that in addition to the system as a whole progressing, individual components in that system should also be able to make progress unless they are prevented

from doing so by other components. It is a weaker, and hence frequently more justifiable, assumption than the fairness assumptions we cover in the next section. In its original presentation, justness is defined with respect to sets of transitions. Which components contribute to a transition and how they contribute to them determines which transitions interfere with each other. We here consider justness defined with respect to actions instead, based on [6]. We do not go into how it is determined which actions interfere with each other here. For discussions on this topic and when the two definitions coincide, see [5, 6, 20].

Intuitively, justness of actions says that if an action a is enabled at some point of a path, then eventually some action that can interfere with the occurrence of a must occur in that path. That action may be a itself. In order to formalise the concept of interference, we require the concept of a *concurrency relation on actions*, \smile .

► **Definition 7.** *Relation $\smile \subseteq Act \times Act$ is a concurrency relation on actions if, and only if:*

1. \smile is irreflexive, and
2. for all $a \in Act$, if π is a path from a state s in which a is enabled to a state $s' \in \mathcal{S}$ such that $a \smile b$ for all actions b occurring in π , then a is enabled in s' .

We write $\not\smile$ for the complement of \smile . Note that \smile may be asymmetric.

Read $a \smile b$ as “ a is concurrent with b ”, and $a \not\smile b$ as “ b interferes with a ” or “ b eliminates a ”. A labelled transition system can be extended with a concurrency relation on actions, which produces a *labelled transition system with concurrency* (LTSC).

We here present the definition for justness of actions with blocking actions.

► **Definition 8.** *A path π satisfies \mathcal{B} -justness of actions (\mathcal{B} -JA) if, and only if, for each action $a \in \overline{\mathcal{B}}$ that is enabled in some state s in π , an action $a' \in Act$ occurs in the suffix π' of π starting in s such that $a \not\smile a'$.*

► **Example 9.** Consider Figure 1, specifically the path $s_0 t_1 (s_1 t_2 s_2 t_3)^\omega$. On this path the user keeps switching the mode of the machine, without paying. To see if this path satisfies \emptyset -JA, we need a concrete \smile . Consider a \smile such that $card \not\smile to_cash$, $cash \not\smile to_card$, and $a \not\smile a$ for all action labels a . These are all required for \smile to be a valid concurrency relation. This is because by Definition 7, \smile must be irreflexive, and when an action is enabled it must remain enabled on any path on which no interfering action occurs. Since $card$ is enabled in s_1 but not s_2 , it must be the case that $card \not\smile to_cash$. Similarly, we must have $cash \not\smile to_card$. With such a concurrency relation, the path satisfies \emptyset -JA since every action that is enabled is subsequently eliminated. In this LTS, there is no valid choice of \smile that makes this path violate \emptyset -JA. However, if we modify Figure 1 by replacing both $card$ and $cash$ with the action pay , then Definition 7 does not enforce that to_cash and to_card interfere with the actions on t_4 and t_5 , since pay is enabled in both s_1 and s_2 . We can choose whether $pay \smile to_cash$ and $pay \smile to_card$. If pay is concurrent with both, then the path $s_0 t_1 (s_1 t_2 s_2 t_3)^\omega$ violates \emptyset -JA. If either interferes with pay , then the path satisfies \emptyset -JA.

4.3 Fairness

There are situations where we want to exclude a larger set of infinite paths than those excluded by justness, or where we do not have a concurrency relation. For this, we can use what are called *fairness assumptions* in the literature. These are a class of predicates on paths that distinguish between *fair* and *unfair* infinite paths. It is assumed that only the fair paths are complete. For an overview of many common fairness assumptions, see [24]. In this paper, we consider weak fairness of actions, strong fairness of actions, and (weak and strong) hyperfairness of actions. Each of the assumptions we discuss has the general shape, adapted

from [2], “if it is sufficiently often possible for an action to occur, it will occur sufficiently often”. What it means for an action to be “sufficiently often possible” and “occur sufficiently often” depends on the exact assumption.

We first discuss *weak fairness of actions*, which says that actions that are always enabled must eventually occur. It is one of the most commonly discussed fairness assumptions. We define weak fairness of actions formally, with respect to a set of blocking actions \mathcal{B} .

► **Definition 10.** *A path π satisfies \mathcal{B} -weak fairness of actions (\mathcal{B} -WFA) if, and only if, for every suffix π' of π , every action $a \in \overline{\mathcal{B}}$ that is perpetually enabled in π' occurs in π' .*

► **Example 11.** Consider again Figure 1, with *card* and *cash* both replaced by *pay*. Then the path $s_0t_1(s_1t_2s_2t_3)^\omega$ violates \emptyset -WFA, since *pay* is perpetually enabled in a suffix of this path without occurring. If there are two separate actions for paying with cash or card, the path satisfies \emptyset -WFA because no actions are perpetually enabled in any suffix.

Next, *strong fairness of actions* says that on a path, all actions that are enabled infinitely often, must occur infinitely often. Formally, we define strong fairness of actions as:

► **Definition 12.** *A path π satisfies \mathcal{B} -strong fairness of actions (\mathcal{B} -SFA) if, and only if, for every suffix π' of π , every action $a \in \overline{\mathcal{B}}$ that is relentlessly enabled in π' occurs in π' .*

Strong fairness is a stronger assumption than weak fairness, since it classifies more paths as incomplete. This follows from perpetual enabledness implying relentless enabledness.

► **Example 13.** The path $s_0t_1(s_1t_2s_2t_3)^\omega$ in Figure 1 satisfies \emptyset -WFA since there are no perpetually enabled actions in any suffix of the path. However, *cash* is relentlessly enabled in suffixes of this path, and yet does not occur. Hence, this path violates \emptyset -SFA.

Finally, we discuss *hyperfairness of actions*. Informally, it says that on all fair paths, every action that can always become enabled must occur infinitely often. The idea is that if there is always a reachable future where the action occurs, then it is merely unlucky if the action does not occur infinitely often. The concept of hyperfairness is introduced and named in [3]. For our presentation of hyperfairness, we use the generalisation from [30]. We first formalise what it means that an action “can become” enabled, by defining *reachability*.

► **Definition 14.** *We say that:*

- *A state $s \in \mathcal{S}$ is \mathcal{B} -reachable from some state $s' \in \mathcal{S}$ if, and only if, there exists a \mathcal{B} -free path starting in s' that ends in s .*
- *An action $a \in \text{Act}$ is \mathcal{B} -reachable from some state $s \in \mathcal{S}$ if, and only if, there exists a state $s' \in \mathcal{S}$ that is \mathcal{B} -reachable from s and in which a is enabled.*
- *A state $s \in \mathcal{S}$ or action $a \in \text{Act}$ is perpetually \mathcal{B} -reachable on a path π if, and only if, it is \mathcal{B} -reachable from every state of π .*
- *A state $s \in \mathcal{S}$ or action $a \in \text{Act}$ is relentlessly \mathcal{B} -reachable on a path π if, and only if, every suffix of π contains a state from which it is \mathcal{B} -reachable.*

From the intuitive description of hyperfairness, it is clear it is a variant of weak or strong fairness with reachability instead of enabledness, giving us two possible definitions of hyperfairness. We name the two interpretations weak hyperfairness and strong hyperfairness respectively. Both interpretations of hyperfairness are reasonable, and in fact when not considering blocking actions, they coincide [30]. However, this is not the case when blocking actions are included in the definitions. We therefore consider both variants.

► **Definition 15.** A path π satisfies weak \mathcal{B} -hyperfairness of actions (\mathcal{B} -WHFA) if, and only if, for every suffix π' of π , every action $a \in \overline{\mathcal{B}}$ that is perpetually \mathcal{B} -reachable in π' occurs in π' .

► **Definition 16.** A path π satisfies strong \mathcal{B} -hyperfairness of actions (\mathcal{B} -SHFA) if, and only if, for every suffix π' of π , every action $a \in \overline{\mathcal{B}}$ that is relentlessly \mathcal{B} -reachable in π' occurs in π' .

Since enabledness implies reachability, WHFA is stronger than WFA, and SHFA is stronger than SFA. Perpetually reachability implies relentless reachability, so SHFA is also stronger than WHFA. However, as the next examples will show, SFA and WHFA are incomparable.

► **Example 17.** The impact of hyperfairness can clearly be seen when non-determinism is used. Consider the path $s_0t_1s_1t_4(s_3t_6)^\omega$ in Figure 1. This path satisfies \emptyset -SFA, since the only action that is relentlessly enabled on this path, *brew*, also occurs infinitely often. However, as long as *deliver* $\notin \mathcal{B}$ and *brew* $\notin \mathcal{B}$, this path does not satisfy \mathcal{B} -WHFA or \mathcal{B} -SHFA: *deliver* is \mathcal{B} -reachable from s_3 , and therefore is perpetually and relentlessly \mathcal{B} -reachable in a suffix of this path, but does not occur. We here see \mathcal{B} -SFA does not imply \mathcal{B} -WHFA.

► **Example 18.** In Figure 1, consider $s_0t_1(s_1t_2s_2t_3)^\omega$ with $\mathcal{B} = \{\textit{order}, \textit{to_cash}, \textit{to_card}\}$. This path satisfies \mathcal{B} -WHFA because *card* and *cash* are only \mathcal{B} -reachable from s_1 and s_2 respectively. They are not perpetually \mathcal{B} -reachable in any suffix of this path, therefore \mathcal{B} -WHFA is satisfied. However, they are relentlessly \mathcal{B} -reachable, so \mathcal{B} -SHFA is violated. This demonstrates that \mathcal{B} -WHFA and \mathcal{B} -SFHA do not coincide when blocking actions are considered. The actions *card* and *cash* are also relentlessly \mathcal{B} -enabled, so \mathcal{B} -SFA is also violated. Hence, \mathcal{B} -WHFA does not imply \mathcal{B} -SFA.

5 A Generalisation of the Property Specification Liveness Patterns

Dwyer, Avrunin and Corbett observed that a significant majority of properties that are used in practice can be fit into a set of property specification patterns [16]. These patterns consist of a *behaviour* that must be satisfied and a *scope* within a path that delimits where the behaviour must be satisfied. We focus on expressing properties that are captured by PSP.

Of all behaviours considered in [16], only *existence*, *existence at least*, *response* and *chain response* represent pure liveness properties. The *global* and *after* scopes, when combined with any of these four behaviours, give liveness properties.¹ All other scopes result in safety properties or properties that combine safety and liveness. Of those, we cover the *until* and *after-until* scopes, since we can incorporate those into our formulae with little difficulty.

For behaviours, *existence at least* says some action in a set S_r must occur at least k times in the scope; when $k = 1$ we call this *existence*. The *response* behaviour requires that whenever an action in a set S_q occurs, it must be followed by the occurrence of an action in S_r . When chains of action occurrences are used instead of individual action occurrences, this is called *chain response*. For the scopes, *global* refers to the full path and *after* to the path after the first occurrence of an action in a set S_a . The *until* scope refers to the path before the first occurrence of an action in a set S_b , or the full path if no such action occurs. Finally, *after-until* combines *after* and *until*, referring to every subpath of the path that starts after any occurrence of an action in S_a and ends before the following occurrence of an action in S_b . If no action in S_b occurs, the behaviour must still be satisfied after S_a .

¹ In the full version, we recap PSP and argue why only these patterns represent pure liveness properties.

► **Example 19.** Consider again the properties we presented in Example 4. *Single order* is *absence after-until*, with $S_a = \{\text{order}\}$, $S_b = \{\text{deliver}\}$ and $S_r = \{\text{order}\}$. *Inevitable delivery* is *global response* with $S_q = \{\text{order}\}$ and $S_r = \{\text{deliver}\}$. *Possible delivery* does not fit into the patterns on occurrences of actions, since it contains a requirement on states, specifically that the state admits a path on which *delivery* occurs.

We want to create formulae for all 16 combinations of the selected behaviours and scopes. To make our results more compact and generic, we first generalise these 16 patterns into a single template property. This template works by describing the shape of a violating path for a property that fits one of these patterns. Intuitively, this shape is: “after the occurrence of ρ , there are no occurrences of α_f up until the (optional) occurrence of α_e ”. For our template formulae to be syntactically correct, it is important that ρ is a regular formula, describing the prefix that a violating path must have, whereas α_f and α_e are sets of actions. The actions in α_f are those that are forbidden from occurring after ρ on a violating path, whereas the actions in α_e indicate the end of the scope in which α_f may not occur.

We formalise this template as follows:

► **Definition 20.** A path π is $(\rho, \alpha_f, \alpha_e)$ -violating if, and only if, there exist π_{pre} and π_{suf} such that:

1. $\pi = \pi_{pre} \cdot \pi_{suf}$, and
2. π_{pre} matches ρ , and
3. π_{suf} satisfies at least one of the following conditions:
 - a. π_{suf} is α_f -free, or
 - b. π_{suf} contains an occurrence of an action in α_e , and the prefix of π_{suf} before the first occurrence of an action in α_e is α_f -free.

For readability, we frequently refer to $(\rho, \alpha_f, \alpha_e)$ -violating paths as violating paths. We sometimes summarise condition 3 as “ π_{suf} is α_f -free up until the first occurrence of α_e ”. See Figure 2 for an illustration of what types of paths are considered violating.



■ **Figure 2** The four types of $(\rho, \alpha_f, \alpha_e)$ -violating paths: finite or infinite, and without or with α_e . Always, it has a prefix matching ρ and is α_f -free up until the first occurrence of an action in α_e .

All 16 patterns can indeed be represented by the non-existence of $(\rho, \alpha_f, \alpha_e)$ -violating paths, albeit some more directly than others. It turns out that ρ , α_f and α_e can mostly be determined separately for behaviour and scope. For these patterns, α_f is only affected by behaviour and α_e only by scope. However, we must split up the regular formula ρ into a behaviour component, ρ_b , and scope component, ρ_s , such that $\rho = \rho_s \cdot \rho_b$. See Table 1a and Table 1b for how the variables should be instantiated for the four scopes and three of the four behaviours. For a compact representation, we use \sum to generalise the union operator on regular formulae (+). We also use x^i to represent i concatenations of x , where $x^0 = \varepsilon$.

We do not include *chain response* in Table 1b, since it does not fit into a single formula. However, it is possible to represent *chain response* as several *response* formulae placed in conjunction with each other.²

² We give an example of this in the full version of this paper.

■ **Table 1** Variable instantiation for templates.

(a) For scopes.			(b) For behaviours.		
Scope	ρ_s	α_e	Behaviour	ρ_b	α_f
Global	ε	\emptyset	Existence	ε	S_r
Until	ε	S_b	Existence at least k	$\sum_{0 \leq i < k} (\overline{\alpha_e \cup S_r^*} \cdot S_r)^i$	S_r
After	$\overline{S_a^*} \cdot S_a$	\emptyset	Response	$\overline{\alpha_e^*} \cdot S_q$	S_r
After-until	$Act^* \cdot S_a$	S_b	Chain response	-	-

6 Template Formulae

In this section, we present the modal μ -calculus formulae representing the non-existence of a violating path, as defined in Section 5, that satisfies one of the completeness criteria from Section 4. We express the non-existence of such a path, rather than expressing the equivalent notion that all complete paths satisfy the property, because we find the resulting formulae to be more intuitive. We first present a formula for \mathcal{B} -progress only. Subsequently, we give the formulae for weak fairness, weak hyperfairness and justness using a common structure all three share. Finally, we present the formulae for strong fairness and strong hyperfairness. In the justness and fairness formulae, \mathcal{B} -progress is also included: these assumptions eliminate unrealistic infinite paths, but we still need progress to discard unrealistic finite paths.

Proofs of the theorems in this section are included in the full version of this paper. A sketch of the proof of Theorem 24 is included in Appendix A to illustrate our approach.

6.1 Progress

A formula for the non-existence of a violating path without progress is uninteresting. If progress is not assumed then all finite paths are complete, and therefore a path consisting of just ρ is a violating path whenever $\alpha_f \neq \emptyset$. The non-existence of a violating path would then be captured by $\neg\langle\rho\rangle tt$. This is why we include progress in all our formulae.

To represent progress, we must capture that as long as non-blocking actions are enabled, some transitions must still be executed. The following formula captures the non-existence of violating paths under \mathcal{B} -progress:

$$\neg\langle\rho\rangle\nu X.((\alpha_e)tt \vee [\overline{\mathcal{B}}]ff \vee \langle\overline{\alpha_f}\rangle X) \quad (1)$$

Intuitively, this formula says that there is no path that starts with a prefix matching ρ , after which infinitely often a transition can be taken that is not labelled with an action in α_f , or such transitions can be taken finitely often before a state is reached that is \mathcal{B} -locked or in which α_e is enabled. In the former case there is a \mathcal{B} -progressing path on which no actions in α_f occur after ρ . If a state in which α_e is enabled is reached, then it is guaranteed a violating and \mathcal{B} -progressing path exists: by arbitrarily extending the path as long as non-blocking actions are still enabled, a \mathcal{B} -progressing and violating path can be constructed.

► **Theorem 21.** *A state in an LTS satisfies Formula 1 if, and only if, it does not admit \mathcal{B} -progressing paths that are $(\rho, \alpha_f, \alpha_e)$ -violating.*

Since representing a liveness pattern without progress leads to uninteresting formulae, it is unsurprising that previous translations of PSP to the μ -calculus have also implicitly included progress. For instance, the translations from [31] for the liveness patterns of PSP are very similar to Formula 1, albeit in positive form and without blocking actions.

6.2 Weak Fairness, Weak Hyperfairness and Justness

For weak fairness, weak hyperfairness and justness, we employ a trick inspired by the formula for justness presented in [6] (which was in turn inspired by [11]): we translate a requirement on a full path into an invariant that can be evaluated within finitely many steps from every state of the path. We illustrate this using weak fairness.

On every suffix of a weakly fair path, every perpetually enabled non-blocking action occurs. To turn this into an invariant, we observe that we can evaluate a property on all suffixes of a path by evaluating it from every state of the path instead. Next we must determine, within finitely many steps, if an action is perpetually enabled on a possibly infinite path. We do this by observing that if an action is not perpetually enabled, it must become disabled within finitely many steps. An equivalent definition of WFA therefore is: a path π satisfies WFA if, and only if, for every state s in π , every action $a \in \bar{\mathcal{B}}$ that is enabled in s occurs or becomes disabled within finitely many steps on the suffix of π starting in s . This translation of WFA determines three things for every non-blocking action a . First, which actions may need to occur because of a ; in the case of WFA this is a itself. Second, when those actions need to occur; for WFA this is when a is enabled. We refer to this as the action being “on”. Finally, when those actions do not need to occur; for WFA this is when a becomes disabled. We refer to this as the action being “off”. When an action that was previously on becomes off, or one of the required actions occurs, we say the action is “eliminated”. By choosing different definitions for an action being on or off, and when an action is eliminated, we can also represent justness and weak hyperfairness in the same way.

We find that completeness criteria for which such a translation can be made can be represented using the same generalised formula. We will present this formula and how to instantiate it for WFA, WHFA and JA. However, we must first formalise what it means for a predicate on paths to be translatable to an invariant that can be evaluated within finitely many steps. We introduce the term *finitely realisable (path) predicates* for this purpose.

► **Definition 22.** *A path predicate P is finitely realisable if, and only if, there exist mappings ϕ_{on} and ϕ_{of} from non-blocking actions to closed modal μ -calculus formulae, and a mapping α_{el} from non-blocking actions to sets of actions, such that:*

1. *A path π satisfies predicate P if, and only if, all states s on π satisfy the following: for all $a \in \bar{\mathcal{B}}$, if s satisfies $\phi_{on}(a)$ then the suffix π' of π starting in s must contain an occurrence of some action in $\alpha_{el}(a)$ or a state that satisfies $\phi_{of}(a)$.*
2. *A state s is a \mathcal{B} -locked state if, and only if, $s \notin \llbracket \phi_{on}(a) \rrbracket$ for all $a \in \bar{\mathcal{B}}$.*
3. *For every state s and for all $a \in \bar{\mathcal{B}}$, $s \in \llbracket \phi_{on}(a) \rrbracket$ implies $s \notin \llbracket \phi_{of}(a) \rrbracket$.*
4. *For all states s and all $a \in \bar{\mathcal{B}}$ such that $s \in \llbracket \phi_{on}(a) \rrbracket$, if there exists a finite path π from s to a state s' such that there is no occurrence of an action in $\alpha_{el}(a)$ on π and there is no state on π that satisfies $\phi_{of}(a)$, then $s' \in \llbracket \phi_{on}(a) \rrbracket$.*

We refer to these four properties as the invariant property, the locking property, the exclusive property and the persistent property, respectively.

The general formula for finitely realisable predicates is as follows:

$$\neg \langle \rho \rangle \nu X. \left(\bigwedge_{a \in \bar{\mathcal{B}}} (\phi_{on}(a) \Rightarrow \langle \bar{\alpha}_f^* \rangle (\langle \alpha_e \rangle tt \vee (\phi_{of}(a) \wedge X) \vee \langle \alpha_{el}(a) \setminus \alpha_f \rangle X)) \right) \quad (2)$$

This formula has similarities to Formula 1, particularly how ρ and α_e are integrated. The important part is that after ρ , it must invariantly hold that all non-blocking actions for which $\phi_{on}(a)$ is satisfied are later eliminated. An action a is eliminated if, within finitely many steps, $\phi_{of}(a)$ is satisfied or an action in $\alpha_{el}(a)$ occurs. In both cases, the invariant must

once again hold. After ρ , no actions in α_f may occur. The formula works correctly for finite paths as well as infinite ones: if it is possible to reach a \mathcal{B} -locked state after ρ without taking actions in α_f , then X is satisfied due to the locking property, and a violating path is found.

Formula 2 is a template formula in two ways: ρ , α_f and α_e determine what property is captured, and ϕ_{on} , ϕ_{of} and α_{el} determine the completeness criterion. In this paper, we only cover how to instantiate the formula for WFA, WHFA and JA, but it can also be used for other finitely realisable predicates. However, the correctness proof of the formula depends on the criterion being *feasible*. Feasibility on paths [2] is defined as follows.

► **Definition 23.** *A predicate on paths P is feasible if, and only if, for every LTS M , every finite path π in M can be extended to a path π' that satisfies P and is still a valid path in M .*

That WFA, WHFA and JA are indeed feasible for finite LTSs is proven in the full version.

► **Theorem 24.** *For all feasible and finitely realisable path predicates P , it holds that an LTSC satisfies Formula 2 if, and only if, its initial state does not admit \mathcal{B} -progressing paths that satisfy P and are $(\rho, \alpha_f, \alpha_e)$ -violating.*

By instantiating the theorem for each completeness criterion, we derive the following:

► **Corollary 25.** *A state in an LTS satisfies Formula 2 with $\phi_{on}(a) = \langle a \rangle tt$, $\phi_{of}(a) = [a]ff$ and $\alpha_{el}(a) = \{a\}$ for all $a \in \overline{\mathcal{B}}$ if, and only if, it does not admit \mathcal{B} -progressing paths that satisfy \mathcal{B} -weak fairness of actions and are $(\rho, \alpha_f, \alpha_e)$ -violating.*

► **Corollary 26.** *A state in an LTS satisfies Formula 2 with $\phi_{on}(a) = \langle \overline{\mathcal{B}}^* \cdot a \rangle tt$, $\phi_{of}(a) = [\overline{\mathcal{B}}^* \cdot a]ff$ and $\alpha_{el}(a) = \{a\}$ for all $a \in \overline{\mathcal{B}}$ if, and only if, it does not admit \mathcal{B} -progressing paths that satisfy weak \mathcal{B} -hyperfairness of actions and are $(\rho, \alpha_f, \alpha_e)$ -violating.*

► **Corollary 27.** *A state in an LTSC satisfies Formula 2 with $\phi_{on}(a) = \langle a \rangle tt$, $\phi_{of}(a) = ff$ and $\alpha_{el}(a) = \{b \in Act \mid a \not\prec b\}$ for all $a \in \overline{\mathcal{B}}$ if, and only if, it does not admit \mathcal{B} -progressing paths that satisfy \mathcal{B} -justness of actions and are $(\rho, \alpha_f, \alpha_e)$ -violating.*

6.3 Strong Fairness and Strong Hyperfairness

SFA is not finitely realisable because we cannot observe within finitely many steps whether an action is relentlessly enabled: even if we observe several times that it is disabled, it may still be infinitely often enabled along the whole path. Hence, we cannot use Formula 2.

Instead we observe that, on a path, actions that are not relentlessly enabled must eventually become perpetually disabled. If the path is strongly fair, then all relentlessly enabled non-blocking actions occur infinitely often. We can therefore say that a path is strongly fair if we can divide all non-blocking actions into two disjoint sets: those that occur infinitely often and those that eventually become perpetually disabled. This observation is also made in [36], where a μ -calculus formula for termination under strong fairness is given.

Using this idea, we give the following template formula for SFA:

$$\neg \langle \rho \cdot \overline{\alpha_f}^* \rangle (\langle \alpha_e \rangle tt \vee [\overline{\mathcal{B}}]ff \vee \bigvee_{\emptyset \neq F \subseteq \overline{\mathcal{B}}} \nu X. (\bigwedge_{a \in F} \mu W. ((\bigwedge_{b \in \overline{\mathcal{B}} \setminus F} [b]ff) \wedge (\langle a \setminus \alpha_f \rangle X \vee \langle \overline{\alpha_f} \rangle W))) \quad (3)$$

The use of negation, the exclusion of α_f , and ρ in the diamond operator at the start of this formula are the same as in Formula 1. We explain the start of the formula after addressing the part starting with $\bigvee_{\emptyset \neq F \subseteq \overline{\mathcal{B}}}$. Here, we use that on a strongly fair path, all non-blocking

actions can be divided into those that occur infinitely often and those that become perpetually disabled. The disjunction over subsets considers all possible ways of selecting some non-empty subset F of $\overline{\mathcal{B}}$ that should occur infinitely often. The greatest fixpoint states that infinitely often, all those actions must indeed occur within finitely many steps. Additionally, at no point may a non-blocking action not in F be enabled. We exclude $F = \emptyset$ because the logic of the greatest fixed point formula we give relies on there being at least one a in F . The special case that F is empty and therefore a \mathcal{B} -locked state should be reached, is instead covered by explicitly considering $[\overline{\mathcal{B}}]ff$ earlier in the formula. Returning to the start of the formula, we allow a finite α_f -free path before the greatest fixpoint is satisfied. The reason is that it may take several steps before all the non-blocking actions that are only finitely often enabled become perpetually disabled. Since we include a finite prefix already, we also add the cases that an action in α_e becomes enabled or that a \mathcal{B} -locked state is reached here, rather than deeper into the formula like in Formula 2.

► **Theorem 28.** *An LTS satisfies Formula 3 if, and only if, its initial state does not admit \mathcal{B} -progressing paths that satisfy \mathcal{B} -strong fairness of actions and are $(\rho, \alpha_f, \alpha_e)$ -violating.*

Due to the quantification over subsets, the formula is exponential in the number of actions in $\overline{\mathcal{B}}$. Beyond small models, it is therefore not practical. However, it can serve as a basis for future work. For instance, if fairness is applied to sets of actions rather than individual actions, the formula is exponential in the number of sets instead, which may be smaller depending on how the sets are formed [35].

We can adapt the formula for strong fairness to a formula for strong hyperfairness, by replacing perpetual disabledness of non-blocking actions not in F with perpetual unreachability.

$$\neg \langle \rho \cdot \overline{\alpha_f}^* \rangle (\langle \alpha_e \rangle tt \vee [\overline{\mathcal{B}}]ff \vee \bigvee_{\emptyset \neq F \subseteq \overline{\mathcal{B}}} \nu X. (\bigwedge_{a \in F} \mu W. ((\bigwedge_{b \in \overline{\mathcal{B}} \setminus F} [\overline{\mathcal{B}}^* \cdot b]ff) \wedge (\langle a \setminus \alpha_f \rangle X \vee \langle \overline{\alpha_f} \rangle W))) \rangle) \quad (4)$$

► **Theorem 29.** *An LTS satisfies Formula 4 if, and only if, its initial state does not admit a \mathcal{B} -progressing path that satisfies strong \mathcal{B} -hyperfairness of actions and is $(\rho, \alpha_f, \alpha_e)$ -violating.*

Since we are not aware of other completeness criteria that fit the same structure, we do not provide a generalised formula here like we did with Formula 2.

7 Application Example

We here give an example of an application of the template formulae. In [25], several mutual exclusion algorithms are analysed using the mCRL2 toolset. Their analysis of Dekker's algorithm [14] presents the following modal μ -calculus formula for starvation freedom of processes with id's 0 and 1. For clarity, the notation has been adjusted to match the previous sections and action names have been simplified.

$$[Act^*] \bigwedge_{i \in \{0,1\}} [\{wish_flag(i, b) \mid b \in \mathbb{B}\}] \mu X. (\overline{\langle enter(i) \rangle} X \wedge \langle Act \rangle tt) \quad (5)$$

Starvation freedom is a *global response* property. In this case, the starvation freedom of a process i is represented as an instantiation of the pattern with $S_q = \{wish_flag(i, b) \mid b \in \mathbb{B}\}$ and $S_r = \{enter(i)\}$. Indeed, the above formula is equivalent to:

$$\bigwedge_{i \in \{0,1\}} \neg \langle Act^* \cdot \{wish_flag(i, b) \mid b \in \mathbb{B}\} \rangle \nu X. (\langle \emptyset \rangle tt \vee [Act]ff \vee \overline{\langle enter(i) \rangle} X) \quad (6)$$

Observe that, when taking $\mathcal{B} = \emptyset$, the above matches a conjunction of two instances of Formula 1, taking ρ , α_e and α_f as suggested in Table 1 for *global response*. Thus, this formula captures starvation freedom under \emptyset -progress. In [25], it is reported that mCRL2 finds a violating path for this formula; a path which the authors note is unfair. The exact fairness assumption considered is not made concrete. As an ad-hoc solution, the modal μ -calculus formula is adjusted to specifically ignore that counterexample. Subsequently, mCRL2 finds another counterexample, which the authors again claim is unfair. Instead of creating yet another formula, they move on to Peterson’s algorithm, which is deemed easier to analyse. Using our template formulae, we can easily produce a formula for starvation freedom under several different completeness criteria. We give the formula for \emptyset -WFA, as an example.

$$\bigwedge_{i \in \{0,1\}} \neg \langle Act^* \cdot \{wish_flag(i, b) \mid b \in \mathbb{B}\} \rangle$$

$$\nu X. (\bigwedge_{a \in Act} (\langle a \rangle tt \Rightarrow \overline{\langle enter(i) \rangle} (\langle \emptyset \rangle tt \vee ([a]ff \wedge X) \vee \langle a \setminus enter(i) \rangle X))) \quad (7)$$

We check this formula on the model from [25] using mCRL2. Since mCRL2 only supports quantification over data parameters and not over actions, the conjunction over Act must be written out explicitly. The tool reports that the formula is violated. Examining the counterexample reveals this is because actions in the model do not show which process performs the action. Therefore, process i reading value v from a register r is labelled with the same action as process j reading v from r . We add the responsible process to each action label, and also define $\mathcal{B} = \{wish_flag(i, i, b) \mid i \in \{0, 1\}, b \in \mathbb{B}\}$, to capture that processes are allowed to remain in their non-critical section indefinitely. This was not considered in Formula 5, but it is part of the mutual exclusion problem [15, 22]. The tool reports that the modified formula is satisfied. We can therefore conclude that Dekker’s algorithm satisfies starvation freedom when assuming weak fairness of actions, as long as it is taken into account for each action which process is responsible for it.

Our other formulae can be used in similar ways. An example of how to use the justness formula in mCRL2, including a method for encoding the concurrency relation, is given in [6].

8 Discussion

In this section, we briefly reflect on the coverage of the properties we consider, and our choice in focusing on the modal μ -calculus.

Firstly, we have exclusively addressed liveness properties in this paper thus far. As indicated previously, the problem we are considering primarily crops up for these properties. This is because, as pointed out in [22], when a completeness criterion is feasible, assuming the criterion holds true or not has no impact on whether a safety property is satisfied or not. The reason is that for safety properties on paths, any path that violates the property must contain a finite prefix such that any extension of that prefix also violates the property [1]. Therefore, if a completeness criterion is feasible, then whenever a model contains incomplete paths that violate a safety property it also contains complete paths that violate the property. All completeness criteria discussed in Section 4 are feasible with respect to finite LTSs, and hence we do not need to consider patterns that capture safety properties. For modal μ -calculus formulae for the safety properties of PSP, without integrated completeness criteria, we refer to [31] and [34]. For properties that are a combination of safety and liveness, the components can be turned into separate formulae and checked separately.

Readers may also wonder about alternative methods of representing properties under completeness criteria, such as using LTL. As indicated in Section 3, there are many contexts where we also want to consider non-linear properties, and hence the modal μ -calculus is preferred. Automatic translations from LTL to the modal μ -calculus exist, but can be exponential in complexity [13] and it is unclear at this time if this blow-up is avoided in this case. Anecdotal evidence [33] suggests this is not the case for existing translations. In [22] several completeness criteria are represented in LTL, but it is noted that this translation requires introducing new atomic propositions which hides the complexity of this translation. The representation of hyperfairness in particular may be expensive, since atomic propositions for all reachable actions are required. It is also unclear how to combine LTL-based translations effectively with symbolic model checking approaches. For these reasons, a direct representation in the modal μ -calculus is preferable.

9 Conclusion

In this paper, we have presented formulae for liveness properties under several completeness criteria. As part of this, we defined a property template that generalises the liveness properties of PSP, which has been estimated to cover a majority of properties found in the literature [16]. The completeness criteria covered are progress, justness, weak fairness, strong fairness, and hyperfairness, all defined with respect to actions and parameterised with a set of blocking actions. The formulae have all been manually proven to be correct.

For future work, one goal is to formalise our manual proofs using a proof assistant. Another avenue for future work is extending our formulae to cover a wider range of completeness criteria and properties. We suggest some potential extensions here.

One of our contributions is the identification of a shared common structure underlying justness, weak fairness and weak hyperfairness: they are finitely realisable path predicates. Our formula for such predicates can be adapted to arbitrary feasible finitely realisable path predicates. While we do not have such a generic formula for other completeness criteria, our characterisation of $(\rho, \alpha_f, \alpha_e)$ -violating paths can be used as a basis to express the non-existence of complete paths violating many common properties for different notions of completeness as well, as we demonstrate with strong fairness and strong hyperfairness. We are especially interested in extending our formulae to allow fairness over sets of actions, rather than individual actions, similar to the task-based definitions from [24].

In terms of properties, we can look at proposed extensions of PSP, such as those suggested in [12]. There is also the *constrained chain* behaviour, which is a modification of precedence chain and response chain given in [16]. There are extensions of PSP to real-time [4, 28] and probabilistic [27] contexts as well. Finally, in [5] the formula from [6] that formed the basis of Formula 2 is extended to also include state information.

There are therefore many potentially useful extensions of the formulae presented in this paper. However, the presented template formulae already cover many completeness criteria and liveness properties, making them useful for model checking in practice.

References

- 1 Mack W. Alford, Leslie Lamport, and Geoff P. Mullery. Basic concepts. In Mack W. Alford, Jean-Pierre Ansart, Günter Hommel, Leslie Lamport, Barbara Liskov, Geoff P. Mullery, and Fred B. Schneider, editors, *Distributed Systems: Methods and Tools for Specification, An Advanced Course, April 3-12, 1984 and April 16-25, 1985, Munich, Germany*, volume 190 of *Lecture Notes in Computer Science*, pages 7–43. Springer, 1984. doi:10.1007/3-540-15216-4_12.
- 2 Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. *Distributed Comput.*, 2(4):226–241, 1988. doi:10.1007/BF01872848.

- 3 Paul C. Attie, Nissim Francez, and Orna Grumberg. Fairness and hyperfairness in multi-party interactions. In Frances E. Allen, editor, *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 292–305. ACM Press, 1990. doi:10.1145/96709.96739.
- 4 Pierfrancesco Bellini, Paolo Nesi, and Davide Rogai. Expressing and organizing real-time specification patterns via temporal logics. *J. Syst. Softw.*, 82(2):183–196, 2009. doi:10.1016/j.jss.2008.06.041.
- 5 Mark S. Bouwman. *Supporting Railway Standardisation with Formal Verification*. Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science, Eindhoven University of Technology, 2023. URL: https://pure.tue.nl/ws/portalfiles/porta1/307965423/20231023_Bouwman_hf.pdf.
- 6 Mark S. Bouwman, Bas Luttik, and Tim A. C. Willemse. Off-the-shelf automated analysis of liveness properties for just paths. *Acta Informatica*, 57(3-5):551–590, 2020. doi:10.1007/s00236-020-00371-w.
- 7 Julian C. Bradfield and Colin Stirling. Modal logics and mu-calculi: an introduction. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier Science, 2001. doi:10.1016/b978-044482830-9/50022-9.
- 8 Julian C. Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, Johan Van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 721–756. Elsevier, 2007. doi:10.1016/s1570-2464(07)80015-2.
- 9 Julian C. Bradfield and Igor Walukiewicz. The mu-calculus and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 871–919. Springer, 2018. doi:10.1007/978-3-319-10575-8_26.
- 10 Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mCRL2 toolset for analysing concurrent systems. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2019. doi:10.1007/978-3-030-17465-1_2.
- 11 Edmund M. Clarke, Orna Grumberg, Kenneth L. McMillan, and Xudong Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In Bryan Preas, editor, *Proceedings of the 32nd Conference on Design Automation, San Francisco, California, USA, Moscone Center, June 12-16, 1995*, pages 427–432. ACM Press, 1995. doi:10.1145/217474.217565.
- 12 Rachel L. Cobleigh, George S. Avrunin, and Lori A. Clarke. User guidance for creating precise and accessible property specifications. In Michal Young and Premkumar T. Devanbu, editors, *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, pages 208–218. ACM, 2006. doi:10.1145/1181775.1181801.
- 13 Sjoerd Cranen, Jan Friso Groote, and Michel A. Reniers. A linear translation from CTL* to the first-order modal μ -calculus. *Theor. Comput. Sci.*, 412(28):3129–3139, 2011. doi:10.1016/j.tcs.2011.02.034.
- 14 Edsger W Dijkstra. Over de sequentialiteit van procesbeschrijvingen (EWD-35). EW dijkstra archive. *Center for American History, University of Texas at Austin*, 1962. URL: <https://www.cs.utexas.edu/~EWD/ewd00xx/EWD35.PDF>.
- 15 Edsger W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569, 1965. doi:10.1145/365559.365617.
- 16 Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In Barry W. Boehm, David Garlan, and Jeff Kramer, editors, *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999*, pages 411–420. ACM, 1999. doi:10.1145/302405.302672.

- 17 E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982. doi:10.1016/0167-6423(83)90017-5.
- 18 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.
- 19 Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *Int. J. Softw. Tools Technol. Transf.*, 15(2):89–107, 2013. doi:10.1007/s10009-012-0244-z.
- 20 Rob J. van Glabbeek. Justness - A completeness criterion for capturing liveness properties (extended abstract). In Mikołaj ojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 505–522. Springer, 2019. doi:10.1007/978-3-030-17127-8_29.
- 21 Rob J. van Glabbeek. Reactive temporal logic. In Ornela Dardha and Jurriaan Rot, editors, *Proceedings Combined 27th International Workshop on Expressiveness in Concurrency and 17th Workshop on Structural Operational Semantics, EXPRESS/SOS 2020, and 17th Workshop on Structural Operational Semantics Online, 31 August 2020*, volume 322 of *EPTCS*, pages 51–68. Open Publishing Association, 2020. doi:10.4204/EPTCS.322.6.
- 22 Rob J. van Glabbeek. Modelling mutual exclusion in a process algebra with time-outs. *Inf. Comput.*, 294:105079, 2023. doi:10.1016/j.ic.2023.105079.
- 23 Rob J. van Glabbeek and Peter Höfner. CCS: It’s not fair! fair schedulers cannot be implemented in CCS-like languages even under progress and certain fairness assumptions. *Acta Informatica*, 52(2-3):175–205, 2015. doi:10.1007/s00236-015-0221-6.
- 24 Rob J. van Glabbeek and Peter Höfner. Progress, Justness, and Fairness. *ACM Comput. Surv.*, 52(4):69:1–69:38, 2019. doi:10.1145/3329125.
- 25 Jan Friso Groote and Jeroen J. A. Keiren. Tutorial: designing distributed software in mCRL2. In Kirstin Peters and Tim A. C. Willemse, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 41st IFIP WG 6.1 International Conference, FORTE 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, volume 12719 of *Lecture Notes in Computer Science*, pages 226–243. Springer, 2021. doi:10.1007/978-3-030-78089-0_15.
- 26 Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, August 2014. URL: <https://mitpress.mit.edu/books/modeling-and-analysis-communicating-systems>.
- 27 Lars Grunske. Specification patterns for probabilistic quality properties. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 31–40. ACM, 2008. doi:10.1145/1368088.1368094.
- 28 Sascha Konrad and Betty H. C. Cheng. Real-time specification patterns. In Gruia-Catalin Roman, William G. Griswold, and Bashar Nuseibeh, editors, *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 372–381. ACM, 2005. doi:10.1145/1062455.1062526.
- 29 Dexter Kozen. Results on the propositional μ -calculus. *Theor. Comput. Sci.*, 27(3):333–354, 1983. doi:10.1016/0304-3975(82)90125-6.
- 30 Leslie Lamport. Fairness and hyperfairness. *Distributed Comput.*, 13(4):239–245, 2000. doi:10.1007/PL00008921.
- 31 Radu Mateescu. Property Pattern Mappings for RAFMC, 2019. Available at: <https://cadp.inria.fr/resources/evaluator/rafmc.html> (Accessed: 26 January 2024).
- 32 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

- 33 Jaco van de Pol and Michael Weber. A multi-core solver for parity games. *Electronic Notes in Theoretical Computer Science*, 220(2):19–34, 2008. Proceedings of the 7th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC 2008). doi:10.1016/j.entcs.2008.11.011.
- 34 Daniela Remenska. *Bringing Model Checking Closer To Practical Software Engineering*. PhD thesis, Vrije U., Amsterdam, 2016. PhD Thesis, available at: <https://hdl.handle.net/1871/53958>.
- 35 Myrthe S. C. Spronck. Fairness assumptions in the modal μ -calculus, 2023. Master’s thesis, Eindhoven University of Technology, available at <https://research.tue.nl/en/studentTheses/fairness-assumptions-in-the-modal-%C2%B5-calculus>.
- 36 Frank A. Stomp, Willem-Paul de Roever, and Rob T. Gerth. The μ -calculus as an assertion-language for fairness arguments. *Inf. Comput.*, 82(3):278–322, 1989. doi:10.1016/0890-5401(89)90004-7.

A Proof Sketch

Full proofs are included in the appendices of the full version of this paper. Here, we provide an outline of the proof of Theorem 24 by presenting all the supporting lemmas and propositions proven. We include brief sketches of how we prove these claims, but not the full proofs. This is done to illustrate the approach we have taken. This appendix corresponds to Appendix D.3 in the full version. We begin with restating Theorem 24.

► **Theorem 24.** *For all feasible and finitely realisable path predicates P , it holds that an LTS satisfies Formula 2 if, and only if, its initial state does not admit \mathcal{B} -progressing paths that satisfy P and are $(\rho, \alpha_f, \alpha_e)$ -violating.*

The following propositions³ give properties of finitely realisable paths that can be derived from the invariant, locking, exclusive and persistent properties.

► **Proposition 46.** *Every \mathcal{B} -progressing, finite path satisfies every finitely realisable path predicate, for all $\mathcal{B} \subseteq \text{Act}$.*

► **Proposition 47.** *Every path that satisfies a finitely realisable path predicate P is \mathcal{B} -progressing.*

► **Proposition 48.** *If a path π satisfies finitely realisable path predicate P , then every path of which π is a suffix also satisfies P .*

► **Proposition 49.** *If a path π satisfies a finitely realisable path predicate P , then every suffix of π also satisfies P .*

Proposition 46 follows from the invariant, locking, and persistent properties. For Proposition 47, we use the invariant, locking, and exclusive properties, and for Proposition 48, the invariant and persistent property are enough. Finally, Proposition 49 follows directly from the invariant property.

For the proof of the main theorem, we fix an LTS M , as well as \mathcal{B} , ρ , α_f and α_e . We also fix a feasible, finitely realisable path predicate P . To characterise the semantics of the formula, we first split it into multiple smaller subformulae.

³ We use the same numbering here as in the full version, hence the jump to 46.

$$\begin{aligned}
 \text{violate}_G &= \langle \rho \rangle \text{invariant}_G \\
 \text{invariant}_G &= \nu X. \left(\bigwedge_{a \in \bar{\mathcal{B}}} (\phi_{on}(a) \Rightarrow \text{eliminate}_G(a)) \right) \\
 \text{eliminate}_G(a) &= \langle \bar{\alpha}_f^* \rangle (\langle \alpha_e \rangle tt \vee (\phi_{of}(a) \wedge X) \vee \langle \alpha_{el}(a) \setminus \alpha_f \rangle X)
 \end{aligned}$$

We have that Formula 2 = $\neg \text{violate}_G$.

The proof proceeds by characterising the semantics of every subformulae. We define the length of a path to be the number of transitions. A path of length 0 is called the empty path.

► **Lemma 50.** *For all environments e , states $s \in \mathcal{S}$, actions $a \in \bar{\mathcal{B}}$ and sets $\mathcal{F} \subseteq \mathcal{S}$, it holds that $s \in \llbracket \text{eliminate}_G(a) \rrbracket_{e[X:=\mathcal{F}]}$ if, and only if, s admits a finite path π with final state s_{final} that satisfies the following conditions:*

1. π is α_f -free, and
2. one of the following three holds:
 - a. at least one action in α_e is enabled in s_{final} , or
 - b. $s_{final} \in \mathcal{F}$ and s_{final} satisfies $\phi_{of}(a)$, or
 - c. $s_{final} \in \mathcal{F}$ and the last transition in π , t_{final} , is labelled with an action in $\alpha_{el}(a) \setminus \alpha_f$.

In the full proof of Lemma 50, we use another supporting proposition that characterises the semantics of a simple least fixpoint formula that generalises $\text{eliminate}_G(a)$, and then show in detail how the lemma follows. For this overview, we only give an intuitive explanation: the $\langle \bar{\alpha}_f^* \rangle$ part of $\text{eliminate}_G(a)$ gives us the finite, α_f -free path π that the lemma refers to. The conditions on the final state of this path follow from the rest of the formula: $\langle \alpha_e \rangle tt$ considers the possibility that an action in α_e is enabled; $\phi_{of}(a) \wedge X$ represents reaching a state in \mathcal{F} where $\phi_{of}(a)$ is satisfied (recall that Lemma 50 refers to the environment where X is mapped to \mathcal{F}); and $\langle \alpha_{el}(a) \setminus \alpha_f \rangle X$ appends one extra transition to a state in \mathcal{F} , eliminating a . Since the total path has to be α_f -free, the eliminating action may not be in α_f .

The next step is invariant_G . This formula exactly describes those states that admit paths that are \mathcal{B} -progressing, $(\varepsilon, \alpha_f, \alpha_e)$ -violating and satisfy P . Since ε is the empty sequence, we are ignoring the ρ -prefix for now. We define the set S_G to be exactly those states in M that admit a path π meeting the following conditions:

- π satisfies P , and
- π is \mathcal{B} -progressing, and
- π satisfies one of the following conditions:
 - π is α_f -free, or
 - π contains an occurrence of an action in α_e , and the prefix of π before the first occurrence of an action in α_e is α_f -free.

Our goal is then to prove that $\llbracket \text{invariant}_G \rrbracket_e = S_G$. This takes two steps: first we prove that S_G is a fixed point of the transformer characterising invariant_G , and then that it is the *greatest* fixed point.

► **Lemma 51.** *S_G is a fixed point of the transformer T_G defined by:*

$$T_G(\mathcal{F}) = \bigcap_{a \in \bar{\mathcal{B}}} \{s \in \mathcal{S} \mid s \in \llbracket \phi_{on}(a) \rrbracket_{e[X:=\mathcal{F}]} \Rightarrow s \in \llbracket \text{eliminate}_G(a) \rrbracket_{e[X:=\mathcal{F}]}\}$$

Proving that S_G is a fixed point means proving $T_G(S_G) = S_G$, which we do by mutual set inclusion. We briefly explain how we reach the conclusion that if a state s is in $T_G(S_G)$, then it must also be in S_G . If there are no non-blocking actions a such that $s \in \llbracket \phi_{on}(a) \rrbracket$, then the empty path witnesses that s is in S_G ; for this we use the locking property as well as Proposition 46. If there is an action $a \in \overline{\mathcal{B}}$ such that $s \in \llbracket \phi_{on}(a) \rrbracket$, then we know from $s \in T_G(S_G)$ that $s \in \llbracket \text{eliminate}_G \rrbracket_{e[X:=S_G]}$. Then Lemma 50 yields a finite path π that is α_f -free and on which a state in which α_e is enabled is reached, or a is eliminated and a state in S_G is reached. In the former case, we can use feasibility of P and Proposition 47 to find a path that satisfies P and is \mathcal{B} -progressing, and that is also α_f -free until the first occurrence of an action in α_e . Hence, we find a path that witnesses $s \in S_G$. If a is eliminated instead, then since π reaches a state that is in S_G , we can create a path $\pi'' = \pi \cdot \pi'$, where π' is a path from the final state of π that is \mathcal{B} -progressing, satisfies P , and is α_f -free up until the first occurrence of an action in α_e . Using Proposition 48, we can then show π'' witnesses $s \in S_G$. The other part of the proof, that an arbitrary state in S_G is also in $T_G(S_G)$, works very similarly, just in the other direction. We need Proposition 49 in that part of the proof in place of Proposition 48.

To prove that S_G is actually the greatest fixed point of T_G , we use the following supporting lemma:

► **Lemma 52.** *For all states s in a fixed point \mathcal{F} of T_G as defined in Lemma 51, if there is no action in α_e that is reachable from s without doing an action in α_f and there exists at least one action $a \in \overline{\mathcal{B}}$ such that s satisfies $\phi_{on}(a)$, then there exists a finite path π from s to some state s' meeting all of the following conditions:*

1. $s' \in \mathcal{F}$, and
2. π has length at least one, and
3. π is α_f -free, and
4. for all actions $a \in \overline{\mathcal{B}}$ such that s satisfies $\phi_{on}(a)$, there is a state on π that satisfies $\phi_{of}(a)$ or there is a transition on π labelled with an action in $\alpha_{el}(a)$.

However, for an intuitive explanation of the proof that S_G is the greatest fixed point of T_G we do not need this supporting lemma. We therefore do not go into its proof here.

► **Lemma 53.** *S_G is the greatest fixed point of the transformer T_G as defined in Lemma 51.*

In the proof of Lemma 53, we take an arbitrary state s in an arbitrary fixed point \mathcal{F} of T_G , and then prove that $s \in S_G$. This is done by constructing a path π from s that satisfies P , is \mathcal{B} -progressing, and $(\varepsilon, \alpha_f, \alpha_e)$ -violating. The proof considers three cases. The first case is when s is \mathcal{B} -locked. In that case, the empty path is trivially \mathcal{B} -progressing and violating, and by Proposition 46 also satisfies P . The second case is that it is possible to reach a state in which an action in α_e is enabled without taking actions in α_f . If this is the case, then that path can be extended using feasibility of P to create a path that is violating, satisfies P , and, by Proposition 47, is \mathcal{B} -progressing. The most complicated case is the one in which neither of the previous two is true. The idea is that we construct a path from s by repeatedly adding α_f -free path segments to an initial path $\pi = s$, in a potentially endless construction. In every iteration, we consider whether the final state of the path constructed thus far is \mathcal{B} -locked. If so, then, similar to the first case, we have found a witness for $s \in S_G$. If not, then there is some non-blocking action a that is “on”. And therefore, by the definition of T_G , there is a finite α_f -free path on which a is eliminated. We can disregard the possibility that we instead reach a state in which α_e is enabled, since we addressed that case separately. The segment we append to π is the finite α_f -free path on which a is eliminated. By the

38:22 Progress, Justness and Fairness in Modal μ -Calculus Formulae

persistent property, non-blocking actions for which ϕ_{on} is satisfied but are not eliminated remain “on”, and hence can be eliminated later in π . We can therefore simply keep track of all the actions for which ϕ_{on} is satisfied in states we encounter, and eliminate them all in turn. This produces an infinite, and therefore \mathcal{B} -progressing, path that satisfies P , and that is entirely α_f -free. So in this case too, we construct a path that witnesses $s \in S_G$.

Since the semantics of $invariant_G$ are characterised as the greatest fixed point of T_G , we can conclude the following from the definition of S_G .

► **Corollary 54.** *The set of states characterised by $invariant_G$ is exactly the set of states that admit \mathcal{B} -progressing, $(\varepsilon, \alpha_f, \alpha_e)$ -violating paths that satisfy P .*

We then prepend the ρ part of the formula.

► **Lemma 55.** *For all environments e and states $s \in \mathcal{S}$, it holds that $s \in \llbracket violate_G \rrbracket_e$ if, and only if, s admits a path that is \mathcal{B} -progressing, satisfies P and is $(\rho, \alpha_f, \alpha_e)$ -violating.*

This step is rather trivial, since it follows directly from Corollary 54 and the basic definition of the modal μ -calculus.

The final step of the proof is then to negate $violate_G$. Theorem 24 follows directly.

Coinductive Techniques for Checking Satisfiability of Generalized Nested Conditions

Lara Stoltenow ✉ 

University of Duisburg-Essen, Germany

Barbara König ✉ 

University of Duisburg-Essen, Germany

Sven Schneider ✉ 


Hasso Plattner Institute at the University of Potsdam, Germany

Andrea Corradini ✉ 

Università di Pisa, Italy

Leen Lambers ✉ 

Brandenburgische Technische Universität Cottbus-Senftenberg, Germany

Fernando Orejas ✉ 

Universitat Politècnica de Catalunya, Spain

Abstract

We study nested conditions, a generalization of first-order logic to a categorical setting, and provide a tableau-based (semi-decision) procedure for checking (un)satisfiability and finite model generation. This generalizes earlier results on graph conditions. Furthermore we introduce a notion of witnesses, allowing the detection of infinite models in some cases. To ensure completeness, paths in a tableau must be fair, where fairness requires that all parts of a condition are processed eventually. Since the correctness arguments are non-trivial, we rely on coinductive proof methods and up-to techniques that structure the arguments. We distinguish between two types of categories: categories where all sections are isomorphisms, allowing for a simpler tableau calculus that includes finite model generation; in categories where this requirement does not hold, model generation does not work, but we still obtain a sound and complete calculus.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Program reasoning

Keywords and phrases satisfiability, graph conditions, coinductive techniques, category theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.39

Related Version *Full Version*: <https://arxiv.org/abs/2407.06864> [28]

Funding *Andrea Corradini*: Research partially supported by the Italian MUR under the PRIN 20228KXFN2 “STENDHAL” and by the University of Pisa under the PRA 2022_99 “FM4HD”.

Fernando Orejas: Research partially supported by MCIN/ AEI /10.13039/501100011033 under grant PID2020-112581GB-C21.

1 Introduction

Nested graph conditions (called graph conditions subsequently) are a well-known specification technique for graph transformation systems [8] where they are used, e.g., to specify graph languages and application conditions. While their definition is quite different from first-order logic (FOL), they have been shown to be equivalent to FOL in [23, 8]. They are naturally equipped with operations such as shift, a form of partial evaluation, which is difficult to specify directly in FOL. This operation can be used to compute weakest preconditions and strongest postconditions for graph transformation systems [1].



© Lara Stoltenow, Barbara König, Sven Schneider, Andrea Corradini, Leen Lambers, and Fernando Orejas;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 39; pp. 39:1–39:20

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In [1] it has also been observed that graph conditions can be generalized to the categorical setting of reactive systems [16] as an alternative to the previously considered instantiation to graphs and injective graph morphisms that is equivalent to FOL. Further possible instantiations include cospan categories where the graphs, equipped with an inner and an outer interface, are the arrows, as well as Lawvere theories. To derive analysis techniques for all such instantiations, we consider nested conditions in the general categorical setting.

Here we are in particular interested in satisfiability checks on the general categorical level. As in FOL, satisfiability can be an undecidable problem (depending on the category), and we propose a semi-decision procedure that can simultaneously serve as a model finder. For FOL there are several well-known methods for satisfiability checking, for instance resolution or tableau proofs [6], while model generation is typically performed separately. The realization that satisfiability checking is feasible directly on graph conditions came in [18, 19], and a set of tableau rules was presented [18] without a proof of (refutational) completeness that was later published in [13], together with a model generation procedure [26]. A generalization to non-injective graph morphisms was given in [17].

The contributions of the current paper can be summarized as follows:

- We generalize the tableau-based semi-decision procedure for graph conditions from [13] to the level of general categories, under some mild constraints (such as the existence of so-called representative squares [1]). We present a procedure that has some resemblance to the construction of a tableau in FOL.
- We distinguish between two cases: one simpler case in which all sections (arrows that have a right inverse) in the category under consideration are isomorphisms (Section 3); and a more involved case where this does not necessarily hold (Section 5). The tableau rules of the former case (Section 3) are easier to present and implement, and we can give additional guarantees, such as model generation whenever there exists a so-called finitely decomposable model, generalizing the notion of finite models. The latter case (Section 5) does not guarantee model generation and has more involved tableau rules, but it allows for instantiations to more categories, such as graphs and arbitrary morphisms. The results of both cases generalize [13, 17, 26] from graphs and graph morphisms to an abstract categorical level, which allows application to additional categories such as cospan categories and Lawvere theories (see [28]).
- The completeness argument for the satisfiability checking procedure – in particular showing that non-termination implies the existence of an infinite model – requires that the tableau construction satisfies a fairness constraint. The resulting proof is non-trivial and – compared to the proof in [13] – we show that it can be reformulated using up-to techniques. Here we give it a completely new and hopefully clarifying structure that relies on coinductive methods [20, 22]. The alternative would be to inline the up-to techniques, or to rely on complex ad-hoc notation that are less clear and further complicate the proof.
- Furthermore we use coinductive techniques to display witnesses for infinite models (Section 4): in some cases where only infinite models exist and hence the tableau construction is non-terminating, we can still stop and determine that there does exist an infinite model. Coinductive techniques [24, 22] are reasoning techniques based on greatest fixpoints, suitable to analyze infinite or cyclic structures. To the best of our knowledge, such techniques have not yet been employed in the context of satisfiability checking for FOL and graph conditions.

The main contribution compared to previous work consists of a categorical generalization to reactive systems on the one hand, and the use of coinductive (up-to) techniques on the other hand. The implication of the first type of contribution is that the theory becomes available

for new instantiations such as adhesive categories (which includes all variants of graphs, such as typed graphs, Petri nets, but also algebraic specifications, cf. [3]), as well as other cases such as cospan categories and Lawvere theories. The second type of contribution implies that the proofs (especially for completeness) can now be presented in a more systematic way.

2 Preliminaries

2.1 Coinductive Techniques

A *complete lattice* is a partially ordered set (L, \sqsubseteq) where each subset $Y \subseteq L$ has a greatest lower bound, denoted by $\bigsqcap Y$ and a least upper bound, denoted by $\bigsqcup Y$.

A function $f: L \rightarrow L$ is *monotone* if for all $l_1, l_2 \in L$, $l_1 \sqsubseteq l_2$ implies $f(l_1) \sqsubseteq f(l_2)$, *idempotent* if $f \circ f = f$, and *extensive* if $l \sqsubseteq f(l)$ for all $l \in L$.

Given a monotone function $f: L \rightarrow L$ we are in particular interested in its *greatest fixpoint* νf . By Tarski's Theorem [29], $\nu f = \bigsqcup \{x \mid x \sqsubseteq f(x)\}$, i.e., the greatest fixpoint is the least upper bound of all post-fixpoints. Hence for showing that $l \sqsubseteq \nu f$ (for some $l \in L$), it is sufficient to prove that l is below some post-fixpoint l' , i.e., $l \sqsubseteq l' \sqsubseteq f(l')$.

In order to employ up-to techniques one defines a monotone function $u: L \rightarrow L$ (the up-to function) and checks whether u is *f-compatible*, that is $u \circ f \sqsubseteq f \circ u$. If that holds every post-fixpoint l of $f \circ u$ (that is $l \sqsubseteq f(u(l))$) is below the greatest fixpoint of f ($l \sqsubseteq \nu f$). This simple technique can often greatly simplify checking whether a given element is below the greatest fixpoint. For more details see [20].

2.2 Categories

We will use standard concepts from category theory. Given an arrow $f: A \rightarrow B$, we write $\text{dom}(f) = A$, $\text{cod}(f) = B$. For two arrows $f: A \rightarrow B$, $g: B \rightarrow C$ we denote their composition by $f;g: A \rightarrow C$. An arrow $s: A \rightarrow B$ is a *section* (also known as *split mono*) if there exists $r: B \rightarrow A$ such that $s;r = \text{id}$. That is, sections are those arrows s that have a right-inverse r . Arrows that have a left-inverse (in this case r) are called *retractions*.

As in graph rewriting we will consider the category $\mathbf{Graph}_{\text{fin}}$, which has finite graphs as objects and graph morphisms as arrows. We also consider $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$, the subcategory of $\mathbf{Graph}_{\text{fin}}$ that has the same objects, but only injective, i.e. mono, graph morphisms. In this category the sections are exactly the isos, while this is not the case in $\mathbf{Graph}_{\text{fin}}$.

Another important example category that will be used in Section 4 is based on cospans: note that reactive systems instantiated with cospans [11, 25, 27] yield exactly double-pushout rewriting [5]. Given a base category \mathbf{D} with pushouts, the category $\mathbf{Cospan}(\mathbf{D})$ has as objects the objects of \mathbf{D} and as arrows *cospans*, which are equivalence classes of pairs of arrows of the form $A \xrightarrow{f_L} X \xleftarrow{f_R} B$, where the middle object is considered up to isomorphism. Cospan composition is performed via pushouts (for details see Appendix A).

A cospan is *left-linear* if its left leg f_L is mono. For adhesive categories [12], the composition of left-linear cospans again yields a left-linear cospan, and $\mathbf{ILC}(\mathbf{D})$ is the subcategory of $\mathbf{Cospan}(\mathbf{D})$ where the arrows are restricted to left-linear cospans.

Note that $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ can be embedded into $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}})$ by transforming an injective graph morphism f to a left-linear cospan with f as the left leg and id as the right leg.

Another application are Lawvere theories, where arrows are (tuples of) terms, an approach we explore in the full version of this paper [28].

2.3 Generalized Nested Conditions

We consider (nested) conditions over an arbitrary category \mathbf{C} in the spirit of reactive systems [16, 15]. Following [23, 8], we define conditions as finite tree-like structures, where nodes are annotated with quantifiers and objects, and edges are annotated with arrows.

► **Definition 1 (Condition).** *Let \mathbf{C} be a category. A condition \mathcal{A} over an object A in \mathbf{C} is defined inductively as follows: it is either*

- *a finite conjunction of universals $\bigwedge_{i \in \{1, \dots, n\}} \forall f_i. \mathcal{A}_i = \forall f_1. \mathcal{A}_1 \wedge \dots \wedge \forall f_n. \mathcal{A}_n$, or*
 - *a finite disjunction of existentials $\bigvee_{i \in \{1, \dots, n\}} \exists f_i. \mathcal{A}_i = \exists f_1. \mathcal{A}_1 \vee \dots \vee \exists f_n. \mathcal{A}_n$*
- where $f_i: A \rightarrow A_i$ are arrows in \mathbf{C} and $\mathcal{A}_i \in \text{Cond}(A_i)$ are conditions. We call $A = \text{RO}(\mathcal{A})$ the root object of the condition \mathcal{A} . Each subcondition $\mathcal{Q}f_i. \mathcal{A}_i$ ($\mathcal{Q} \in \{\forall, \exists\}$) is called a child of \mathcal{A} . The constants true_A (empty conjunction) and false_A (empty disjunction) serve as the base cases. We will omit subscripts in true_A and false_A when clear from the context.

The set of all conditions over A is denoted by $\text{Cond}(A)$.

Instantiated with $\mathbf{Graph}_{\text{fin}}$ respectively $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$, conditions are equivalent to graph conditions as defined in [8], and equivalence to first-order logic has been shown in [23]. Intuitively, these conditions require the existence of certain subgraphs or patterns, or that whenever a given subgraph occurs, the surroundings of the match satisfy a child condition. For instance, $\forall \emptyset \rightarrow \textcircled{1} \rightarrow \textcircled{2} . \exists \textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{1} \leftarrow \textcircled{2} . \text{true}$ requires that for every edge, a second edge in the reverse direction also exists. For additional examples of conditions we refer to Examples 19, 24, and 27 given later.

To simplify our algorithms and their proofs, the definition of conditions requires that conjunctions contain only universal children and disjunctions only existential children (e.g., $\exists f. \mathcal{A} \wedge \exists g. \mathcal{B}$ is excluded). However, this can be simulated using $\forall \text{id}. \exists f. \mathcal{A} \wedge \forall \text{id}. \exists g. \mathcal{B}$, and similarly for disjunctions of universals. Hence we sometimes write $\mathcal{A} \wedge \mathcal{B}$ or $\mathcal{A} \vee \mathcal{B}$ for arbitrary conditions in the proofs.

While in [1] a model for a condition was a single arrow, we have to be more general, since there are some satisfiable conditions that have no finite models. Here we want to work in categories of finite graphs (so that conditions are finite), but at the same time we want to consider infinite models. The solution is to evaluate conditions on infinite sequences of arrows $\bar{a} = [a_1, a_2, a_3, \dots]$, where $A \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \dots$, called *composable sequences*.¹ We define $\text{dom}(\bar{a}) = \text{dom}(a_1) = A$ and we call such a sequence *finite* iff for some index k all a_i with $i > k$ are identities.

Intuitively, the model is represented by the “composition” of the infinite sequence of arrows. In the category $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ this would amount to taking the limit of this sequence. As we will later see, it does not play a role how exactly an infinite structure is decomposed into arrows, as all decompositions are equivalent with respect to satisfaction.

► **Definition 2 (Satisfaction).** *Let $\mathcal{A} \in \text{Cond}(A)$. Let $\bar{a} = [a_1, a_2, a_3, \dots]$ be a composable sequence with $A = \text{dom}(\bar{a})$. We define the satisfaction relation $\bar{a} \models \mathcal{A}$ as follows:*

- $\bar{a} \models \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$ iff for every $i \in I$ and every arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and all $n \in \mathbb{N}_0$ we have: if $a_1; \dots; a_n = f_i; g$, then $[g, a_{n+1}, \dots] \models \mathcal{A}_i$.
- $\bar{a} \models \bigvee_i \exists f_i. \mathcal{A}_i$ iff there exists $i \in I$ and an arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and some $n \in \mathbb{N}_0$ such that $a_1; \dots; a_n = f_i; g$ and $[g, a_{n+1}, \dots] \models \mathcal{A}_i$.

¹ Another option would be to work in the category of potentially infinite graphs. However, that would allow conditions based on infinite graphs for which satisfiability checks become algorithmically infeasible.

Note that this covers the base cases ($\bar{a} \models \text{true}$, $\bar{a} \not\models \text{false}$ for every sequence \bar{a}). Furthermore $a_1; \dots; a_n$ equals the identity whenever $n = 0$. For a finite sequence $\bar{a} = [a_1, \dots, a_k, \text{id}, \text{id}, \dots]$ this means that $\bar{a} \models \mathcal{A}$ iff $a = a_1; \dots; a_k$ is a model for \mathcal{A} according to the definition of satisfaction given in [1]. In this case we write $[a_1, \dots, a_k] \models \mathcal{A}$ or simply $a \models \mathcal{A}$.

► **Remark 3.** In the following we use **Cond** to denote the set² of all conditions and **Seq** as the set of all composable sequences of arrows (potential models). ◻

We write $\mathcal{A} \models \mathcal{B}$ (\mathcal{A} implies \mathcal{B}) if $\text{RO}(\mathcal{A}) = \text{RO}(\mathcal{B})$ and for every \bar{a} with $\text{dom}(\bar{a}) = \text{RO}(\mathcal{A})$ we have: if $\bar{a} \models \mathcal{A}$, then $\bar{a} \models \mathcal{B}$. Two conditions are equivalent ($\mathcal{A} \equiv \mathcal{B}$) if $\mathcal{A} \models \mathcal{B}$ and $\mathcal{B} \models \mathcal{A}$.

Every condition can be transformed to an equivalent condition that alternates between \forall, \exists by inserting $\forall \text{id}$ or $\exists \text{id}$ as needed. Such conditions are called *alternating*.

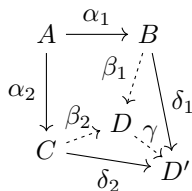
We also define what it means for two conditions to be isomorphic. It is easy to see that isomorphic conditions are equivalent, but not necessarily vice versa.

► **Definition 4 (Isomorphic Conditions).** For conditions \mathcal{A}, \mathcal{B} and an iso $h: \text{RO}(\mathcal{B}) \rightarrow \text{RO}(\mathcal{A})$, we say that \mathcal{A}, \mathcal{B} are isomorphic ($\mathcal{A} \cong \mathcal{B}$) wrt. h , whenever both are universal, i.e., $\mathcal{A} = \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$, $\mathcal{B} = \bigwedge_{j \in J} \forall g_j. \mathcal{B}_j$, and for each $i \in I$ there exists $j \in J$ and an iso $h_{j,i}: \text{RO}(\mathcal{B}_j) \rightarrow \text{RO}(\mathcal{A}_i)$ such that $h; f_i = g_j; h_{j,i}$ and $\mathcal{A}_i \cong \mathcal{B}_j$ wrt. $h_{j,i}$; and vice versa (for each $j \in J$ there exists $i \in I \dots$). Analogously if both conditions are existential.

2.4 Representative Squares and the Shift Operation

We will now define the notion of representative squares, which describe representative ways to close a span of arrows. They generalize idem pushouts [16] and borrowed context diagrams [4].

► **Definition 5 (Representative squares [1]).** A class κ of commuting squares in a category \mathbf{C} is representative if for every commuting square $\alpha_1; \delta_1 = \alpha_2; \delta_2$ in \mathbf{C} there exists a representative square $\alpha_1; \beta_1 = \alpha_2; \beta_2$ in κ and an arrow γ such that $\delta_1 = \beta_1; \gamma$ and $\delta_2 = \beta_2; \gamma$.



For two arrows $\alpha_1: A \rightarrow B$, $\alpha_2: A \rightarrow C$, we define $\kappa(\alpha_1, \alpha_2)$ as the set of pairs of arrows (β_1, β_2) which, together with α_1, α_2 , form representative squares in κ .

Compared to weak pushouts, more than one square might be needed to represent all commuting squares that extend a given span (α_1, α_2) . In categories with pushouts (such as $\mathbf{Graph}_{\text{fin}}$), pushouts are the most natural candidate for representative squares. In $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ pushouts do not exist, but jointly epi squares can be used instead. For cospan categories, one can use borrowed context diagrams [4] (see Appendix A for a summary).

For many categories of interest – such as $\mathbf{Graph}_{\text{fin}}$ and $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}})$ – we can guarantee a choice of κ such that each set $\kappa(\alpha_1, \alpha_2)$ is finite and computable. In the rest of this paper, we assume that we work in such a category, and use such a class κ . Hence the constructions described below are effective since the finiteness of the transformed conditions is preserved.

² Actually, without restrictions these are proper classes rather than sets. We tacitly assume that we are working in the corresponding skeleton category where no two different objects are isomorphic and assume that we can consider **Cond**, **Seq** as sets.

One central operation is the shift of a condition along an arrow. The name shift is taken from an analogous operation for nested application conditions (see [19]).

► **Definition 6** (Shift of a Condition). *Given a fixed class of representative squares κ , the shift of a condition \mathcal{A} along an arrow $c: \text{RO}(\mathcal{A}) \rightarrow B$ is inductively defined as follows:*

$$\left(\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \right)_{\downarrow c} = \bigwedge_{i \in I} \bigwedge_{(\alpha, \beta) \in \kappa(f_i, c)} \forall \beta. (\mathcal{A}_i)_{\downarrow \alpha}$$

$$\begin{array}{ccc} & f_i & \\ c \downarrow & \xrightarrow{\quad} & \downarrow \alpha \\ & \beta & \end{array}$$

Shifting of existential conditions is performed analogously.

The shift operation can be understood as a partial evaluation of \mathcal{A} under the assumption that c is already “present”. In particular it satisfies $[c; d_1, d_2, \dots] \models \mathcal{A} \iff [d_1, d_2, \dots] \models \mathcal{A}_{\downarrow c}$. This implies that while the representation of the shifted condition may differ depending on the chosen class of representative squares, the resulting conditions are equivalent. Since we assume that each set $\kappa(f_i, c)$ is finite, shifting a finite condition will again result in a finite condition.

As an example in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$, shifting $\forall \emptyset \rightarrow \textcircled{1}. \exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}. \text{true}$ (every node has an outgoing edge) over $\emptyset \rightarrow \textcircled{1}$ (a node exists) yields $\forall \textcircled{1} \rightarrow \textcircled{1}. \exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{1}. \text{true} \wedge \forall \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{1}. (\exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}). \text{true} \vee \exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{1}. \text{true}$ (the designated node has an outgoing edge, and so does every other node, possibly to the designated node).

In the case where α_1 in Definition 5 is an iso, we can always assume that $\kappa(\alpha_1, \alpha_2) = \{(\alpha_1^{-1}; \alpha_2, \text{id})\}$ and we will use this assumption in the paper.

2.5 Further Concepts

Our goal is to develop a procedure that finds a finite model if one exists, produces unsatisfiability proofs if a condition has neither finite nor infinite models, and otherwise does not terminate. In order to state the correctness of this procedure, we will need an abstract notion of finiteness and to this aim we introduce *finitely decomposable morphisms*. Intuitively this means that every infinite decomposition contains only finitely many non-isomorphisms.

► **Definition 7** (Finitely decomposable morphism). *A morphism $m: A \rightarrow B$ is finitely decomposable if for every infinite sequence of (f_i, g_i) , $i \in \mathbb{N}_0$, such that $f_0 = m$ and $f_i = g_i; f_{i+1}$ (cf. the diagram below), only finitely many g_i are non-isomorphisms.*

$$\begin{array}{ccccc} A & \xrightarrow{g_0} & \xrightarrow{g_1} & \cdots & \\ m = f_0 \downarrow & \searrow f_1 & \searrow f_2 & \searrow & \\ B & & & & \end{array}$$

Note that in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ all arrows are finitely decomposable, while this is not the case in $\mathbf{Graph}_{\text{fin}}$. In $\mathbf{Graph}_{\text{fin}}$, there exists a section s (with associated retraction r such that $s; r = \text{id}$) that is *not* an iso (example: $s = \textcircled{1} \rightarrow \textcircled{1} \textcircled{2}$, $r = \textcircled{1} \textcircled{2} \rightarrow \textcircled{1}$). Then, the identity on the domain of s has a decomposition into infinitely many non-isos (an alternating sequence of s and r , more concretely: $g_{2i} = s$, $g_{2i+1} = r$ and $f_{2i} = \text{id}$, $f_{2i+1} = r$) and is hence not finitely decomposable.

While satisfaction is typically defined inductively (as in Definition 2), i.e., as a least fixpoint, we can also view it coinductively, i.e., as a greatest fixpoint, due to the fact that all conditions are finite.

► **Proposition 8** (Fixpoint function for satisfaction). *Let $\bar{a} = [a_1, a_2, a_3, \dots] \in \text{Seq}$ be a composable sequence of arrows. We define the function $s: \mathcal{P}(\text{Seq} \times \text{Cond}) \rightarrow \mathcal{P}(\text{Seq} \times \text{Cond})$ as follows: Let $P \subseteq \text{Seq} \times \text{Cond}$, then*

- $(\bar{a}, \bigwedge_i \forall f_i. \mathcal{A}_i) \in s(P)$ iff for every $i \in I$ and every arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and all $n \in \mathbb{N}_0$ we have: if $a_1; \dots; a_n = f_i; g$, then $([g, a_{n+1}, \dots], \mathcal{A}_i) \in P$.
- $(\bar{a}, \bigvee_i \exists f_i. \mathcal{A}_i) \in s(P)$ iff there exists $i \in I$ and an arrow $g: \text{RO}(\mathcal{A}_i) \rightarrow B$ and some $n \in \mathbb{N}_0$ such that $a_1; \dots; a_n = f_i; g$ and $([g, a_{n+1}, \dots], \mathcal{A}_i) \in P$.

The least and greatest fixpoint of s ($\mu s, \nu s$) coincide and they equal the satisfaction relation \models .

3 Satisfiability Checking in the Restricted Case

Given a condition \mathcal{A} , we want to know whether \mathcal{A} is satisfiable and generates a finitely decomposable model, if it exists. Here we provide a procedure that works under the assumption that we are working in a category where all sections are isos. This is for instance true for $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ and $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, where non-trivial right-inverses do not exist. It does not hold for non-injective graph morphisms (see counterexample above) or left-linear cospans (counterexample: $\text{id} = \textcircled{1} \rightarrow \textcircled{1} \leftarrow \textcircled{1} \textcircled{2}$; $\textcircled{1} \textcircled{2} \rightarrow \textcircled{1} \textcircled{2} \leftarrow \textcircled{1}$).

The general case where this assumption does not hold will be treated in Section 5.

3.1 Tableau Calculus

The underlying idea is fairly straightforward: we take an alternating condition \mathcal{A} and whenever it is existential, that is $\mathcal{A} = \bigvee_{i \in I} \exists f_i. \mathcal{A}_i$, we branch and check whether some \mathcal{A}_i is satisfiable. If instead it is universal, i.e., $\mathcal{A} = \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$, we check whether some f_i is an iso. If that is not the case, clearly the sequence of identities on $\text{RO}(\mathcal{A})$ is a model, since there is no arrow g such that $\text{id} = f_i; g$, assuming that all sections are isos. If however some f_i is an iso, we invoke a pull-forward rule (see below for more details) that transforms the universal condition into an existential condition and we continue from there. We will show that this procedure works whenever the pull-forward follows a *fair* strategy: in particular every iso (respectively one of its successors) must be pulled forward eventually.

The pull-forward rule relies on the equivalence $(\mathcal{A} \wedge \exists f. \mathcal{B}) \equiv \exists f. (\mathcal{A} \downarrow_f \wedge \mathcal{B})$.

► **Lemma 9** (Pulling forward isomorphisms). *Let $\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i$ be a universal condition and assume for some $p \in I$, f_p is an iso and $\mathcal{A}_p = \bigvee_{j \in J} \exists g_j. \mathcal{B}_j$. Then f_p can be pulled forward:*

$$\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \equiv \exists f_p. \bigvee_{j \in J} \exists g_j. \left(\mathcal{B}_j \wedge \left(\bigwedge_{m \in I \setminus \{p\}} \forall f_m. \mathcal{A}_m \right) \downarrow_{f_p; g_j} \right)$$

► **Definition 10** (SatCheck tableau construction rules). *Given an alternating condition \mathcal{A} , we give rules for the construction of a tableau for \mathcal{A} that has conditions as nodes, \mathcal{A} as root node, and edges (\rightarrow) labeled with arrows. The tableau is extended at its leaf nodes as follows:*

$$\left. \begin{array}{l} \text{For every } p \in I: \\ \bigvee_{i \in I} \exists f_i. \mathcal{A}_i \xrightarrow{f_p} \mathcal{A}_p \end{array} \right| \begin{array}{l} \text{For one } p \in I \text{ such that } f_p \text{ is iso and } \mathcal{A}_p = \bigvee_{j \in J} \exists g_j. \mathcal{B}_j: \\ \bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \xrightarrow{f_p} \underbrace{\bigvee_{j \in J} \exists g_j. \mathcal{B}_j}_{\mathcal{A}_p} \wedge \underbrace{\left(\bigwedge_{m \in I \setminus \{p\}} \forall f_m. \mathcal{A}_m \right) \downarrow_{f_p; g_j}}_{\text{other children, shifted to include } g_j} \end{array}$$

- For existential conditions, for each(!) child condition $\exists f_p. \mathcal{A}_p$, add a new descendant.
- For universal conditions, non-deterministically pick one(!) child condition $\forall f_p. \mathcal{A}_p$ that can be pulled forward (f_p is an iso), pull it forward (cf. Lemma 9), and add the result as its (only) descendant. If a universal condition contains no isos, then add no descendant.

A branch of a tableau is a (potentially infinite) path $\mathcal{A}_0 \xrightarrow{u_1} \mathcal{A}_1 \xrightarrow{e_1} \mathcal{A}_2 \xrightarrow{u_2} \dots$ starting from the root node. A finite branch is extendable if one of the tableau construction rules is applicable at its leaf node and would result in new nodes (hence, a branch where the leaf is an empty existential or a universal without isomorphisms is not extendable). A branch is closed if it ends with an empty existential, otherwise it is open.

Due to Lemma 9 each universal condition is (up to iso f_p) equivalent to its (unique) descendant (if one exists), while an existential condition is equivalent to the disjunction of its descendants prefixed with existential quantifiers.

The labels along one branch of the tableau are arrows between the root objects of the conditions. Their composition corresponds to the prefix of a potential model being constructed step by step. Finite paths represent a model if they are open and not extendable. For infinite paths, we need an additional property to make sure that the procedure does not “avoid” a possibility to show unsatisfiability of a condition.

To capture that, we introduce the notion of fairness, meaning that all parts of a condition are eventually used in a proof and are not postponed indefinitely (a related concept is saturation, see e.g. [13], though the definition deviates due to a different setup). For this we first need to track how pulling forward one child condition changes the other children by shifting. We define a *successor relation* that, for each pair of \forall -condition and one of its \forall -grandchildren, relates child conditions of the \forall -condition to their shifted counterparts (the *successors*) in the \forall -condition of the second-next nesting level. The successor relation is similar in spirit to the one used in [13]. In this work, saturation is given in a more descriptive way and has to account for nesting levels in the tableau, a complication that we were able to avoid in the present paper.

► **Definition 11** (Successor relation). *Assume in the construction of a tableau we have a path*

$$\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \xrightarrow{f_p} \mathcal{C} \xrightarrow{g_j} \mathcal{B}_j \wedge \left(\bigwedge_{m \in I \setminus \{p\}} \forall f_m. \mathcal{A}_m \right) \downarrow_{f_p; g_j} = \mathcal{B}_j \wedge \bigwedge_{m \in I \setminus \{p\}} \bigwedge_{(\alpha, \beta) \in \kappa(f_m, f_p; g_j)} \forall \beta. (\mathcal{A}_m)_{\downarrow \alpha}$$

where \mathcal{C} is the existential condition given in Definition 10. Then for each $m \in I \setminus \{p\}$, each $\forall \beta. (\mathcal{A}_m)_{\downarrow \alpha}$ where $(\alpha, \beta) \in \kappa(f_m, f_p; g_j)$ is a successor of $\forall f_m. \mathcal{A}_m$. The transitive closure of the successor relation induces the indirect successor relation.

► **Definition 12** (Fairness). *An infinite branch of a tableau is fair if for each universal condition \mathcal{A} on the branch and each child condition $\forall f_i. \mathcal{A}_i$ of \mathcal{A} where f_i is an iso, it holds that some indirect successor of $\forall f_i. \mathcal{A}_i$ is eventually pulled forward.*

► **Remark 13** (Fairness strategies). One possible strategy that ensures fairness is to maintain for each incomplete branch a queue of child conditions for which a successor must be pulled forward. Then the first entry in this queue is processed. Note that by the assumption on κ made earlier at the end of Section 2.4, each iso in a universal condition that is not pulled forward has exactly one successor and the queue is modified by replacing each condition accordingly and adding newly generated child conditions with isos at the end. \lrcorner

3.2 Up-To Techniques, Fair Branches and Models

While showing soundness of the tableau method is relatively straightforward, the crucial part of the completeness proof is to show that every infinite and fair branch of the tableau corresponds to a model. The proof strategy is the following: given such a branch, we aim to construct a witness for this model, by pairing conditions on this path with the suffix

consisting of the sequence of arrows starting from this condition. If one could show that the set $P \subseteq \text{Seq} \times \text{Cond}$ of pairs so obtained is a post-fixpoint of the satisfaction function s defined in Proposition 8 ($P \subseteq s(P)$), we could conclude, as the satisfaction relation \models is the greatest fixpoint of s (Proposition 8) and hence above any post-fixpoint.

However, P is in general not a post-fixpoint, which is mainly due to the fact that universal conditions are treated “sequentially” one after another and are “pulled forward” only if they become isos. Hence, if we want to show that for a chain $[a_1, a_2, \dots]$ the universal condition of the form $\bigwedge_i \forall f_i. \mathcal{A}_i$ is satisfied, we have to prove for every child $\forall f_i. \mathcal{A}_i$ that whenever $a_1; \dots; a_n = f_i; g$ it holds that $[g, a_{n+1}, \dots] \models \mathcal{A}_i$. If $\forall f_i. \mathcal{A}_i$ actually is the child that is pulled forward in the next tableau step, P contains the tuple required by s . If not, there is a “delay” and intuitively that means that we can not guarantee that P is indeed a post-fixpoint.

However it turns out that it is a post-fixpoint up-to ($P \subseteq s(u(P))$), where u is a combination of one or more suitable up-to functions. We first explore several such up-to functions: $u_\wedge(P)$ obtains new conditions by non-deterministically removing parts of conjunctions; with $u_\S(P)$ we can arbitrarily recompose (decompose and compose) the arrows in a potential model; with $u_\downarrow(P)$ we can undo a shift; and $u_\cong(P)$ allows replacing conditions with isomorphic conditions. For each, we show their s -compatibility (i.e., $u(s(P)) \subseteq s(u(P))$).

► **Theorem 14 (Up-to techniques).** *Let $P \subseteq \text{Seq} \times \text{Cond}$, i.e. tuples of potential model and condition. Then the following four up-to functions are s -compatible:*

- *Conjunction removal: We inductively define a relation \mathcal{U}_\wedge containing a pair of conditions $(\mathcal{A}, \mathcal{T})$ iff \mathcal{T} is the same as \mathcal{A} but with some conjunctions removed. That is, \mathcal{U}_\wedge contains*

$$\begin{aligned} & (\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i, \bigwedge_{j \in J \subseteq I} \forall f_j. \mathcal{T}_j) \quad \text{whenever } (\mathcal{A}_j, \mathcal{T}_j) \in \mathcal{U}_\wedge \text{ for all } j \in J \\ & (\bigvee_{i \in I} \exists f_i. \mathcal{A}_i, \bigvee_{i \in I} \exists f_i. \mathcal{T}_i) \quad \text{whenever } (\mathcal{A}_i, \mathcal{T}_i) \in \mathcal{U}_\wedge \text{ for all } i \in I \end{aligned}$$

Then define: $u_\wedge(P) = \{(\bar{c}, \mathcal{T}) \mid (\bar{c}, \mathcal{A}) \in P, (\mathcal{A}, \mathcal{T}) \in \mathcal{U}_\wedge\}$

- *Recomposition: $u_\S(P) = \{([b_1, \dots, b_\ell, \bar{c}], \mathcal{A}) \mid ([a_1, \dots, a_k, \bar{c}], \mathcal{A}) \in P, a_1; \dots; a_k = b_1; \dots; b_\ell\}$*
- *Shift: $u_\downarrow(P) = \{([c; c_1, c_2, \dots], \mathcal{B}) \mid ([c_1, c_2, \dots], \mathcal{B}_{\downarrow c}) \in P\}$*
- *Isomorphic condition: $u_\cong(P) = \{([h; c_1, c_2, \dots], \mathcal{B}) \mid ([c_1, c_2, \dots], \mathcal{A}) \in P, \mathcal{A} \cong \mathcal{B} \text{ with iso } h: \text{RO}(\mathcal{B}) \rightarrow \text{RO}(\mathcal{A})\}$*

Note however that up-to equivalence u_\cong is *not* a valid up-to technique: let \mathcal{U} be an unsatisfiable condition and let $P = \{([\text{id}, \dots], \forall \text{id.} \mathcal{U})\}$. As $\mathcal{U} \equiv \forall \text{id.} \mathcal{U}$, then also $([\text{id}, \dots], \mathcal{U}) \in u_\cong(P)$ and hence $P \subseteq s(u_\cong(P))$. If the technique were correct, this would imply $\text{id} \models \mathcal{U}$.

A convenient property of compatibility is that it is preserved by various operations, in particular, composition, union and iteration ($f^\omega = \bigcup_{i \in \mathbb{N}_0} f^i$). This can be used to combine multiple up-to techniques into a new one that also has the compatibility property. [21]

► **Lemma 15 (combining up-to techniques).** *Let $u = (u_\S \cup u_\wedge \cup u_\downarrow \cup u_\cong)^\omega$ be the iterated application of the up-to techniques from Theorem 14, then u is s -compatible.*

We are now able to prove the central theorem needed for showing completeness.

► **Theorem 16 (Fair branches are models).** *Let \mathcal{A}_0 be an alternating condition. Let a fixed tableau constructed by the rules of Definition 10 be given. Let $\mathcal{A}_0 \xrightarrow{b_1} \mathcal{A}_1 \xrightarrow{b_2} \mathcal{A}_2 \xrightarrow{b_3} \dots$ be a branch of the tableau that is either not extendable and ends with a universal quantification (i.e., it is open), or is infinite and fair. For such a branch, we define $P = \{(\bar{b}, \mathcal{A}_i) \mid i \in \mathbb{N}_0, \bar{b} = [b_{i+1}, b_{i+2}, \dots]\} \subseteq \text{Seq} \times \text{Cond}$, i.e., the relation P pairs suffixes of the branch with the corresponding conditions. Finally, let u be the combination of up-to techniques defined in Lemma 15. Then, $P \subseteq s(u(P))$, which implies that $P \subseteq \models$. In other words, every such branch in a tableau of Definition 10 corresponds to a model of \mathcal{A}_0 .*

Proof sketch. Let $([c_1, c_2, \dots], \mathcal{C}_0) \in P$, which corresponds to a suffix $\mathcal{C}_0 \xrightarrow{c_1} \mathcal{C}_1 \xrightarrow{c_2} \mathcal{C}_2 \xrightarrow{c_3} \dots$ of the chosen branch. We show that $([c_1, c_2, \dots], \mathcal{C}_0) \in s(u(P))$:

- **\mathcal{C}_0 is existential:** The next label on the branch is the arrow of some child $\exists c_1.\mathcal{C}_1$ of \mathcal{C}_0 and $([c_2, c_3, \dots], \mathcal{C}_1) \in P$ implies $([c_1, c_2, \dots], \mathcal{C}_0) \in s(P) \subseteq s(u(P))$.
- **\mathcal{C}_0 is universal and contains no iso:** The branch ends at this point, so the sequence of arrows in the tuple is empty and represents id , where $(\text{id}, \mathcal{C}_0) \in s(u(P))$: no f_i is an iso, therefore $f_i; g = \text{id}$ is never true and hence the universal condition is trivially satisfied.
- **\mathcal{C}_0 is universal and contains at least one iso:** By definition of s , we need to be able to satisfy any given child $\forall d_0.\mathcal{D}_0$, i.e., show that whenever $c_1; \dots; c_n = d_0; g$ for some g, n , then $([g, c_{n+1}, \dots], \mathcal{D}_0) \in u(P)$.

Fairness guarantees that an indirect successor $\forall d_q.\mathcal{D}_q$ of $\forall d_0.\mathcal{D}_0$ is pulled forward, which results (after up-to conjunction removal) in a tuple $([c_m, \dots], \mathcal{D}_q) \in u(P)$. The intermediate steps on the branch, where other children are pulled forward instead, allow expressing \mathcal{D}_q as $(\mathcal{D}_0)_{\downarrow \alpha_1 \downarrow \dots \downarrow \alpha_q}$. Use up-to shift to transform to $([\alpha_1; \dots; \alpha_q; c_m, \dots], \mathcal{D}_0) \in u(P)$, then up-to recomposition to the required $([g_0, c_{n+1}, \dots], \mathcal{D}_0) \in u(P)$. ◀

3.3 Soundness and Completeness

We are finally ready to show soundness and completeness of our method.

As a condition is essentially equivalent to any of its tableaux, which break it down into existential subconditions, a closed tableau represents an unsatisfiable condition.

► **Theorem 17 (Soundness).** *If there exists a tableau T for a condition \mathcal{A} where all branches are closed, then the condition \mathcal{A} in the root node is unsatisfiable.*

Proof sketch. By induction over the depth of T . Base case is false (obviously unsatisfiable). Induction step for $\exists: \bigvee_i \exists f_i.\mathcal{A}_i$ is unsatisfiable if all \mathcal{A}_i are. For \forall : by construction, the only child contains an equivalent condition. ◀

We now prove completeness, which – to a large extent – is a corollary of Theorem 16.

► **Theorem 18 (Completeness).** *If a condition \mathcal{A} is unsatisfiable, then every tableau constructed by obeying the fairness constraint is a finite tableau where all branches are closed. Furthermore, at least one such tableau exists.*

Proof. The contraposition follows from Theorem 16: If the constructed tableau is finite with open branches or infinite, then \mathcal{A} is satisfiable. Furthermore, a fair tableau must exist and can be constructed by following the strategy described in Remark 13. ◀

In the next section we will show how the open branches in a fully expanded tableau can be interpreted as models, thus giving us a procedure for model finding.

► **Example 19 (Proving unsatisfiability).** We work in $\mathbf{Graph}_{\text{fn}}^{\text{inj}}$. For this example, we use the following shorthand notation for graph morphisms: $[\textcircled{1} \rightarrow \textcircled{2}]$ means $\textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}$, i.e., the morphism is the inclusion from the light-gray graph elements to the full graph.

Consider the condition $\mathcal{A} = \forall[\emptyset].\exists[\textcircled{1}].\text{true} \wedge \forall[\textcircled{2}].\text{false}$, meaning (1) there exists a node and (2) no node must exist. It is easily seen that these contradict each other and hence \mathcal{A} is unsatisfiable. We obtain a tableau with a single branch for this condition:

$$\mathcal{A} \xrightarrow{[\emptyset]} \exists[\textcircled{1}].(\text{true} \wedge (\forall[\textcircled{2}].\text{false})_{\downarrow[\textcircled{1}]}) \xrightarrow{[\textcircled{1}]} \forall[\textcircled{1}].\text{false} \wedge \forall[\textcircled{1} \rightarrow \textcircled{2}].\text{false} \xrightarrow{[\textcircled{1}]} \text{false}$$

In the first step, \mathcal{A} is universal with an isomorphism $\emptyset \rightarrow \emptyset$, which is pulled forward. Together with its (only) existential child, this results in the partial model $[\textcircled{\text{D}}]$ and another universal condition with the meaning: (1) the just created node $\textcircled{\text{D}}$ must not exist, and (2) no additional node $\textcircled{\text{X}}$ may exist either.

This condition includes another isomorphism ($[\textcircled{\text{D}}]$) to be pulled forward. Its child $\mathcal{A}_p = \bigvee_{j \in J} \exists g_j. \mathcal{B}_j = \text{false}$ is an empty disjunction, so the tableau rule for universals adds an empty disjunction (false) as the only descendant. This closes the (only) branch, hence the initial condition \mathcal{A} can be recognized as unsatisfiable. \lrcorner

3.4 Model Finding

We will now discuss the fact that the calculus not only searches for a logical contradiction to show unsatisfiability, but at the same time tries to generate a (possibly infinite) model. We can show that *every* finitely decomposable (i.e., “finite”) model (or a prefix thereof) can be found after finitely many steps in a *fully expanded* tableau, i.e., a tableau where all branches are extended whenever possible, including infinite branches. This is a feature that distinguishes it from other known calculi for first-order logic.

The following lemma shows that an infinite branch always makes progress towards approximating the infinite model.

► **Lemma 20.** *Let \mathcal{A} be a condition and T be a fully expanded tableau for \mathcal{A} . Then, for each branch it holds that it either is finite, or that there always eventually is another non-isomorphism on the branch.*

Proof sketch. We define the size of a condition and show that it decreases if an iso occurs on a path. This means that eventually there will always occur another non-iso on a path. ◀

► **Theorem 21 (Model Finding).** *Let \mathcal{A} be a condition, m a finitely decomposable arrow such that $m \models \mathcal{A}$ and let T be a fully expanded tableau for \mathcal{A} .*

Then, there exists an open and unextendable branch with arrows c_1, \dots, c_n in T , having condition \mathcal{R} in the leaf node, where $m = c_1; \dots; c_n; r$ for some r with $r \models \mathcal{R}$. Furthermore the finite prefix is itself a model for \mathcal{A} (i.e., $[c_1, \dots, c_n] \models \mathcal{A}$).

Note that this finite branch can be found in finite time, assuming a suitable strategy for exploration of the tableau such as breadth-first search or parallel processing.

► **Algorithm 22 (Satisfiability Check).** *Given a condition \mathcal{A} , we define the procedure $\text{SAT}(\mathcal{A})$ that may either produce a model c : $\text{RO}(\mathcal{A}) \rightarrow C$, answer *unsat* or does not terminate.*

- Initialize the tableau with \mathcal{A} in the root node.
- While the tableau still has open branches:
 - Select one of the open branches as the current branch, using an appropriate strategy that extends each open branch eventually.
 - If the leaf is a universal condition without isomorphisms, terminate and return the labels of the current branch as model.
 - Otherwise, extend the branch according to the rules of Definition 10, obeying the fairness constraint.
- If all branches are closed, terminate and answer *unsat*.

This procedure has some similarities to the tableau-based reasoning from [13]. The aspect of model generation was in particular considered in [26]. Overall, we obtain the following result:

► **Theorem 23.** *There is a one-to-one correspondence between satisfiability of a condition (\mathcal{A} unsatisfiable; \mathcal{A} has a finitely decomposable model; \mathcal{A} is satisfiable, but has no finitely decomposable model) and the output of Algorithm 22 ($SAT(\mathcal{A})$) (terminates with unsat, terminates with a model, does not terminate).*

Proof.

- \mathcal{A} unsatisfiable \iff algorithm outputs unsat: (\Rightarrow) Theorem 18, (\Leftarrow) Theorem 17
- \mathcal{A} has a finitely decomposable model \iff algorithm finds finitely decomposable model: (\Rightarrow) Theorem 21, (\Leftarrow) Theorem 16
- \mathcal{A} has only models that are not finitely decomposable \iff algorithm does not terminate: (\Rightarrow) exclusion of other possibilities for non-termination, Lemma 20 for model in the limit (\Leftarrow) Theorem 16 ◀

► **Example 24** (Finding finite models). We now work in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ and use the shorthand notation introduced in Example 19. Let the following condition be given:

$$\begin{aligned} & \forall \emptyset \rightarrow \emptyset. \exists \emptyset \rightarrow \textcircled{1}. \text{true} && \text{(there exists a node } \textcircled{1}\text{)} \\ & \wedge \forall \emptyset \rightarrow \textcircled{1}. \exists \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2}. \text{true} && \text{and every node has an outgoing edge to some other node)} \end{aligned}$$

This condition has finite models, the smallest being the cycle $\textcircled{1} \leftrightarrow \textcircled{2}$. When running Algorithm 22 on this condition, it obtains the model in the following way:

1. The given condition is universal with an iso $\emptyset \rightarrow \emptyset$, which is pulled forward. Together with its (only) existential child, this results in the partial model $\llbracket \textcircled{1} \rrbracket$ and the condition

$$\begin{aligned} & \text{true} \wedge (\forall \llbracket \textcircled{1} \rrbracket. \exists \llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket. \text{true}) \downarrow_{\llbracket \textcircled{1} \rrbracket} \\ & = \forall \llbracket \textcircled{1} \rrbracket. \exists \llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket. \text{true} \wedge \forall \llbracket \textcircled{1} \text{ } \mathbb{A} \rrbracket. \overbrace{(\exists \llbracket \textcircled{1} \text{ } \mathbb{A} \rrbracket \rightarrow \textcircled{B} \rrbracket. \text{true} \vee \exists \llbracket \textcircled{1} \leftarrow \mathbb{A} \rrbracket. \text{true})}^{\mathcal{B}} \end{aligned}$$

meaning: (1) the just created node $\textcircled{1}$ must have an outgoing edge; (2) and every other node A must also have an outgoing edge to either another node or to the existing node.

2. Pull forward iso $\llbracket \textcircled{1} \rrbracket$ and extend the partial model by $\llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket$, resulting in:

$$\begin{aligned} & \text{true} \wedge (\forall \llbracket \textcircled{1} \text{ } \mathbb{A} \rrbracket. \mathcal{B}) \downarrow_{\llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket} = \forall \llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket. \mathcal{B} \downarrow_{\llbracket \textcircled{1} \rightarrow \mathbb{A} \rrbracket} \wedge \forall \llbracket \textcircled{1} \rightarrow \textcircled{2} \text{ } \mathbb{A} \rrbracket. \mathcal{B} \downarrow_{\llbracket \textcircled{1} \rightarrow \textcircled{2} \text{ } \mathbb{A} \rrbracket} \\ & = \forall \llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket. (\exists \llbracket \textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3} \rrbracket. \text{true} \vee \exists \llbracket \textcircled{1} \leftarrow \textcircled{2} \rrbracket. \text{true}) \\ & \quad \wedge \forall \llbracket \textcircled{1} \rightarrow \textcircled{2} \text{ } \mathbb{A} \rrbracket. (\exists \llbracket \textcircled{1} \rightarrow \textcircled{2} \text{ } \mathbb{A} \rrbracket \rightarrow \textcircled{B} \rrbracket. \text{true} \vee \exists \llbracket \textcircled{1} \rightarrow \textcircled{2} \leftarrow \mathbb{A} \rrbracket. \text{true} \vee \exists \llbracket \textcircled{1} \rightarrow \textcircled{2} \text{ } \mathbb{A} \rrbracket. \text{true}) \end{aligned}$$

meaning: (1) the second node has an edge to a third node or to the first one; (2) and every other node A also has an edge to either another node or to one of the existing nodes.

3. Next, we pull forward $\llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket$ and extend the model by $\llbracket \textcircled{1} \leftrightarrow \textcircled{2} \rrbracket$:

$$\text{true} \wedge (\forall \llbracket \textcircled{1} \rightarrow \textcircled{2} \text{ } \mathbb{A} \rrbracket. \dots) \downarrow_{\llbracket \textcircled{1} \leftrightarrow \textcircled{2} \rrbracket} = \forall \llbracket \textcircled{1} \leftrightarrow \textcircled{2} \text{ } \mathbb{A} \rrbracket. \dots$$

4. This condition does not have any children with isos, so it is satisfiable by id. Hence the composition of the partial models so far ($\llbracket \textcircled{1} \leftrightarrow \textcircled{2} \rrbracket$) is a model for the original condition. \dashv

4 Witnesses for infinite models

If there is no finitely decomposable model for a satisfiable condition \mathcal{A} (such as in Example 27 below), then the corresponding infinite branch produces a model in the limit. To detect such models in finite time we introduce an additional use of coinductive techniques based on the tableau calculus previously introduced: We will show that under some circumstances, it is

possible to find some of these infinite models while checking for satisfiability. Naturally, one can not detect all models, since this would lead to a decision procedure for an undecidable problem. We first have to generalize the notion of fairness to finite path fragments:

► **Definition 25.** Let $\mathcal{C}_0 \xrightarrow{b_1} \mathcal{C}_1 \xrightarrow{b_2} \dots \xrightarrow{b_r} \mathcal{C}_r$ be a finite path (also called segment) in a tableau (cf. Definition 30). Such a finite path is called fair if for every child of \mathcal{C}_0 where the morphism is an iso, an indirect successor is pulled forward at some point in the path.

This notion of fairness does not preclude that new isos appear. It only states that all isos present at the beginning are pulled forward at some point.

► **Theorem 26 (Witnesses).** Let \mathcal{C}_0 be an alternating, universal condition. Let a fixed tableau constructed by the rules of Definition 10 be given. Let $\mathcal{C}_0 \xrightarrow{b_1} \mathcal{C}_1 \xrightarrow{b_2} \dots \xrightarrow{b_r} \mathcal{C}_r$ be a fair segment of a branch of the tableau where $r > 0$, and let some arrow m be given such that $\mathcal{C}_0 \cong (\mathcal{C}_r)_{\downarrow m}$ for an iso $\iota: \text{RO}(\mathcal{C}_{r\downarrow m}) \rightarrow \text{RO}(\mathcal{C}_0)$. Then, $[b_1, \dots, b_r, m; \iota]^\omega := [b_1, \dots, b_r, m; \iota, b_1, \dots, b_r, m; \iota, \dots]$ is a model for \mathcal{C}_0 .

Proof sketch. Construct the relation

$$P = \{([b_1, b_2, \dots, b_r, m; \iota, (b_1, \dots, b_r, m; \iota)^\omega], \mathcal{C}_0), \\ ([b_2, \dots, b_r, m; \iota, (b_1, \dots, b_r, m; \iota)^\omega], \mathcal{C}_1), \dots, ([m; \iota, (b_1, \dots, b_r, m; \iota)^\omega], \mathcal{C}_r)\}$$

and show that $P \subseteq s(u(P))$, using an approach similar to that of Theorem 16. Steps b_1, \dots, b_r are handled in the same way as in Theorem 16. For the newly introduced step based on $m; \iota$, the next element of the sequence of representative squares and the successor d_i are chosen from a child of $\mathcal{C}_{r\downarrow m}$ instead of from a successor of \mathcal{D}_i . ◀

► **Example 27 (Finding witnesses).** Consider the following condition:

$$\begin{aligned} \forall \emptyset \rightarrow \emptyset. \exists \emptyset \rightarrow \textcircled{1}. \forall \textcircled{1} \rightarrow \textcircled{\rightarrow} \textcircled{1}. \text{false} & \quad (\text{there is a node } \textcircled{1} \text{ without an incoming edge} \\ \wedge \forall \emptyset \rightarrow \textcircled{\rightarrow} \textcircled{\rightarrow} \textcircled{\oplus}. \text{true} & \quad \text{and every node has an outgoing edge to some other node} \\ \wedge \forall \emptyset \rightarrow \textcircled{\rightarrow} \textcircled{\leftarrow} \textcircled{2}. \text{false} & \quad \text{and no node has two incoming edges}) \end{aligned}$$

This condition has an infinite model, namely an infinite path $(\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3} \rightarrow \dots)$. It does not have any finite model.

In order to display a witness for this model, we need to consider the condition in the category of cospans $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, into which $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ can be embedded. We use the following shorthand notation for cospans: $\llbracket \textcircled{1} \rightarrow \textcircled{2} \rrbracket$ means $\textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{2} \leftarrow \textcircled{1} \rightarrow \textcircled{2}$, i.e., the left object consists of only the light-gray graph elements, the center and right objects consist of the full graph, the left leg is the inclusion and the right leg is always the identity.

If we execute our algorithm on the condition, after one step we obtain a condition \mathcal{C}_0 that is rooted at $\textcircled{1}$, and spells out the requirements of the original condition for node $\textcircled{1}$ and all other nodes separately ($\textcircled{1}$ has a successor, and all other nodes have successors, and so on).

After another step, we obtain \mathcal{C}_1 , which is rooted at $\textcircled{1} \rightarrow \textcircled{2}$, and does the same for node $\textcircled{2}$ separately as well. (These steps are displayed more concretely in the appendix, Table 1.)

Now let $m = \textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{1} \rightarrow \textcircled{2} \leftarrow \textcircled{2}$. Then, we can compute $(\mathcal{C}_1)_{\downarrow m}$, which essentially “forgets” node $\textcircled{1}$ from all subconditions of \mathcal{C}_1 . (Subconditions that contain edges from or to this node disappear entirely, which is a consequence of the way borrowed context diagrams are constructed (via pushout complements).) Then, $(\mathcal{C}_1)_{\downarrow m}$ is similar in structure to \mathcal{C}_0 , and in fact, using a renaming isomorphism $\iota = \textcircled{2} \rightarrow \textcircled{\rightarrow} \textcircled{\leftarrow} \textcircled{1}$, it holds that $(\mathcal{C}_1)_{\downarrow m} \cong \mathcal{C}_0$ wrt. ι . ◻

In general this witness construction will almost never be applicable for simple graph categories, we need to work in other categories, such as cospan categories.

5 Satisfiability in the General Case

Section 3 heavily depends on the fact that all sections are isos, i.e., only isos have a right inverse. However, in the general case we might have conditions of the form $\forall s.\mathcal{A}$ where s has a right inverse r with $s;r = \text{id}$ (note that r need not be unique). This would invalidate our reasoning in the previous sections, since the identity is not necessarily a model of $\forall s.\mathcal{A}$.

► **Example 28.** We work in the category $\mathbf{Graph}_{\text{fin}}$. Consider the following condition $\mathcal{A} = \forall \textcircled{1} \rightarrow \textcircled{1} \textcircled{2} . \exists \textcircled{1} \textcircled{2} \rightarrow \textcircled{1} \rightarrow \textcircled{2} . \text{true}$, defined over a single node $\textcircled{1}$ as root object, which states that the distinguished node has an edge to every other node – including itself, since a non-injective match may merge the two nodes. The first morphism of \mathcal{A} is a section, while the second is injective, but not a section.

The identity on the single node is not a model of \mathcal{A} , but $\textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{1}$ is. However, the condition $\mathcal{A} \wedge \forall \textcircled{1} \rightarrow \textcircled{1} \rightarrow \textcircled{1} . \text{false}$ is unsatisfiable, a fact that would not be detected by Algorithm 22, since neither of the universal quantifiers contains an iso. \lrcorner

Pulling forward isos is still sound even in the general case as the equivalence of Lemma 9 still holds, but it is not sufficient for completeness. Hence we will now adapt the tableau calculus to deal with sections.

► **Lemma 29** (Pulling forward sections). *Let $\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i$ be a universal condition and assume that f_p , $p \in I$, is a section, and r_p is a right inverse of f_p (i.e., $f_p ; r_p = \text{id}$). Furthermore let $\mathcal{A}_{p \downarrow r_p} = \bigvee_{j \in J} \exists h_j . \mathcal{H}_j$ be the result of shifting the p -th child over the right inverse. Then f_p can be pulled forward:*

$$\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \equiv \bigvee_{j \in J} \exists h_j . \left(\mathcal{H}_j \wedge \left(\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \right) \downarrow_{h_j} \right)$$

As for the analogous Lemma 9, pulling forward sections produces an equivalent condition. However, here the child being pulled forward is still included in the children shifted by h_j . Hence the condition will increase in size, unlike for the special case. This is necessary, since f_p might have other inverses which can be used in pulling forward and might lead to new results. This leads to the following adapted rules.

► **Definition 30** (SatCheck rules, general case). *Let \mathcal{A} be an alternating condition. We can construct a tableau for \mathcal{A} by extending it at its leaf nodes as follows:*

$$\bigvee_{i \in I} \exists f_i . \mathcal{A}_i \xrightarrow{f_p} \mathcal{A}_p \quad \left| \quad \begin{array}{l} \text{For every } p \in I: \\ \text{For one } p \in I \text{ such that } f_p \text{ is section, } f_p ; r_p = \text{id}, \\ \text{and } \bigvee_{j \in J} \exists h_j . \mathcal{H}_j = \mathcal{A}_{p \downarrow r_p}: \\ \bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \xrightarrow{\text{id}} \bigvee_{j \in J} \exists h_j . \left(\mathcal{H}_j \wedge \left(\bigwedge_{i \in I} \forall f_i . \mathcal{A}_i \right) \downarrow_{h_j} \right) \end{array} \right.$$

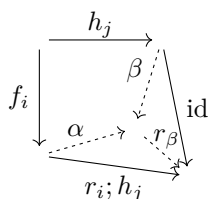
- For existential conditions, for each(!) child condition $\exists f_p . \mathcal{A}_p$, add a new descendant.
- For universal conditions, pick one(!) child condition $\forall f_p . \mathcal{A}_p$ that can be pulled forward in the sense of Lemma 29 and add the result as its (only) descendant.

The rules are similar to those of the specialized case (Definition 10) and as in the special case, we need to define a successor relation on children with sections, that in addition tracks the corresponding right-inverses. The definition of the successor relation is slightly more complex than in the previous case (Definition 11).

► **Definition 31** (Successor relation and tracking right-inverses). We define a relation on pairs of (f_i, r_i) , where f_i is a morphism of a child of a universal condition and r_i is one of its right-inverses. Assume that in the construction of a tableau we have a path

$$\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \xrightarrow{\text{id}} \mathcal{C} \xrightarrow{h_j} \mathcal{H}_j \wedge \left(\bigwedge_{i \in I} \forall f_i. \mathcal{A}_i \right) \downarrow_{h_j} = \mathcal{H}_j \wedge \bigwedge_{i \in I} \bigwedge_{(\alpha, \beta) \in \kappa(f_i, h_j)} \forall \beta. (\mathcal{A}_i) \downarrow_{\alpha}$$

where \mathcal{C} is the existential condition given in Lemma 29. Given (f_i, r_i) (where $f_i; r_i = \text{id}$), we can conclude that the outer square below commutes and hence there exists an inner representative square $(\alpha, \beta) \in \kappa(f_i, h_j)$ and r_β such that the diagram below commutes (in particular β is a section and r_β is a retraction). In this situation, we say that (β, r_β) is a (retraction) successor of (f_i, r_i) .



We extend the definition of fairness to cover sections (not just isomorphisms) and also require that all right-inverses of each section are eventually used in a pull-forward step:

► **Definition 32** (Fairness in the general case). A branch of a tableau is fair if for each universal condition \mathcal{A} on the branch, each child condition $\forall f_i. \mathcal{A}_i$ of \mathcal{A} where f_i is a section, and each right-inverse r_i of f_i , there is $n \in \mathbb{N}_0$ such that in the n -th next step, for some indirect successor (f'_i, r'_i) of (f_i, r_i) it holds that f'_i is pulled forward using the right inverse r'_i . (Every universally-quantified section is eventually pulled forward with every inverse.)

For this definition to be effective, we need to require that every section has only finitely many right-inverses (this is true for e.g. **Graph_{fin}**). Given that property, one way to implement a fairness strategy is to use a queue, to which child conditions (and the corresponding right-inverses) are added. This queue has to be arranged in such a way that for each section/retraction pair a successor is processed eventually.

We now show how to adapt the corresponding results of the previous section (Theorems 16–18) and in particular show that infinite and fair branches are always models, from which we can infer soundness and completeness.

► **Theorem 33** (Fair branches are models (general case)). Let \mathcal{A}_0 be an alternating condition. Let a fixed tableau constructed by the rules of Definition 30 be given. Let $\mathcal{A}_0 \xrightarrow{b_1} \mathcal{A}_1 \xrightarrow{b_2} \mathcal{A}_2 \xrightarrow{b_3} \dots$ be a branch of the tableau that is either unextendable and ends with a universal quantification, or is infinite and fair. For such a branch, we define: $P = \{(\bar{b}, \mathcal{A}_i) \mid i \in \mathbb{N}_0, \bar{c} = [b_{i+1}, b_{i+2}, \dots]\} \subseteq \text{Seq} \times \text{Cond}$. Then, $P \subseteq s(u(P))$. (Every open and unextendable or infinite and fair branch in a tableau of Definition 30 corresponds to a model.)

► **Theorem 34** (Soundness and Completeness).

- If all branches in a tableau are closed, then the condition in the root node is unsatisfiable.
- If a condition \mathcal{A} is unsatisfiable, then in every tableau constructed by obeying the fairness constraint all branches are closed.

Proof. Use the proof strategies of Theorems 17 and 18. Tableaux constructed by the rules of Definition 30 have all properties that are required for the proofs (in particular, universal steps lead to an equivalent condition). Use Theorem 33 to obtain models for open branches. ◀

In the general case, although soundness and completeness still hold, we are no longer able to find all finite models as before. This is intuitively due to the fact that a condition might have a finite model, but what we find is a seemingly infinite model that always has the potential to collapse to the finite model.

On the other hand, the weaker requirements on the category imply that more instantiations are possible, such as to $\mathbf{Graph}_{\text{fin}}$ (graphs and arbitrary morphisms). Here, pushouts can be used for representative squares, which allows for a more efficient shift operation that avoids the blowup of the size of the conditions that is associated with jointly epi squares.

6 Conclusion

We introduced a semi-decision procedure for checking satisfiability of nested conditions at the general categorical level. The correctness of this tableau-based procedure has been established using a novel combination of coinductive (up-to) techniques. In the restricted case we also considered the generation of finite models and witnesses for (some) infinite models. Our procedure thereby generalizes prior work [13, 26, 17] on nested graph conditions that are equivalent to first-order logic [6]. As a result, we can also handle cospan categories over adhesive categories (using borrowed context diagrams for representative squares) and other categories, such as Lawvere theories.

There is a notion of Q-trees [7] reminiscent of the nested conditions studied in this paper, but to our knowledge no generic satisfiability procedures have been derived for Q-trees.

We plan to transfer the technique of counterexample-guided abstraction refinement (CEGAR) [9], a program analysis technique based on abstract interpretation and predicate abstraction, to graph transformation and reactive systems. The computation of weakest preconditions and strongest postconditions for nested conditions is fairly straightforward [1] and satisfiability checks give us the necessary machinery to detect and eliminate spurious counterexamples. One still has to work around undecidability issues and understand whether there is a generalization of Craig interpolation, used to simplify conditions.

One further direction for future work is to understand the mechanism for witness generation in more detail. In particular, since it is known that FOL satisfiability for graphs of bounded treewidth is decidable [2], the question arises whether we can find witnesses for all models of bounded treewidth (or a suitable categorical generalization of this notion).

Another direction is to further explore instantiation with a Lawvere theory [14], where arrows are n -tuples of m -ary terms. In this setting representative squares are closely related to unification. The full version [28] contains some initial results.

Finally we plan to complete development of a tool that implements the satisfiability check and explore the potential for optimizations regarding its runtime.

References

- 1 H.J. Sander Bruggink, Raphaël Cauderlier, Mathias Hülsbusch, and Barbara König. Conditional reactive systems. In *Proc. of FSTTCS '11*, volume 13 of *LIPICs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. URL: http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=3325.
- 2 Bruno Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.
- 3 Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.
- 4 Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *MSCS*, 16(6):1133–1163, 2006.

- 5 Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph grammars: An algebraic approach. In *Proc. 14th IEEE Symp. on Switching and Automata Theory*, pages 167–180, 1973.
- 6 Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1996.
- 7 Peter J. Freyd and Andre Scedrov. *Categories, Allegories*. North-Holland, 1990.
- 8 Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.
- 9 Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In *Proc. of POPL '04*, pages 232–244. ACM, 2004.
- 10 Mathias Hülsbusch and Barbara König. Deriving bisimulation congruences for conditional reactive systems. In *Proc. of FOSSACS '12*, pages 361–375. Springer, 2012. LNCS/ARCoSS 7213.
- 11 Mathias Hülsbusch, Barbara König, Sebastian Küpper, and Lara Stoltenow. Conditional Bisimilarity for Reactive Systems. *Logical Methods in Computer Science*, Volume 18, Issue 1, January 2022. doi:10.46298/lmcs-18(1:6)2022.
- 12 Stephen Lack and Paweł Sobociński. Adhesive and quasiadhesive categories. *RAIRO – Theoretical Informatics and Applications*, 39(3), 2005.
- 13 Leen Lambers and Fernando Orejas. Tableau-based reasoning for graph properties. In *Proc. of ICGT '14*, pages 17–32. Springer, 2014. LNCS 8571.
- 14 William Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963.
- 15 James J. Leifer. *Operational congruences for reactive systems*. PhD thesis, University of Cambridge Computer Laboratory, September 2001.
- 16 James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR 2000*, pages 243–258. Springer, 2000. LNCS 1877.
- 17 Marisa Navarro, Fernando Orejas, Elvira Pino, and Leen Lambers. A navigational logic for reasoning about graph properties. *Journal of Logical and Algebraic Methods in Programming*, 118:100616, 2021.
- 18 Karl-Heinz Pennemann. An algorithm for approximating the satisfiability problem of high-level conditions. In *GT-VC@CONCUR*, volume 213.1 of *Electronic Notes in Theoretical Computer Science*, pages 75–94. Elsevier, 2007.
- 19 Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Universität Oldenburg, May 2009.
- 20 Damien Pous. Complete lattices and up-to techniques. In *Proc. of APLAS '07*, pages 351–366. Springer, 2007. LNCS 4807.
- 21 Damien Pous. *Techniques modulo pour les bisimulations*. PhD thesis, ENS Lyon, February 2008. URL: <https://hal.archives-ouvertes.fr/tel-01441480>.
- 22 Damien Pous and Davide Sangiorgi. Enhancements of the coinductive proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2011.
- 23 Arend Rensink. Representing first-order logic using graphs. In *Proc. of ICGT '04*, pages 319–335. Springer, 2004. LNCS 3256.
- 24 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011.
- 25 Vladimiro Sassone and Paweł Sobociński. Reactive systems over cospans. In *Proc. of LICS '05*, pages 311–320. IEEE, 2005.
- 26 Sven Schneider, Leen Lambers, and Fernando Orejas. Symbolic model generation for graph properties. In *Proc. of FASE '17*, pages 226–243. Springer, 2017. LNCS 10202.
- 27 Paweł Sobociński. *Deriving process congruences from reaction rules*. PhD thesis, Department of Computer Science, University of Aarhus, 2004.

- 28 Lara Stoltenow, Barbara König, Sven Schneider, Andrea Corradini, Leen Lambers, and Fernando Orejas. Coinductive techniques for checking satisfiability of generalized nested conditions, 2024. arXiv:2407.06864. URL: <https://arxiv.org/abs/2407.06864>.
- 29 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

A Additional Material for §2 (Preliminaries)

Graphs and graph morphisms

We will define in more detail which graphs and graph morphisms we are using: in particular, a graph is a tuple $G = (V, E, s, t, \ell)$, where V, E are sets of nodes respectively edges, $s, t: E \rightarrow V$ are the source and target functions and $\ell: V \rightarrow \Lambda$ (where Λ is a set of labels) is the node labelling function. In the examples we will always omit node labels by assuming that there is only a single label.

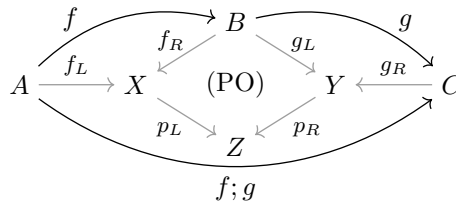
A graph G is finite if both V and E are finite.

Furthermore, given two graphs $G_i = (V_i, E_i, s_i, t_i, \ell_i)$, $i \in \{1, 2\}$, a graph morphism $\varphi: G_1 \rightarrow G_2$ consists of two maps $\varphi_V: V_1 \rightarrow V_2$, $\varphi_E: E_1 \rightarrow E_2$ such that $\varphi_V \circ s_1 = s_2 \circ \varphi_E$, $\varphi_V \circ t_1 = t_2 \circ \varphi_E$ and $\ell_1 = \ell_2 \circ \varphi_V$.

In the examples, the mapping of a morphism is given implicitly by the node identifiers: for instance, $\textcircled{1} \textcircled{2} \rightarrow \textcircled{1} \textcircled{3} \textcircled{2}$ adds the node identified by 3 and adds two edges from the existing nodes identified by 1 and 2.

Cospans and cospan composition

Two cospans $f: A \xrightarrow{f_L} X \xleftarrow{f_R} B$, $g: B \xrightarrow{g_L} Y \xleftarrow{g_R} C$ are composed by taking the pushout (p_L, p_R) of (f_R, g_L) as shown in Figure 1. The result is the cospan $f; g: A \xrightarrow{f_L; p_L} Z \xleftarrow{g_R; p_R} C$, where Z is the pushout object of f_R, g_L . We see an arrow $f: A \rightarrow C$ of **Cospan(D)** as an object B of **D** equipped with two interfaces A, C and corresponding arrows f_L, f_R to relate the interfaces to B , and composition glues the inner objects of two cospans via their common interface.



■ **Figure 1** Composition of cospans f and g is done via pushouts.

In order to make sure that arrow composition in **Cospan(D)** is associative on the nose, we quotient cospans up to isomorphism. In more detail: two cospans $f: A \xrightarrow{f_L} X \xleftarrow{f_R} B$, $g: A \xrightarrow{g_L} Y \xleftarrow{g_R} B$ are equivalent whenever there exists an iso $\iota: X \rightarrow Y$ such that $f_L; \iota = g_L$, $f_R; \iota = g_R$. Then, arrows are equivalence classes of cospans.

Equivalence laws for conditions

We rely on the results given in the following two propositions that were shown in [1]. They were originally stated for satisfaction with single arrows, but it is easy to see they are valid for possibly infinite sequences as well: conditions have finite depth and satisfaction only refers to finite prefixes of the sequence.

► **Proposition 35** (Adjunction). *Let \mathcal{A}, \mathcal{B} be two conditions with root object A , let \mathcal{C}, \mathcal{D} be two conditions with root object B and let $\varphi: A \rightarrow B$. Then it holds that:*

1. $\mathcal{A} \models \mathcal{B}$ implies $\mathcal{A}_{\downarrow\varphi} \models \mathcal{B}_{\downarrow\varphi}$.
2. $\mathcal{C} \models \mathcal{D}$ implies $\mathcal{Q}\varphi.\mathcal{C} \models \mathcal{Q}\varphi.\mathcal{D}$ for $\mathcal{Q} \in \{\exists, \forall\}$.
3. $\exists\varphi.(\mathcal{A}_{\downarrow\varphi}) \models \mathcal{A}$ and for every \mathcal{C} with $\exists\varphi.\mathcal{C} \models \mathcal{A}$ we have that $\mathcal{C} \models \mathcal{A}_{\downarrow\varphi}$.
4. $\mathcal{A} \models \forall\varphi.(\mathcal{A}_{\downarrow\varphi})$ and for every \mathcal{C} with $\mathcal{A} \models \forall\varphi.\mathcal{C}$ we have that $\mathcal{A}_{\downarrow\varphi} \models \mathcal{C}$.

► **Proposition 36** (Laws for conditions). *One easily obtains the following laws for shift and quantification, conjunction and disjunction:*

$$\begin{array}{ll}
 \mathcal{A}_{\downarrow\text{id}} \equiv \mathcal{A} & \mathcal{A}_{\downarrow\varphi; \psi} \equiv (\mathcal{A}_{\downarrow\varphi})_{\downarrow\psi} \\
 \forall\text{id}.\mathcal{A} \equiv \mathcal{A} & \forall(\varphi; \psi).\mathcal{A} \equiv \forall\varphi.\forall\psi.\mathcal{A} \\
 \exists\text{id}.\mathcal{A} \equiv \mathcal{A} & \exists(\varphi; \psi).\mathcal{A} \equiv \exists\varphi.\exists\psi.\mathcal{A} \\
 (\mathcal{A} \wedge \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \wedge \mathcal{B}_{\downarrow\varphi} & (\mathcal{A} \vee \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \vee \mathcal{B}_{\downarrow\varphi} \\
 \forall\varphi.(\mathcal{A} \wedge \mathcal{B}) \equiv \forall\varphi.\mathcal{A} \wedge \forall\varphi.\mathcal{B} & \exists\varphi.(\mathcal{A} \vee \mathcal{B}) \equiv \exists\varphi.\mathcal{A} \vee \exists\varphi.\mathcal{B}
 \end{array}$$

Borrowed context diagrams

For cospan categories over adhesive categories (such as $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}})$), borrowed context diagrams – initially introduced as an extension of DPO rewriting [4] – can be used as representative squares. Before we can introduce such diagrams, we first need the notion of jointly epi.

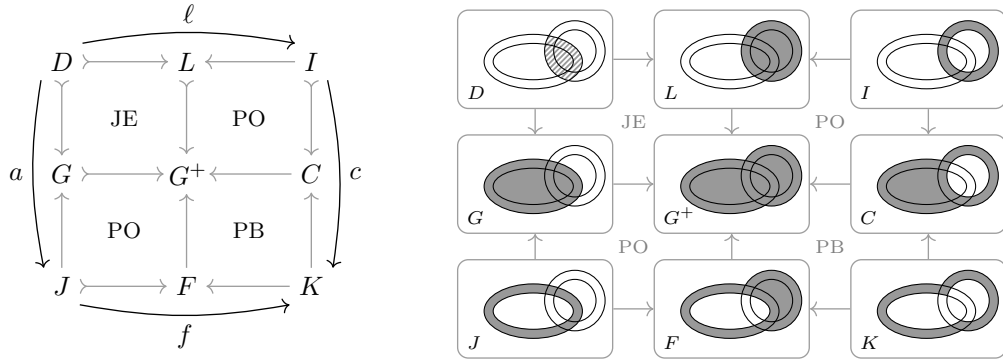
► **Definition 37** (Jointly epi). *A pair of arrows $f: B \rightarrow D$, $g: C \rightarrow D$ is jointly epi (JE) if for each pair of arrows $d_1, d_2: D \rightarrow E$ the following holds: if $f; d_1 = f; d_2$ and $g; d_1 = g; d_2$, then $d_1 = d_2$.*

In $\mathbf{Graph}_{\text{fin}}$ jointly epi equals jointly surjective, meaning that each node or edge of D is required to have a preimage under f or g or both (it contains only images of B or C).

This criterion is similar to, but weaker than a pushout: For jointly epi graph morphisms $d_1: B \rightarrow D$, $d_2: C \rightarrow D$, there are no restrictions on which elements of B, C can be merged in D . However, in a pushout constructed from morphisms $a_1: A \rightarrow B$, $a_2: A \rightarrow C$, elements in D can (and must) only be merged if they have a common preimage in A . (Hence every pushout generates a pair of jointly epi arrows, but not vice versa.)

► **Definition 38** (Borrowed context diagram [10]). *A commuting diagram in the category $\mathbf{ILC}(\mathbf{C})$, where \mathbf{C} is adhesive, is a borrowed context diagram whenever it has the form of the diagram shown in Figure 2a, and the four squares in the base category \mathbf{C} are pushout (PO), pullback (PB) or jointly epi (JE) as indicated. Arrows depicted as \rightsquigarrow are mono. In particular $L \rightsquigarrow G^+$, $G \rightsquigarrow G^+$ must be jointly epi.*

Figure 2b shows a more concrete version of Figure 2a, where graphs and their overlaps are depicted by Venn diagrams (assuming that all morphisms are injective). Because of the two pushout squares, this diagram can be interpreted as composition of cospans $a; f = \ell; c = D \rightarrow G^+ \leftarrow K$ with extra conditions on the top left and the bottom right square. The top left square fixes an overlap G^+ of L and G , while D is contained in the intersection of L and G (shown as a hatched area). Being jointly epi ensures that it really is an overlap and does not contain unrelated elements. The top right pushout corresponds to the left pushout of a DPO rewriting diagram. It contains a total match of L in G^+ . Then, the bottom left pushout gives us the minimal borrowed context F such that applying the rule becomes possible. The top left and the bottom left squares together ensure that the contexts to be considered are not larger than necessary. The bottom right pullback ensures that the interface K is as large as possible.



(a) Structure of a borrowed context diagram. The inner, lighter arrows are morphisms of the base category \mathbf{C} , while the outer arrows are morphisms of $\mathbf{ILC}(\mathbf{C})$.

(b) Borrowed context diagrams represented as Venn diagrams. The outer circles represent graphs L, G , and the area between the inner and outer circles represents their interfaces I, J .

■ **Figure 2** Borrowed context diagrams.

For more concrete examples of borrowed context diagrams, we refer to [4, 11].

For cospan categories over adhesive categories, borrowed context diagrams form a representative class of squares [1]. Furthermore, for some categories (such as $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$), there are – up to isomorphism – only finitely many jointly epi squares for a given span of monos and hence only finitely many borrowed context diagrams given a, ℓ (since pushout complements along monos in adhesive categories are unique up to isomorphism).

Whenever the two cospans ℓ, a are in $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, it is easy to see that f, c are in $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$, i.e., they consist only of monos, i.e., injective morphisms.

Note also that representative squares in $\mathbf{Graph}_{\text{fin}}^{\text{inj}}$ are simply jointly epi squares and they can be straightforwardly extended to squares of $\mathbf{ILC}(\mathbf{Graph}_{\text{fin}}^{\text{inj}})$.

Visualization of shifts

Given a condition \mathcal{A} and an arrow $c: A = \text{RO}(\mathcal{A}) \rightarrow B$, we will visualize shifts in diagrams as follows:

$$\begin{array}{c} \mathcal{A} \qquad \mathcal{A}_{\downarrow c} \\ \begin{array}{ccc} \triangle & & \triangle \\ A & \xrightarrow{c} & B \xrightarrow{d} X \end{array} \end{array}$$

Remember that for an arrow $d: B \rightarrow X$ it holds that $d \models \mathcal{A}_{\downarrow c} \iff c; d \models \mathcal{A}$.

B Additional Material for §4 (Witnesses for infinite models)

■ **Table 1** Steps for the condition of Example 27, showing that a repeating infinite model exists.

\mathcal{C}_0	\mathcal{C}_1	$(\mathcal{C}_1)_{\downarrow m}$
$\forall [\textcircled{C} \rightarrow \textcircled{D}]. \text{false}$	$\forall [\textcircled{C} \rightarrow \textcircled{1} \rightarrow \textcircled{2}]. \text{false}$	
$\wedge \forall [\textcircled{1} \oplus \textcircled{2}]. (\exists [\textcircled{1} \oplus \textcircled{3} \rightarrow \textcircled{O}]. \text{true} \vee \exists [\textcircled{1} \oplus \textcircled{3}]. \text{true})$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2}]. \text{false}$ $\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \oplus]. (\exists [\textcircled{1} \rightarrow \textcircled{2} \oplus \rightarrow \textcircled{O}]. \text{true} \vee \exists [\textcircled{1} \rightarrow \textcircled{2} \oplus]. \text{true})$	$\forall [\textcircled{2} \oplus]. (\exists [\textcircled{2} \oplus \rightarrow \textcircled{O}]. \text{true} \vee \exists [\textcircled{2} \oplus]. \text{true})$
$\wedge \forall [\textcircled{1}]. \exists [\textcircled{1} \rightarrow \textcircled{2}]. \text{true}$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2}]. (\exists [\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3}]. \text{true} \vee \exists [\textcircled{1} \rightarrow \textcircled{2}]. \text{true})$	$\wedge \forall [\textcircled{2}]. \exists [\textcircled{2} \rightarrow \textcircled{3}]. \text{true}$
$\wedge \forall [\textcircled{1} \rightarrow \textcircled{A} \rightarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{A} \rightarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{2} \rightarrow \textcircled{A} \rightarrow \textcircled{B}]. \text{false}$
$\wedge \forall [\textcircled{1} \rightarrow \textcircled{A} \rightarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{A} \rightarrow \textcircled{B}]. \text{false}$	$\wedge \forall [\textcircled{2} \rightarrow \textcircled{A} \rightarrow \textcircled{B}]. \text{false}$
$\wedge \dots$	$\wedge \forall [\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{B}]. \text{false}$ $\wedge \dots$	$\wedge \forall [\textcircled{2} \rightarrow \textcircled{B}]. \text{false}$ $\wedge \dots$

A PSPACE Algorithm for Almost-Sure Rabin Objectives in Multi-Environment MDPs

Marnix Suilen¹  

Radboud University, Nijmegen, The Netherlands

Marck van der Vegt¹  

Radboud University, Nijmegen, The Netherlands

Sebastian Junges  

Radboud University, Nijmegen, The Netherlands

Abstract

Markov Decision Processes (MDPs) model systems with uncertain transition dynamics. Multiple-environment MDPs (MEMDPs) extend MDPs. They intuitively reflect finite sets of MDPs that share the same state and action spaces but differ in the transition dynamics. The key objective in MEMDPs is to find a single strategy that satisfies a given objective in every associated MDP. The main result of this paper is PSPACE-completeness for almost-sure Rabin objectives in MEMDPs. This result clarifies the complexity landscape for MEMDPs and contrasts with results for the more general class of partially observable MDPs (POMDPs), where almost-sure reachability is already EXP-complete, and almost-sure Rabin objectives are undecidable.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Markov Decision Processes, partial observability, linear-time Objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.40

Related Version *Full Version:* <https://arxiv.org/abs/2407.07006> [38]

Funding *Marnix Suilen:* NWO OCENW.KLEIN.187

Sebastian Junges: NWO Veni Grant ProMiSe (222.147)

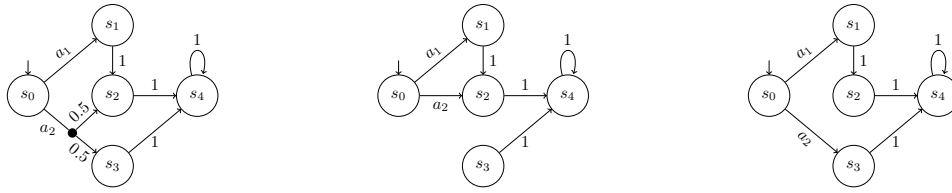
1 Introduction

Markov decision processes (MDPs) are the ubiquitous model for decision-making under uncertainty [34]. An elementary question in MDPs concerns the existence of strategies that satisfy qualitative temporal properties, such as *is there a strategy such that the probability of reaching a set of target states is one?* Qualitative properties in MDPs have long been considered as pre-processing for probabilistic model checking of quantitative properties [6, 24]. Recently, however, qualitative properties have received interest in the context of shielding [1, 29], *i.e.*, the application of model-based reasoning to ensure safety in reinforcement learning [25, 39].

An often prohibitive assumption in using MDPs is that the strategy can depend on the precise state. To follow such a strategy, one must precisely observe the state of the system, *i.e.*, of an agent and its environment. The more general partially observable MDPs (POMDPs) [30] do not make this assumption. In POMDPs, a strategy cannot depend on the precise states of the system but only on the (sequence of) observed labels of visited states. As a consequence, and in contrast to MDPs, winning strategies may require memory. Indeed, the existence of strategies that satisfy qualitative objectives on MDPs is efficiently decidable in polynomial time using standard graph-algorithms [15, 14]. In contrast, in POMDPs, deciding almost-sure reachability is already EXPTIME-complete [4, 14], and the existence of strategies for a more general class of almost-sure Rabin objectives is undecidable [4].

¹ Both authors contributed equally





■ **Figure 1** A MEMDP with three environments.

Multi-environment MDPs (MEMDPs) [36] model a finite set of MDPs, called *environments*, that share the state space but whose transition relation may be arbitrarily different. For any given objective, the key decision problem asks to find a single strategy that satisfies the objective in every associated MDP. MEMDPs are particularly suitable to model settings where one searches for a winning strategy that is robust to perturbations or random initialization problems. Examples of MEMDPs range from code-breaking games such as Mastermind, card games such as Free-cell, or Minesweeper, to more serious applications in robotics [17], *e.g.*, high-level planning where an artifact with unknown location must be recovered.

MEMDPs are special POMDPs [11]: An agent can observe the current state but not the transition relation determining the outcomes of its actions. This type of partial observation has an important effect: When an agent observes the next state, it may rule out that a certain MDP describes the true system. Thus, the set of MDPs that may describe the system is monotonically decreasing [11]. We call this property as *monotonic information gain*.

We illustrate MEMDPs in Fig. 1. This MEMDP consists of three environments. When playing action a_1 in state s_0 , we end up in state s_1 in every environment. If we play action a_2 and observe that we end up in state s_2 , we can infer that we have to either be in the left or the middle environment, while observing s_3 after a_2 rules out the middle environment. Such information cannot be lost in a MEMDP, hence *monotonic* information gain.

The most relevant results for qualitative properties on MEMDPs are by Raskin and Sankur [36], and by Van der Vegt et al. [40]. The former paper focuses on the case with only *two environments*, which we refer to as 2-MEMDPs (and more generally, k -MEMDPs). It shows, among others, that almost-sure parity can be decided in polynomial time. In 2-MEMDPs, the memory for a winning strategy is polynomial in the size of the MEMDP, while for arbitrary environments, winning strategies for almost-sure reachability may be exponential [40]. However, despite the need for exponential strategies, almost-sure *reachability* in MEMDPs is decidable in PSPACE via a recursive algorithm that exploits the aforementioned monotonic information gain [40].

■ **Table 1** Known complexity (completeness) for MEMDPs, new results are in boldface. NL and EXP denote the classes NLOGSPACE and EXPTIME, and UD denotes UNDECIDABLE.

Semantics Model	Almost-sure					Possible		
	MDP	2-MEMDP	k -MEMDP	MEMDP	POMDP	MDP	MEMDP	POMDP
Reachability	P [14]	P [36]	P Cor. 49	PSPACE [40]	EXP [4, 14]	NL [14]	NL Thm. 6	NL [14]
Safety	P [14]	P [36]	P Cor. 49	PSPACE Thm. 41	EXP [8, 35]	P [14]	P Thm. 6	EXP [14]
Büchi	P [14]	P [36]	P Cor. 49	PSPACE Thm. 41	EXP [4, 14]	P [14]	P Thm. 6	UD [4]
Co-Büchi	P [14]	P [36]	P Cor. 49	PSPACE Thm. 41	UD [4]	P [14]	P Thm. 6	EXP [14]
Parity	P [14]	P [36]	P Cor. 49	PSPACE Thm. 41	UD [4]	P [14]	P Thm. 6	UD [4]
Rabin	P [13]	P Cor. 49	P Cor. 49	PSPACE Thm. 41	UD [4]	P [6]	P Thm. 6	UD [4]

The main result of this paper is a landscape of qualitative Rabin objectives and their subclasses in MEMDPs, see Tbl. 1. The key novelty is a PSPACE algorithm to decide the existence of strategies in MEMDPs that satisfy an almost-sure Rabin objective. The algorithm relies on two key ingredients: First, as shown in Sect. 4, for almost-sure Rabin objectives, a particular type of finite-memory strategies (with memory exponential in the number of environments) is sufficient, in contrast to POMDPs. Second, towards an algorithm, we observe that a traditional, per Rabin-pair, approach for Rabin objectives does not generalize to MEMDPs (Sect. 6.1). It does, however, generalize to what we call belief-local MEMDPs, in which one, intuitively, cannot gain any information (Sect. 6.3). Exploiting the monotonic information gain of (general) MEMDPs, we construct a recursive algorithm with polynomial stack size, inspired by [40], that solves Rabin objectives in belief-local MEMDPs (Sect. 6.4). Finally, we establish PSPACE-hardness for almost-sure safety and clarify that for possible objectives, MEMDPs can be solved as efficiently as MDPs. Proofs are in the appendix of the full version of this paper [38].

Related Work

Besides almost-sure objectives, Raskin and Sankur [36] also study *limit-sure* objectives for MEMDPs of two environments. Where almost-sure objectives require that the satisfaction probability of the objective equals one, an objective is satisfied *limit-surely* whenever for any $\epsilon > 0$, there is a strategy under which the objective is satisfied with probability at least $1 - \epsilon$. For 2-MEMDPs, limit-sure parity objectives are decidable in P [36].

Closely related to the study of almost-sure objectives in MEMDPs and POMDPs is the *value 1 problem* for probabilistic automata (PA). A PA can be seen as a POMDP where all states have the same observation and are thus indistinguishable. The value 1 problem is to decide whether the supremum of the acceptance probability over all words equals one. This problem is undecidable for general PA [26], but recent works have studied several subclasses of PA for which the value 1 problem is decidable. Most notably, *#-acyclic* PA [26], *structurally simple* and *simple* PA [16], and *leaktight* PA [23]. Leaktight PA are the most general of these subclasses [20]. They contain the others, and the value 1 problem is PSPACE-complete [22].

The interpretation of MEMDPs as a special POMDP is successfully used in the quantitative setting, where the goal is to find a strategy that maximizes the probability of reaching a target. Finding a strategy that maximizes the finite-horizon expected reward in MEMDPs is PSPACE-complete [9], as is also the case for the same problem in more general POMDPs [33].

Besides a special class of POMDP, MEMDPs are also a class of *robust* MDP with discrete uncertainty sets [32, 27]. In the robotics and AI communities MEMDPs are studied in that context, primarily for quantitative objectives such as maximizing discounted reward or regret minimization [37]. *Parametric* MDPs (pMDPs) are another formalism for defining MDPs with a range of transition functions [28]. Where we seek a single strategy that is winning for all environments, parameter synthesis is often about finding a single parameter instantiation (or: environment) such that all strategies are winning [18, 2]. That is, the quantifiers are reversed. A notable exception is work on quantitative properties in pMDPs by Arming et al. [3], which interprets a parametric MDP as a MEMDP and solves it as a POMDP. With the (altered) quantifier order in pMDPs, memoryless deterministic strategies are sufficient: The complexity of finding a parameter instantiation such that under all (memoryless deterministic) strategies a quantitative reachability objective is satisfied is in NP when the number of parameters is fixed and ETR-complete in the general case [41]. Determining whether a memoryless deterministic policy is robust is both ETR and co-ETR-hard [41].

Concurrent parameterized games have a similar type of partial observability as MEMDPs but lack a probabilistic transition function. The complexity of deciding reachability objectives in concurrent parameterized games is PSPACE-complete [7], equal to that of almost-sure reachability objectives in MEMDPs [40].

2 Background and Notation

Let \mathbb{N} denote the natural numbers. For a set X , the powerset of X is denoted by $\mathcal{P}(X)$, and the disjoint union of two sets X, Y is denoted $X \sqcup Y$. A discrete probability distribution over a finite set X is a function $\mu: X \rightarrow [0, 1]$ with $\sum_{x \in X} \mu(x) = 1$, the set of all discrete probability distributions over X is $\text{Dist}(X)$. The support of a distribution $\mu \in \text{Dist}(X)$ is the set of elements x with $\mu(x) > 0$ and is denoted by $\text{Supp}(\mu)$. We denote the uniform distribution over X by $\text{unif}(X)$ and the Dirac distribution with probability 1 on x by $\text{dirac}(x)$.

2.1 Markov Decision Processes

We briefly define standard (discrete-time) Markov decision processes and Markov chains.

► **Definition 1** (MDP). A Markov decision process (MDP) is a tuple $M := \langle S, A, \iota, p \rangle$ where S is a finite set of states and $\iota \in S$ is the initial state, A is the finite set of actions, and $p: S \times A \rightarrow \text{Dist}(S)$ is the partial probabilistic transition function. By $A(s)$, we denote the set of enabled actions for s , which are the actions for which $p(s, a)$ is defined.

For readability, we write $p(s, a, s')$ for $p(s, a)(s')$. A *path* in an MDP is a sequence of successive states and actions, $\pi = s_0 a_0 s_1 a_1 \dots \in (SA)^* S$, such that $s_0 = \iota$, $a_i \in A(s_i)$, and $p(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 0$, and we write $\bar{\pi}$ for only the sequence of states in π . The probability of following a path π in an MDP with transition function p is defined as $p(\pi) = p(s_0 a_0 \dots) = \prod_{i=0}^{\infty} p(s_i, a_i, s_{i+1})$. The set of all (finite) paths on an MDP M is $\text{Path}(M)$ (resp. $\text{Path}_{\text{fin}}(M)$). Whenever clear from the context, we omit the MDP M from these notations. We write $\text{first}(\pi)$ and $\text{last}(\pi)$ for the first and last state in a finite path, respectively, and the concatenation of two paths π_1, π_2 is written as $\pi_1 \cdot \pi_2$. The set of *reachable states* from $S' \subseteq S$ is $\text{Reachable}(S') := \{s' \in S \mid \exists \pi \in \text{Path}_{\text{fin}}: \text{first}(\pi) \in S', \text{last}(\pi) = s'\}$. A state $s \in S$ is a *sink state* if $\text{Reachable}(\{s\}) = \{s\}$. An MDP is *acyclic* if each state is a sink state or not reachable from its successor states. The *underlying graph* of an MDP is a tuple $\langle V, E \rangle$ with vertices $V := \{v_s \mid s \in S\}$ and edges $E := \{ \langle v_s, v_{s'} \rangle \mid \forall s, s' \in S: \exists a \in A: p(s, a, s') > 0 \}$.

A *sub-MDP* of an MDP $M = \langle S, A, \iota, p \rangle$ is a tuple $\langle S', A', \iota', p' \rangle$ with states $\emptyset \neq S' \subseteq S$, actions $\emptyset \neq A' \subseteq A$, initial state $\iota' \in S'$, and a transition function p' such that $\forall s \in S: \emptyset \neq A'(s) \subseteq A(s)$ and $\forall s, s' \in S', a \in A'(s): \text{Supp}(p(s, a)) \subseteq S'$ and $p'(s, a, s') := p(s, a, s')$. An *end-component* of an MDP M is a sub-MDP where $\text{Reachable}(S') = S'$. Sub-MDPs and end-components are standard notions; for details, cf. [19, 6, 36].

A *Markov chain* is an MDP where there is only one action available at every state: $\forall s \in S: |A(s)| = 1$. We write an MC as a tuple $C := \langle S, \iota, p \rangle$ where S is a set of states, $\iota \in S$ is the initial state, and $p: S \rightarrow \text{Dist}(S)$ is the transition function. Paths in MCs are sequences of successive states, and their underlying graph is analogously defined as for MDPs. A subset $T \subseteq S$ is *strongly connected* if for each pair of states $\langle s, s' \rangle \in T$ there exists a finite path π with $\text{first}(\pi) = s$ and $\text{last}(\pi) = s'$. A *strongly connected component* (SCC) is a strongly connected set of states T such that no proper superset of T is strongly connected. A *bottom SCC* (BSCC) is an SCC S' where no state $s \in S \setminus S'$ is reachable.

2.2 Strategies and Objectives

We now formally define strategies and their objectives. Strategies resolve the action choices in MDPs. A strategy is a measurable function $\sigma: \text{Path}_{\text{fin}} \rightarrow \text{Dist}(A)$ such that for all finite paths $\pi \in \text{Path}_{\text{fin}}$ we have $\text{Supp}(\sigma(\text{last}(\pi))) \subseteq A(\text{last}(\pi))$. A strategy is deterministic if it maps only to Dirac distributions, and it is memoryless if the action (distribution) only depends on the last state of every path. We write Σ for the set of all strategies.

A strategy σ applied to an MDP M induces an infinite-state MC $M[\sigma] = \langle S^*, \iota, p_\sigma \rangle$ such that for any path $\pi: p_\sigma(\bar{\pi}, \bar{\pi} \cdot s') = \sum_{a \in A} \sigma(\pi)(a) \cdot p(\text{last}(\pi), a, s')$. This MC has a probability space with a unique probability measure $\mathbb{P}_{M[\sigma]}$ via the cylinder construction [6, 21].

A strategy is a *finite-memory* strategy if it can be encoded by a stochastic Moore machine, also known as a finite-state controller (FSC) [31]. An FSC is a tuple $\mathcal{F} = \langle N, n_\iota, \alpha, \eta \rangle$, where N is a finite set of *memory nodes*, the *initial node* $n_\iota \in N$, $\alpha: S \times N \rightarrow \text{Dist}(A)$ is the *action mapping*, and $\eta: N \times A \times S \rightarrow \text{Dist}(N)$ is the *memory update function*. The induced MC $M[\mathcal{F}]$ of an MDP M and finite-memory strategy represented by an FSC \mathcal{F} is finite and defined by the following product construction: $M[\mathcal{F}] = \langle S \times N, \langle \iota, n_\iota \rangle, p_{\mathcal{F}} \rangle$, where $p_{\mathcal{F}}(\langle s, n \rangle, \langle s', n' \rangle) = \sum_{a \in A} \alpha(s, n, a) \cdot p(s, a, s') \cdot \eta(n, a, s', n')$. A strategy is *memoryless* if its FSC representation has a single memory node, *i.e.*, $|N| = 1$.

We consider both *almost-sure* and *possible* objectives for MDPs and MCs with state space S . An *objective* Φ is a measurable subset of $P \subseteq S^\omega$. An MC C is almost-surely (or possibly) *winning* for an objective Φ iff $\mathbb{P}_C(\text{Path}(C) \cap \Phi) = 1$ (or $\mathbb{P}_C(\text{Path}(C) \cap \Phi) > 0$). A state s is winning whenever the MC with its initial state replaced by s is winning. We write $C \models \Phi$ and $s \models^C \Phi$ to denote that MC C and state s are winning for Φ .

► **Definition 2 (Winning).** *An MDP M is winning for Φ if there exists a strategy $\sigma \in \Sigma$ such that the induced MC $M[\sigma]$ is winning for Φ , and the strategy σ is then also called winning.*

Like above, we denote winning in MDPs with $M \models \Phi$ or $s \models^M \Phi$, respectively. Sometimes, we explicitly add the winning strategy and write $M[\sigma] \models \Phi$ and $s \models^{M[\sigma]} \Phi$ for the MDP winning Φ under σ from its initial state or some other state s , respectively.

► **Definition 3 (Winning Region).** *We call the set of states of an MDP (or MC) that are winning objective Φ the winning region, denoted as $\text{Win}_M(\Phi) = \{s \in S \mid s \models^M \Phi\}$.*

We define Rabin objectives. Let $C = \langle S, \iota, p \rangle$ be a MC with associated probability measure \mathbb{P}_C , $\pi \in \text{Path}(C)$ a path, and $\text{Inf}(\pi) \subseteq S$ the set of states reached infinitely often along π .

► **Definition 4 (Rabin objective).** *A Rabin objective is a set of Rabin pairs: $\Phi = \{\langle \mathfrak{B}_i, \mathfrak{C}_i \rangle \mid 1 \leq i \leq k. \mathfrak{C}_i \subseteq \mathfrak{B}_i \subseteq S\}$. A path $\pi \in \text{Path}(C)$ wins Φ if there is a Rabin pair $\langle \mathfrak{B}_i, \mathfrak{C}_i \rangle$ in Φ where the path leaves the states in \mathfrak{B}_i only finitely many times, and states in \mathfrak{C}_i are visited infinitely often. The MC C wins a Rabin objective almost-surely (or possibly) if $\mathbb{P}_C(\pi \in \text{Path}(C) \mid \exists \langle \mathfrak{B}_i, \mathfrak{C}_i \rangle \in \Phi: \text{Inf}(\pi) \subseteq \mathfrak{B}_i \wedge \text{Inf}(\pi) \cap \mathfrak{C}_i \neq \emptyset) = 1$ (or possibly when > 0).*

For MDPs, almost-sure and possible Rabin objectives can be solved in polynomial time [6, Thm. 10.127], and the strategies are memoryless deterministic [13, Thm. 4]. Other objectives, specifically reachability ($\diamond T$), safety ($\square T$), Büchi ($\square \diamond T$), co-Büchi ($\diamond \square T$), and parity are included in Rabin objectives [13] for a set $T \subseteq S$. App. A contains formal definitions.

3 Multi-Environment MDPs and the Problem Statement

Next, we introduce the multi-environment versions of MDPs and MCs. Intuitively, these can be seen as finite sets of MDPs and MCs that share the same states and actions.

► **Definition 5** (MEMDP). A multi-environment MDP is a tuple $\mathcal{M} = \langle S, A, \iota, \{p_i\}_{i \in I} \rangle$ with S, A, ι as for MDPs, and $\{p_i\}_{i \in I}$ is a finite set of transition functions, where I are the environment indices. We also write $\mathcal{M} = \{M_i\}_{i \in I}$ as a set of MDPs, where $M_i = \langle S, A, \iota, p_i \rangle$.

For a MEMDP \mathcal{M} and a set $I' \subseteq I$, we define the restriction to I' as the MEMDP $\mathcal{M}_{\downarrow I'} = \langle S, A, \iota, \{p_i\}_{i \in I'} \rangle$. To change the initial state of \mathcal{M} , we define $\mathcal{M}' = \langle S, A, \iota', \{p_i\}_{i \in I} \rangle$.

A multi-environment MC (MEMC) is a MEMDP with $\forall s \in S: |A(s)| = 1$. A MEMC is a tuple $\mathcal{C} = \langle S, \iota, \{p_i\}_{i \in I} \rangle$ or equivalently a set of MCs $\mathcal{C} = \{C_i\}_{i \in I}$. A BSCC in a MEMC is a set of states $S' \subseteq S$ such that S' forms a BSCC in every MC $C_i \in \mathcal{C}$. The underlying graph of a MEMDP or MEMC is the disjoint union of the graphs of the environments.

Similarly to Def. 2 for MDPs, a strategy σ for a MEMDP \mathcal{M} is winning for objective Φ if and only if the induced MEMC $\mathcal{M}[\sigma] = \{\mathcal{M}[\sigma]_i\}_{i \in I}$ is winning in all environments: $\forall i \in I: \mathcal{M}[\sigma]_i$ is winning for Φ . Winning regions, Def. 3, extend similarly to MEMDPs: $Win_{\mathcal{M}}(\Phi) = \{s \in S \mid s \models^{\mathcal{M}} \Phi\}$.

The central decision problem in this paper is:

Given a MEMDP \mathcal{M} and a Rabin objective Φ , is there a winning strategy for Φ in \mathcal{M} .

We assume MEMDPs are encoded as an explicit list of MDPs and each MDP is given by the explicit transition function. The value of the probabilities are not relevant.

We first consider possible semantics in MEMDPs, completing Tbl. 1. For POMDPs, co-Büchi objectives are known to be undecidable [4]. We show that for various objectives, deciding them in MEMDPs is equally hard as in their MDP counterparts.

► **Theorem 6.** *Deciding possible reachability objectives for MEMDPs is in NL. Deciding possible safety, Büchi, co-Büchi, parity and Rabin objectives for MEMDPs is in P.*

Using results on MDPs from [14], these upper bounds are tight. The main observation for membership is that a MEMDP is winning possibly objectives iff each MDP is possibly winning, due to a randomization over the individual winning strategies. We can then construct algorithms that solve each environment sequentially to answer the query on MEMDPs.

From here on, we focus exclusively on the almost-sure objectives.

► **Theorem 7.** *Almost-sure reach, safety, (co-)Büchi, and Rabin objectives for MEMDPs are PSPACE-hard.*

This theorem follows from [40], which shows that almost-sure reachability is PSPACE-complete. PSPACE-hardness of almost-sure safety can be established by minor modifications to the proof: In particular, the PSPACE-hardness proof for reachability operates on acyclic MEMDPs, where we may reverse the target and non-target states to change the objective from almost-sure reachability to safety. PSPACE-hardness of almost-sure (co-)Büchi, parity, and Rabin objectives follows via reduction from almost-sure reachability.

4 Belief-Based Strategies are Sufficient

In this section, we fix a MEMDP $\mathcal{M} = \langle S, A, \iota, \{p_i\}_{i \in I} \rangle$ with an almost-sure Rabin objective Φ , and constructively show a more refined version of the following statement.

► **Corollary 8.** *For a MEMDP \mathcal{M} and an almost-sure Rabin objective Φ , if there exists a winning strategy for Φ , there also exists a winning finite-memory strategy σ^* such that the finite-state controller (FSC) for σ^* is exponential (only) in the number of environments.*

This corollary to Thm. 15 below immediately gives rise to EXP algorithms for the decision problem that simply iterate over all strategies. The particular shape of the strategy, a notion that we call *belief-based*, will be essential later to establish PSPACE algorithms.

4.1 Beliefs in MEMDPs

It is helpful to consider the strategy as a model for a decision-making agent. Then, in MEMDPs, the agent observes in which state $s \in S$ it currently is but does not have access to the environment $i \in I$. The hiding of environments gives rise to the notion of a *belief-distribution* in MEMDPs, akin to beliefs in partially observable MDPs [30]. A belief distribution in a MEMDP is a probability distribution over environments $\mu \in \text{Dist}(I)$ that assigns a probability to how likely the agent is operating in each environment. As we show below, we only need to consider the *belief-support*, *i.e.*, a subset of environments that keeps track of whether it is *possible* that the agent operates in those environments. From now on, we shall simply write *belief* instead of *belief-support*.

► **Definition 9** (Belief, belief update). *Given a finite path π , we define its belief as its last state together with the set of environments for which this path has positive probability: $\text{Belief}(\pi) = \langle \text{last}(\pi), \{i \in I \mid p_i(\pi) > 0\} \rangle$. For path $\pi \cdot as'$, the belief can be characterized recursively by the belief update function $\text{BU}: S \times \mathcal{P}(I) \times A \times S \rightarrow S \times \mathcal{P}(I)$. Let $\langle s, J \rangle = \text{Belief}(\pi)$, then: $\langle s', J' \rangle = \text{BU}(\langle s, J \rangle, a, s') := \text{Belief}(\pi \cdot as')$, where $J' = \{j \in J \mid p_j(s, a, s') > 0\}$. We also liberally write $\text{BU}(\langle s, J \rangle, a)$ for the set of beliefs $\langle s', J' \rangle$ that are possible from $\langle s, J \rangle$ via action a , and define the two projection functions: $\text{St}(\langle s, J \rangle) = s$ and $\text{Env}(\langle s, J \rangle) = J$.*

Key to MEMDPs is the notion of *revealing transitions* [36]. A revealing transition is a tuple $\langle s, a, s' \rangle$ such that there exist two environments $i, i' \in I, i \neq i'$ with $p_i(s, a, s') > 0$ and $p_{i'}(s, a, s') = 0$. Intuitively, a transition is revealing whenever observing this transition reduces the belief over environments the agent is currently in, since we observed a transition that is not possible in one or more environments. From this notion of revealing transitions immediately follows the property of monotonic information gain in MEMDPs.

► **Corollary 10.** *Let $\pi \cdot as'$ be a finite path. Then: (1) $\text{Env}(\text{Belief}(\pi \cdot as')) \subseteq \text{Env}(\text{Belief}(\pi))$, and (2) If there are environments $j, j' \in \text{Env}(\text{Belief}(\pi))$ with $p_j(\text{last}(\pi), a, s') > 0 = p_{j'}(\text{last}(\pi), a, s')$, then $j' \notin \text{Env}(\text{Belief}(\pi \cdot as'))$ and thus $\text{Env}(\text{Belief}(\pi \cdot as')) \subset \text{Env}(\text{Belief}(\pi))$.*

Key to our analysis of MEMDPs is the notion of *belief-based* strategies.

► **Definition 11** (Belief-based strategy). *A strategy σ is belief-based when for all finite paths π, π' such that $\text{Belief}(\pi) = \text{Belief}(\pi')$ implies $\sigma(\pi) = \sigma(\pi')$. Belief-based strategies can also be written as a function $\sigma: S \times \mathcal{P}(I) \rightarrow \text{Dist}(A)$.*

Belief-based strategies are a form of finite-memory strategies and are representable by FSCs.

► **Lemma 12.** *A belief-based strategy $\sigma: S \times \mathcal{P}(I) \rightarrow \text{Dist}(A)$ for a MEMDP \mathcal{M} with states S and actions A can be represented by an FSC.*

Similar to how states may be winning, beliefs can also be winning.

► **Definition 13** (Winning belief). *We call the belief $\langle s, J \rangle$ winning for objective Φ in \mathcal{M} , written as $\langle s, J \rangle \models^{\mathcal{M}} \Phi$, if there exists a strategy $\sigma: \text{Path}_{\text{fin}} \rightarrow \text{Dist}(A)$ such that for every environment $j \in J$, the induced MC is winning. That is, $\forall j \in J: \mathcal{M}_j[\sigma] \models \Phi$.*

A MEMDP \mathcal{M} is winning, $\mathcal{M} \models \Phi$, iff the initial belief $\langle \iota, I \rangle$ is winning. We can extend the notion of a winning region to beliefs. The (belief) winning region of a MEMDP \mathcal{M} is $Win_{\mathcal{M}}(\Phi) = \{\langle s, J \rangle \in S \times \mathcal{P}(I) \mid \langle s, J \rangle \models^{\mathcal{M}} \Phi\}$. The notion of winning beliefs has been used before in the context of POMDPs with other almost-sure objectives [10].

The induced MEMC of a MEMDP and FSC conservatively extends the standard product construction between an MDP and FSC to be applied to each transition function $\{p_i\}_{i \in I}$ individually. In that MEMC, the objective must be lifted to the new state space $S \times N$.

► **Definition 14** (Lifted Rabin objective). *For Rabin objective $\Phi = \{\langle \mathfrak{B}_i, \mathfrak{C}_i \rangle \mid 1 \leq i \leq k. \mathfrak{C}_i \subseteq \mathfrak{B}_i \subseteq S\}$ on S , the lifted Rabin objective to $S \times N$ is $\hat{\Phi} := \{\langle \mathfrak{C}_i \times N, \mathfrak{B}_i \times N \rangle \mid 1 \leq i \leq k\}$.*

When clear from the context, we implicitly apply this lifting where needed.

4.2 Constructing Winning Belief-Based Strategies

We are now ready to state the main theorem of this section.

► **Theorem 15.** *For MEMDP \mathcal{M} and Rabin objective Φ , there exists a winning strategy σ for Φ iff there exists a belief-based strategy σ^* that is winning for Φ .*

The remainder of this section is dedicated to the necessary ingredients to prove Thm. 15.

► **Definition 16** (Allowed actions). *The set of allowed actions for a winning belief $\langle s, J \rangle$ is $Allow(\langle s, J \rangle) := \{a \in A(s) \mid \forall \langle s', J' \rangle \in BU(\langle s, J \rangle, a): \langle s', J' \rangle \models^{\mathcal{M}} \Phi\}$.*

That is, an action at a winning belief is allowed if all possible resulting successor beliefs are still winning. Using allowed actions we define the belief-based strategy $\sigma_{Allow}: S \times \mathcal{P}(I) \rightarrow Dist(A)$:

$$\sigma_{Allow}(\langle s, J \rangle) := \begin{cases} \text{unif}(Allow(\langle s, J \rangle)) & \text{if } Allow(\langle s, J \rangle) \neq \emptyset, \\ \text{unif}(A(s)) & \text{otherwise.} \end{cases}$$

The strategy σ_{Allow} randomizes uniformly over all allowed actions when the successor beliefs are still winning and over all actions when the belief cannot be winning. We now sketch how to use σ_{Allow} to construct a winning belief-based strategy. See App. C for the details.

When playing σ_{Allow} , the induced MEMC $\mathcal{M}[\sigma_{Allow}]$ will almost-surely end up in a BSCC. Given belief $\langle s, J \rangle$, we compute all environments $j \in J$ for which $\mathcal{M}_j[\sigma_{Allow}]$ is in a BSCC S_B : $\mathcal{J}_{\langle s, J \rangle} := \{j \in J \mid \exists S_B \subseteq S \times \mathcal{P}(I): S_B \text{ is a BSCC in MC } \mathcal{M}[\sigma_{Allow}]_j \wedge \langle s, J \rangle \in S_B\}$.

As a consequence, since σ_{Allow} is a belief-based strategy, every BSCC of the MEMC $\mathcal{M}[\sigma_{Allow}]$ has a fixed set of environments that cannot change anymore. As σ_{Allow} uniformly randomizes over all allowed actions, and it remains possible to win, there has to exist a strategy $\sigma_{BSCC}: S_B \rightarrow Dist(A)$ that is a sub-strategy of σ_{Allow} , *i.e.*, it only plays a subset of actions that are also played by σ_{Allow} . We construct appropriate sub-MDPs to compute σ_{BSCC} for a belief $\langle s, J \rangle$ that is almost-surely winning for Φ . Using σ_{Allow} and σ_{BSCC} , we construct the following belief-based strategy and show it is indeed winning. The other direction follows since beliefs are based on paths, which proves Thm. 15.

$$\sigma^*(\langle s, J \rangle) := \begin{cases} \sigma_{Allow}(\langle s, J \rangle) & \text{if } \mathcal{J}_{\langle s, J \rangle} = \emptyset, \\ \sigma_{BSCC}(\langle s, J \rangle) & \text{otherwise.} \end{cases}$$

Thm. 15 shows that belief-based strategies are sufficient for almost-sure Rabin objectives in MEMDPs, which is not true on more general POMDPs [12]. Key is the monotonic information gain in MEMDPs, a property that POMDPs do not have in general [11].

► **Remark.** For the remainder of this paper, we assume all strategies are finite-memory strategies, and the induced (ME)MCs are defined via the product construction from Sect. 2.2.

5 Explicitly Adding Belief to MEMDPs

Above, we showed that it is sufficient for a strategy to reason over the beliefs. Now, we show how to add beliefs to the MEMDP, yielding a *belief observation MDP* (BOMDP). We discuss their construction (Sect. 5.1) and then algorithms on BOMDPs for reachability (Sect. 5.2) and so-called safe Büchi objectives (Sect. 5.3). We discuss Rabin objectives in Sect. 6.

5.1 Belief-Observation MDPs

We create a product construction between the MEMDP and the beliefs $\mathcal{P}(I)$ such that the beliefs of the MEMDP \mathcal{M} are directly encoded in the state space:

► **Definition 17** (BOMDP). *The belief observation MDP (BOMDP) of MEMDP $\mathcal{M} = \langle S, A, \iota, \{p_i\}_{i \in I} \rangle$ is a MEMDP $\mathcal{B}_{\mathcal{M}} = \langle S', A, \iota', \{p'_i\}_{i \in I} \rangle$ with states $S' = S \times \mathcal{P}(I)$, initial state $\iota' = \langle \iota, I \rangle$, and partial transition functions that are defined when $a \in A(s)$ such that*

$$p'_j(\langle s, J \rangle, a, \langle s', J' \rangle) = \begin{cases} p_j(s, a, s') & \text{if } \langle s', J' \rangle = \text{BU}(\langle s, J \rangle, a, s') \wedge j \in J, \\ 0 & \text{otherwise.} \end{cases}$$

BOMDPs are special MEMDPs; hence, all definitions for MEMDPs apply to BOMDPs. Due to the product construction, a belief-support-based strategy for \mathcal{M} can be turned into a memoryless strategy for $\mathcal{B}_{\mathcal{M}}$, and vice versa. In BOMDPs, the belief J is already part of the state, so we simplify the satisfaction notation to $\langle s, J \rangle \models^{\mathcal{B}_{\mathcal{M}}} \Phi$ instead of $\langle \langle s, J \rangle, J \rangle \models^{\mathcal{B}_{\mathcal{M}}} \Phi$.

► **Definition 18** (Lifted strategy). *Given MEMDP \mathcal{M} and a belief-based strategy σ . The lifted memoryless strategy $\hat{\sigma}: (S \times \mathcal{P}(I)) \rightarrow \text{Dist}(A)$ on $\mathcal{B}_{\mathcal{M}}$ is $\hat{\sigma}(s, J) := \sigma(\langle s, J \rangle)$.*

This lifting ensures that belief-based strategies and their liftings to BOMDPs coincide.

► **Lemma 19.** *Given a MEMDP \mathcal{M} , its BOMDP $\mathcal{B}_{\mathcal{M}}$, a belief-based strategy σ for \mathcal{M} and its lifted strategy $\hat{\sigma}$ for $\mathcal{B}_{\mathcal{M}}$, we have that the two induced MEMCs coincide: $\mathcal{M}[\sigma] = \mathcal{B}_{\mathcal{M}}[\hat{\sigma}]$.*

Consequently, satisfaction of objectives is preserved by the transformation.

► **Theorem 20.** *Let \mathcal{M} be a MEMDP with state space S and Φ a Rabin objective. Let $\hat{\Phi}$ be the lifted Rabin objective to $S \times \mathcal{P}(I)$ by Def. 14. A belief-based strategy σ for \mathcal{M} is winning the Rabin objective Φ iff the lifted strategy $\hat{\sigma}$ is winning the lifted objective $\hat{\Phi}$ for $\mathcal{B}_{\mathcal{M}}$: $\forall \sigma: \langle s, J \rangle \models^{\mathcal{M}[\sigma]} \Phi \Leftrightarrow \langle s, J \rangle \models^{\mathcal{B}_{\mathcal{M}}[\hat{\sigma}]} \hat{\Phi}$.*

As a result of Thm. 20, we will implicitly lift strategies and objectives.

5.2 An Algorithm for Reachability in BOMDPs

In this subsection, we establish an algorithm for computing the winning region for reachability objectives in a BOMDP. The winning region of a BOMDP $\mathcal{B}_{\mathcal{M}}$ is precisely the set of winning beliefs of its MEMDP \mathcal{M} : $\text{Win}_{\mathcal{B}_{\mathcal{M}}}(\Phi) = \{\langle s, J \rangle \in S \times \mathcal{P}(I) \mid \langle s, J \rangle \models^{\mathcal{M}} \Phi\}$. The algorithm specializes a similar fixed-point computation for POMDPs [12] to BOMDPs.

Alg. 1 computes these winning regions. It relies on a state-remove operation defined below. Intuitively, the algorithm iteratively removes losing states, which does not affect the winning region until all states that remain in $\mathcal{B}_{\mathcal{M}}$ are winning.

Removing state s from a BOMDP removes the state and disables outgoing action from any state where that action that could reach s with positive probability. This operation thus also removes any action and its transitions that could reach the designated state.

■ **Algorithm 1** Reachability algorithm for a BOMDP $\mathcal{B}_{\mathcal{M}}$ of MEMDP \mathcal{M} .

```

1: function REACH(BOMDP  $\mathcal{B}_{\mathcal{M}}$ ,  $T \subseteq S$ )
2:   do
3:     for  $i \in I$  do
4:        $S_i \leftarrow \{\langle s, J \rangle \in S \times \mathcal{P}(I) \mid i \in J\}$ 
5:       for  $\langle s, J \rangle \in S_i \setminus \text{Win}_{\mathcal{B}_{\mathcal{M}_i}}(\diamond T)$  do ▷ Iterate over all losing states
6:          $\mathcal{B}_{\mathcal{M}} \leftarrow \text{StateRemove}(\mathcal{B}_{\mathcal{M}}, \langle s, J \rangle)$  ▷ See Def. 21
7:       while  $\bigwedge_{i \in I} S_i \neq \text{Win}_{\mathcal{B}_{\mathcal{M}_i}}(\diamond T)$  ▷ Check if stable
8:       return  $S^{\mathcal{B}_{\mathcal{M}}}$ 

```

► **Definition 21** (State removal). Let $\mathcal{B}_{\mathcal{M}} = \langle S \times \mathcal{P}(I), A, \iota, \{p_i\}_{i \in I} \rangle$ be a BOMDP, and $\perp \notin S \times \mathcal{P}(I)$ a sink state. The BOMDP $\text{StateRemove}(\mathcal{B}_{\mathcal{M}}, \langle s, J \rangle)$ for $\mathcal{B}_{\mathcal{M}}$ and state $\langle s, J \rangle \in S \times \mathcal{P}(I)$ is given by $\langle \{\perp\} \cup S \times \mathcal{P}(I) \setminus \{\langle s, J \rangle\}, A, \iota', \{p'_i\}_{i \in I} \rangle$, where $\iota' = \perp$ if $\langle s, J \rangle = \iota$, and $\iota' = \iota$ otherwise, and for all states $\langle s', J' \rangle \neq \langle s, J \rangle$ and environments $i \in I$ we have

$$p'_i(\langle s', J' \rangle, a) = \begin{cases} p_i(\langle s', J' \rangle, a) & \text{if } \langle s, J \rangle \notin \text{Supp}(p_i(\langle s', J' \rangle, a)), \\ \text{dirac}(\perp) & \text{if } \langle s, J \rangle \in \text{Supp}(p_i(\langle s', J' \rangle, a)). \end{cases}$$

The main results in this section are the correctness and the complexity of Alg. 1:

► **Theorem 22.** For BOMDP $\mathcal{B}_{\mathcal{M}}$ and targets T : $\text{Win}_{\mathcal{B}_{\mathcal{M}}}(\diamond T) = \text{REACH}(\mathcal{B}_{\mathcal{M}}, T)$ in Alg. 1.

Towards a proof, the notions of losing states and strategies as defined for MDPs also apply to BOMDP states and strategies. For BOMDPs, we additionally define *losing actions* as state-action pairs that lead with positive probability to a losing state. It follows that a BOMDP state is losing iff every action from that state is losing, and a single environment where a BOMDP state is losing suffices as a witness that the state is losing in the BOMDP (see the App. D). Finally, the following lemma is the key ingredient to the main theorem.

► **Lemma 23.** Removing losing states from $\mathcal{B}_{\mathcal{M}}$ does not affect the winning region, i.e., $\langle s, J \rangle \notin \text{Win}_{\mathcal{B}_{\mathcal{M}}}(\diamond T)$ implies $\text{Win}_{\text{StateRemove}(\mathcal{B}_{\mathcal{M}}, \langle s, J \rangle)}(\diamond T) = \text{Win}_{\mathcal{B}_{\mathcal{M}}}(\diamond T)$.

► **Lemma 24.** Alg. 1 takes polynomial time in the size of $\mathcal{B}_{\mathcal{M}}$.

5.3 Safe Büchi in BOMDPs

In this section, we consider winning regions for *safe Büchi objectives* of the form $\square \mathfrak{B} \wedge \square \diamond \mathfrak{C}$, where $\mathfrak{C} \subseteq \mathfrak{B} \subseteq S$. The condition $\mathfrak{C} \subseteq \mathfrak{B}$ is convenient but does not restrict the expressivity. These objectives are essential for our Rabin algorithm in Sect. 6. The main result is:

► **Theorem 25.** For BOMDP $\mathcal{B}_{\mathcal{M}}$, $\text{Win}_{\mathcal{B}_{\mathcal{M}}}(\square \mathfrak{B} \wedge \square \diamond \mathfrak{C})$ is computable in polynomial time.

We provide the main ingredients for the proof below. We first consider arbitrary MEMDPs.

► **Definition 26** (State restricted (ME)MDP). Let $M = \langle S, A, \iota, p \rangle$ be an MDP and $S' \subseteq S$ a set of states. The MDP $M_{\square S'} := \langle S' \cup \{\perp\}, A, \iota', p' \rangle$ is M restricted to S' , with \perp a sink state, $\iota' = \iota$ if $\iota \in S'$ and \perp otherwise, and for $s \in S'$, $a \in A(s)$ and $s' \in S' \cup \{\perp\}$, we define:

$$p'(s, a, s') := \begin{cases} \sum_{s'' \in S \setminus S'} p(s, a, s'') & \text{if } s' = \perp, \\ p(s, a, s') & \text{otherwise.} \end{cases}$$

This definition conservatively extends to MEMDPs per environment i : $(\mathcal{M}_{\square S'})_i = (\mathcal{M}_i)_{\square S'}$.



■ **Figure 2** Example of a BOMDP fragment with Rabin objective $\Phi = \{\langle\{s_1\}, \{s_1\}\rangle, \langle\{s_2\}, \{s_2\}\rangle\}$.

The winning regions of a MEMDP \mathcal{M} and $\mathcal{M}_{\square\mathfrak{B}}$ coincide as, intuitively, winning strategies must remain in \mathfrak{B} , thus removing other states does not affect the winning region.

► **Lemma 27.** *The winning regions for $\square\mathfrak{B} \wedge \square\Diamond\mathfrak{C}$ with $\mathfrak{C} \subseteq \mathfrak{B}$ in $\mathcal{M}_{\square\mathfrak{B}}$ and \mathcal{M} coincide.*

Satisfying the Büchi objective $\square\Diamond\mathfrak{C}$ inside $\mathcal{M}_{\square\mathfrak{B}}$ implies satisfying the safety condition, thus:

► **Lemma 28.** *The winning regions for $\square\mathfrak{B} \wedge \square\Diamond\mathfrak{C}$ with $\mathfrak{C} \subseteq \mathfrak{B}$ and $\square\Diamond\mathfrak{C}$ in $\mathcal{M}_{\square\mathfrak{B}}$ coincide.*

We can lift these lemmas to the BOMDP associated with a MEMDP.

► **Lemma 29.** *The winning regions for $\square\mathfrak{B} \wedge \square\Diamond\mathfrak{C}$ with $\mathfrak{C} \subseteq \mathfrak{B}$ in $\mathcal{B}_{\mathcal{M}}$ and $\mathcal{B}_{(\mathcal{M}_{\square\mathfrak{B}})}$ coincide.*

Almost-sure Büchi objectives can be reduced to almost-sure reachability objectives using a construction similar to the one in [5], see the proof in the App. D for details.

► **Lemma 30.** *Büchi in BOMDPs is decidable in polynomial time.*

Now, to prove Thm. 25, $\text{Win}_{\mathcal{B}_{\mathcal{M}}}(\square\mathfrak{B} \wedge \square\Diamond\mathfrak{C})$ is computable as Büchi objective on a polynomially larger MEMDP (Lem. 29) in polynomial time (Lem. 30).

6 A Recursive PSPACE Algorithm for Rabin Objectives

We now show how to exploit the structure of BOMDPs to arrive at our PSPACE algorithm for Rabin objectives in MEMDPs. We first discuss the *non-local* behavior of Rabin objectives, and in particular, why the standard approach for almost-sure Rabin objectives for MDPs fails on BOMDPs. Then, in Sect. 6.2, we introduce *J-local MEMDPs*, which are MEMDPs where the belief J does not change. These *J-local* MEMDPs also occur as fragments of the BOMDPs. In *J-local* MEMDPs, whenever a transition is made that would cause a belief update to a strict subset of J , we transition to dedicated sink states, which we refer to as *frontier states*. These frontier states reflect transitioning into a different fragment of the BOMDP, from which all previously accessed BOMDP states are unreachable due to the monotonicity of the belief update operator. Next, in Sect. 6.3, we present an algorithm for efficiently computing the winning region of Rabin objectives on *J-local* MEMDPs. Finally, in Sect. 6.4, we prove that frontier states can be summarized as being either winning or losing, ultimately leading to a PSPACE algorithm for deciding Rabin objectives in MEMDPs.

6.1 Non-Local Behavior of Rabin Objectives

The traditional approach for checking almost-sure Rabin objectives on MDPs, see *e.g.* [6], computes for each state $s \in S$, whether there is a strategy that *immediately* satisfies a Rabin pair $\Phi_i = \langle\mathfrak{B}_i, \mathfrak{C}_i\rangle$, *i.e.*, satisfying $\square\mathfrak{B}_i \wedge \square\Diamond\mathfrak{C}_i$, and is a stronger condition. A state satisfies the Rabin condition Φ iff it almost-surely reaches the set of immediately winning states (the *win set*). The example below illustrates why this approach fails to generalize to MEMDPs.

► **Example 31.** In Fig. 2, we see a BOMDP for which the “MDP approach” does *not* work. First, note that the only strategy that always plays a is winning in every state. Now, consider the algorithm and the first Rabin pair $\Phi_1 = \langle\{s_1\}, \{s_1\}\rangle$. State $\langle s_2, \{2\} \rangle$ does not satisfy

$\Box\{s_1\} \wedge \Box\Diamond\{s_1\}$. State $\langle s_1, \{1, 2\} \rangle$ also does not belong to the win set, as in \mathcal{M}_2 there is a $1/2$ probability of reaching the sink state $\langle s_2, \{2\} \rangle$. For the second Rabin pair, (only) state $\langle s_2, \{2\} \rangle$ is immediately winning. Thus, the win set is the singleton set containing $\langle s_2, \{2\} \rangle$. From the initial state $\langle s_1, \{1, 2\} \rangle$, it is not possible to almost-surely reach the state $\langle s_2, \{2\} \rangle$, due to \mathcal{M}_1 . Therefore, a straightforward adaption of the traditional algorithm for MDPs would yield that the initial state is losing.

The difficulty in the example above lies in the fact that in the different environments, a different Rabin pair is satisfied. However, taking the self-loop in s_1 does not update the belief and it remains unclear whether we will eventually satisfy Φ_1 or Φ_2 .

6.2 Local View on BOMDPs

We formalize J -local MEMDPs, that transition into frontier states if the belief updates.

► **Definition 32** (J -local MEMDPs). *Given a MEMDP $\mathcal{M} = \langle S, A, \iota, \{p_i\}_{i \in I} \rangle$, the J -local MEMDP $\mathcal{M}\{J\} = \langle S \sqcup F, A, \{p'_j\}_{j \in J}, \iota \rangle$ is a MEMDP, with as state space the disjoint union of the (original) states S and the frontier states $F := S \times A \times S$. The transition functions $\{p'_j: S \sqcup F \times A \rightarrow \text{Dist}(S \sqcup F)\}_{j \in J}$ are defined s.t. (1) $p'_j(f, a, f) = 1$ for all $f \in F$, (2) $p'_j(s, a)$ is undefined if $p_j(s, a)$ is undefined, and (3) for every state $s \in S$ and $a \in A(s)$, we define $p'_j(s, a, \langle s, a, s' \rangle) = p_j(s, a, s')$ if $\text{BU}(\langle s, J \rangle, a, s') \neq \langle s', J \rangle$ and $p'_j(s, a, s') = p_j(s, a, s')$ otherwise.*

By definition of the transition functions $\{p'_j\}_{j \in J}$ of a J -local MEMDP $\mathcal{M}\{J\}$, all environments of $\mathcal{M}\{J\}$ share the same underlying graph within the states of S . Transitions to the frontiers may, however, differ (made formal in App. E). As both \mathcal{M} and $\mathcal{M}\{J\}$ have states in S , a Rabin objective Φ can readily be applied to both. To give meaning to the frontier states F in $\mathcal{M}\{J\}$, we introduce *localized Rabin objectives*:

► **Definition 33** (Localized Rabin objective, winning frontier). *Given Rabin objective $\Phi = \{\langle \mathfrak{B}_i, \mathfrak{C}_i \rangle \mid 1 \leq i \leq k\}$ and some subset of frontier state $WF \subseteq F$, the localized Rabin objective for J -local MEMDP $\mathcal{M}\{J\}$ is $\Phi^{\text{Loc}}(WF) := \{\langle \mathfrak{B}_i \cup WF, \mathfrak{C}_i \cup WF \rangle \mid 1 \leq i \leq k\}$. We call WF the winning frontier, as any path that reaches a state in WF is winning.*

6.3 An Algorithm for Localized Rabin Objectives

Below, we present an algorithm to compute the winning region of a localized Rabin objective on a J -local MEMDP, using some auxiliary definitions on winning in a J -local MEMDPs.

► **Definition 34** (Immediately winning Rabin pair/state). *A J -local MEMDP state $s \in S \sqcup F$ has an immediately winning Rabin pair $\Phi_i = \langle \mathfrak{B}_i, \mathfrak{C}_i \rangle$ when $s \models^{\mathcal{M}\{J\}} \Box\mathfrak{B}_i \wedge \Box\Diamond\mathfrak{C}_i$. A state $s \in S \sqcup F$ is immediately winning if it has an immediately winning Rabin pair.*

Immediately winning states are, in particular, also winning states (see Lem. 59, App. E). It is natural also to consider specialized winning regions for just immediately winning states:

► **Definition 35.** *The Rabin win set $W_{\Phi^{\text{Loc}}}$ is $\{s \in S \sqcup F \mid s \text{ is immediately winning}\}$.*

The crux of our algorithm is that in J -local MEMDPs, as in MDPs but unlike in BOMDPs, winning a Rabin objective is equivalent to almost-surely reaching the Rabin win set.

► **Lemma 36.** *A state s in a J -local MEMDP is winning iff it can almost-surely reach $W_{\Phi^{\text{Loc}}}$.*

■ **Algorithm 2** Local Rabin Algorithm.

```

1: function RABIN(Local MEMDP  $\mathcal{L} = \mathcal{M}\{J\}$ ,  $WF$ ,  $\Phi = \{\langle \mathfrak{B}_1, \mathfrak{C}_1 \rangle, \dots, \langle \mathfrak{B}_n, \mathfrak{C}_n \rangle\}$ )
2:    $S_{win} \leftarrow \emptyset$ 
3:   for  $1 \leq i \leq n$  do
4:      $\mathfrak{B}'_i \leftarrow \mathfrak{B}_i \cup WF$  ;  $\mathfrak{C}'_i \leftarrow \mathfrak{C}_i \cup WF$ 
5:      $S_{win} \leftarrow S_{win} \cup Win_{\mathcal{L}}(\Box \mathfrak{B}'_i \wedge \Box \Diamond \mathfrak{C}'_i)$  ▷ See Thm. 25
6:   return  $Win_{\mathcal{L}}(\Diamond S_{win})$ 

```

We sketch the proof ingredients later. We first introduce Alg. 2, which lifts the MDP approach (Sect. 6.1) to J -local MEMDPs. The set S_{win} on line 2 stores states for which an immediately winning Rabin pair has been found. For each Rabin pair Φ_i , the algorithm computes the localized Rabin pair Φ_i^{Loc} . Next, in line 5, it compute the winning region $Win_{\mathcal{L}}(\Box \mathfrak{B}'_i \wedge \Box \Diamond \mathfrak{C}'_i)$ using the approach described in Sect. 5.3. These are exactly the states that have Φ_i^{Loc} as an immediately winning Rabin pair, *i.e.*, they constitute the win set S_{win} . Finally, the algorithm outputs the winning region by computing states that almost-surely reach S_{win} using Alg. 1.

► **Theorem 37.** *Alg. 2 yields winning regions for local MEMDPs and localized Rabin objectives.*

The remainder of this subsection discusses the ingredients for proving Lem. 36 and the theorem above. Therefore, we consider the induced Markov chain C of environment j under any strategy, *i.e.*, $C = \mathcal{M}\{J\}[\sigma]_j$. In any state that is in a BSCC of C , we notice that the reachable states *in any environment* are contained by the BSCC and the frontier states. Furthermore, we observe that in any environment, either the BSCCs in those states are the original BSCC or are (trivial) BSCCs in the frontier. Formal statements are given in App. E. The next lemma shows that states that are (under a winning strategy and in some environment) in a BSCC are immediately winning with some Rabin pair. The main challenge is that this BSCC may not be a BSCC in every environment. Using the observations above, if the states do not constitute a BSCC, they will almost surely reach (winning) frontier states, which allows us to derive the following formal statement:

► **Lemma 38.** *Given a J -local MEMDP $\mathcal{M}\{J\}$ and a winning strategy σ . Every state that is in a BSCC S_{B_j} of $\mathcal{M}\{J\}[\sigma]_j$ of some environment $j \in J$, is in $W_{\Phi^{Loc}}$.*

With this statement, we can now prove Lem. 36 as under any winning strategy, we almost-surely end up in BSCCs. We return to the proof of the main theorem about the correctness of Alg. 2. First, we observe that we correctly identify the immediately winning states.

► **Lemma 39.** *Alg. 2 computes the set of states that are immediately winning, $W_{\Phi^{Loc}}$.*

Lems. 36 and 39 together prove Thm. 37. Finally, we remark:

► **Lemma 40.** *Alg. 2 is a polynomial time algorithm.*

6.4 Recursive Computation of Winning Regions

We now detail how to combine the local computations of winning regions towards a global winning region. Furthermore, we show that to obtain the winning region at the root (*i.e.*, I -local), we can forget about the winning regions below and, consequently, present a recursive approach (akin to [40]) to decide almost-sure Rabin objectives for MEMDPs in PSPACE.

► **Theorem 41.** *Winning almost-sure Rabin objectives in MEMDPs is decidable in PSPACE.*

■ **Algorithm 3** Generic recursive algorithm for MEMDPs.

```

1: function CHECK(MEMDP  $\mathcal{M} = \langle S, A, \iota, \{p_i\}_{i \in I}, \Phi \rangle$ )
2:    $\mathcal{L} \leftarrow \mathcal{M}\{I\}$ 
3:    $RF \leftarrow \text{Reachable}(S^{\mathcal{L}}) \cap F^{\mathcal{L}}$  ▷ Compute the reachable frontier states
4:    $WF \leftarrow \{(s, a, s') \in RF \mid \langle s', J' \rangle = \text{BU}(\langle s, J \rangle, a, s') \wedge \text{CHECK}(\mathcal{M}^{\downarrow_{J'}}, \Phi)\}$ 
5:   return  $\iota \in \text{RABIN}(\mathcal{L}, WF, \Phi)$  ▷ Compute winning set with winning frontier

```

In the remainder, we show this by providing a recursive algorithm and proving its correctness. An important construction is to project the winning region into a particular set of beliefs.

► **Definition 42** (Belief-restricted winning regions). *For a Rabin objective Φ , we define the following restrictions of the winning region: (1) $\text{Win}_{\mathcal{M}}(\Phi)_J := \text{Win}_{\mathcal{M}}(\Phi) \cap (S \times \{J\})$, (2) $\text{Win}_{\mathcal{M}}(\Phi)_{\subset J} := \bigcup_{J' \subset J} \text{Win}_{\mathcal{M}}(\Phi)_{J'}$, and (3) $\text{Win}_{\mathcal{M}}(\Phi)_{\subseteq J} := \text{Win}_{\mathcal{M}}(\Phi)_{\subset J} \cup \text{Win}_{\mathcal{M}}(\Phi)_J$.*

We now define the localized Rabin objective where we determine the winning frontiers based on the actual winning states in a BOMDP. We use the following auxiliary notation: We define the reachable frontier $RF := \text{Reachable}(S) \cap F$. Then, we can determine where a local transition $s \xrightarrow{a} s'$ leads in the global system, $\text{ToGlob}_J(\langle s, a, s' \rangle) := \text{BU}(\langle s, J \rangle, a, s')$ and finally consider $\text{WinLocal}_J(F, B) := \{f \in F \mid \text{ToGlob}_J(f) \in B\}$.

► **Definition 43** (Correct localized Rabin objective). *For belief J , the correct localized Rabin objective is $\Phi^{CLoc}(J) := \Phi^{Loc}(\text{WinLocal}_J(RF, \text{Win}_{\mathcal{M}}(\Phi)_{\subset J}))$.*

The notion of correctness in the definition above is justified by the following theorem, which says that computing the correct localized Rabin objective provides the belief-restricted winning region. That is, the winning region of the J -local MEMDP $\mathcal{M}\{J\}$ with its correct localized Rabin objective is equal to the global winning region restricted to J .

► **Theorem 44.** *For Rabin objective Φ : $(\text{Win}_{\mathcal{M}\{J\}}(\Phi^{CLoc}(J)) \cap S) \times \{J\} = \text{Win}_{\mathcal{M}}(\Phi)_J$.*

The theorem immediately leads to the following characterization of the winning region.

► **Corollary 45.** *For Rabin objective Φ : $\text{Win}_{\mathcal{M}}(\Phi) = \bigcup_J (\text{Win}_{\mathcal{M}\{J\}}(\Phi^{CLoc}(J)) \cap S) \times \{J\}$.*

Cor. 45 suggests computing the winning region from local MEMDPs. The computation can go bottom-up, as the winning region of a MEMDP restricted to a belief J only depends on the J -local MEMDP $\mathcal{M}\{J\}$ and the winning regions of beliefs $J' \subset J$. These observations lead us to Alg. 3. We construct the J -local MEMDP, recursively determine the winning status of all its frontier states, and then compute the local winning region of $\mathcal{M}\{J\}$.

► **Theorem 46.** *In Alg. 3 with Rabin objective Φ : $\text{CHECK}(\mathcal{M}, \Phi)$ iff $\iota \in \text{Win}_{\mathcal{M}}(\Phi)$.*

► **Lemma 47.** *Alg. 3 runs in polynomial space.*

This lemma follows from observing that a local MEMDP and thus its frontier is polynomial and that the recursion depth is limited by $|I|$. Thm. 46 and Lem. 47 together prove the main theorem Thm. 41: The decision problem of almost-sure Rabin objectives in MEMDPs is in PSPACE. Thus, almost-sure safety, Büchi, co-Büchi, and parity are in PSPACE too [13].

► **Theorem 48.** *The time complexity of Alg. 3 is in $O((|S|^2 \cdot |A|)^{|I|} \cdot \text{poly}(|\mathcal{M}|, |\Phi|))$.*

The bound in Thm. 48 is conservative², and it shows that deciding almost-sure Rabin objectives for 2-MEMDPs is in P. Almost-sure parity objectives for 2-MEMDPs were already known to be in P [36]. Indeed, it establishes the complexity for any fixed number of constants³.

► **Corollary 49.** *For constant k , deciding almost-sure Rabin for k -MEMDPs is in P.*

7 Conclusion

We have presented a PSPACE algorithm for almost-sure Rabin objectives in MEMDPs. This result establishes PSPACE-completeness for many other almost-sure objectives, including parity, and completes the complexity landscape for MEMDPs. We additionally showed that all objectives under the possible semantics we consider in MEMDPs belong to the same complexity classes as MDPs. Interesting directions for future work are to investigate whether the constructions used in this paper can also be of benefit for quantitative objectives in MEMDPs or more expressive subclasses of POMDPs, for example, a form of MEMDPs where the environments may change over time.

References

- 1 Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *AAAI*, pages 2669–2678. AAAI Press, 2018.
- 2 Roman Andriushchenko, Milan Ceska, Sebastian Junges, Joost-Pieter Katoen, and Simon Stupinský. PAYNT: A tool for inductive synthesis of probabilistic programs. In *CAV (1)*, volume 12759 of *LNCS*, pages 856–869. Springer, 2021.
- 3 Sebastian Arming, Ezio Bartocci, Krishnendu Chatterjee, Joost-Pieter Katoen, and Ana Sokolova. Parameter-independent strategies for pMDPs via POMDPs. In *QEST*, volume 11024 of *LNCS*, pages 53–70. Springer, 2018.
- 4 Christel Baier, Nathalie Bertrand, and Marcus Größer. On decision problems for probabilistic Büchi automata. In *FoSSaCS*, volume 4962 of *LNCS*, pages 287–301. Springer, 2008.
- 5 Christel Baier, Marcus Größer, and Nathalie Bertrand. Probabilistic ω -automata. *J. ACM*, 59(1):1:1–1:52, 2012.
- 6 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 7 Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. Concurrent parameterized games. In *FSTTCS*, volume 150 of *LIPICs*, pages 31:1–31:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 8 Dietmar Berwanger and Laurent Doyen. On the power of imperfect information. In *FSTTCS*, volume 2 of *LIPICs*, pages 73–82. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008.
- 9 Iadine Chades, Josie Carwardine, Tara G. Martin, Samuel Nicol, Régis Sabbadin, and Olivier Buffet. MoMDPs: A solution for modelling adaptive management problems. In *AAAI*. AAAI Press, 2012.
- 10 Krishnendu Chatterjee, Martin Chmelík, Raghav Gupta, and Ayush Kanodia. Optimal cost almost-sure reachability in POMDPs. *Artif. Intell.*, 234:26–48, 2016.
- 11 Krishnendu Chatterjee, Martin Chmelík, Deep Karkhanis, Petr Novotný, and Amélie Royer. Multiple-environment Markov decision processes: Efficient analysis and applications. In *ICAPS*, pages 48–56. AAAI Press, 2020.

² A more precise bound can likely be obtained from the number of revealing transitions in the MEMDP.

³ That is, the decidability problem is in XP with parameter *number of environments* k .

- 12 Krishnendu Chatterjee, Martin Chmelík, and Mathieu Tracol. What is decidable about partially observable Markov decision processes with ω -regular objectives. *J. Comput. Syst. Sci.*, 82(5):878–911, 2016.
- 13 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. The complexity of stochastic Rabin and Streett games’. In *ICALP*, volume 3580 of *LNCS*, pages 878–890. Springer, 2005.
- 14 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Qualitative analysis of partially-observable Markov decision processes. In *MFCS*, volume 6281 of *LNCS*, pages 258–269. Springer, 2010.
- 15 Krishnendu Chatterjee, Marcin Jurdzinski, and Thomas A. Henzinger. Quantitative stochastic parity games. In *SODA*, pages 121–130. SIAM, 2004.
- 16 Krishnendu Chatterjee and Mathieu Tracol. Decidable problems for probabilistic automata on infinite words. In *LICS*, pages 185–194. IEEE Computer Society, 2012.
- 17 Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. POMDP-lite for robust robot planning under uncertainty. In *ICRA*, pages 5427–5433. IEEE, 2016.
- 18 Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, and Ufuk Topcu. Convex optimization for parameter synthesis in MDPs. *IEEE Trans. Autom. Control.*, 67(12):6333–6348, 2022.
- 19 Luca de Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford University, USA, 1997.
- 20 Nathanaël Fijalkow. What is known about the value 1 problem for probabilistic automata? *CoRR*, abs/1410.3770, 2014.
- 21 Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, and Mateusz Skomra. Games on graphs. *CoRR*, abs/2305.10546, 2023.
- 22 Nathanaël Fijalkow, Hugo Gimbert, Edon Kelmendi, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. *Log. Methods Comput. Sci.*, 11(2), 2015.
- 23 Nathanaël Fijalkow, Hugo Gimbert, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. In *LICS*, pages 295–304. IEEE Computer Society, 2012.
- 24 Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.
- 25 Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16:1437–1480, 2015.
- 26 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *ICALP (2)*, volume 6199 of *LNCS*, pages 527–538. Springer, 2010.
- 27 Garud N. Iyengar. Robust dynamic programming. *Math. Oper. Res.*, 30(2):257–280, 2005.
- 28 Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Parameter synthesis in Markov models: A gentle survey. In *Principles of Systems Design*, volume 13660 of *LNCS*, pages 407–437. Springer, 2022.
- 29 Sebastian Junges, Nils Jansen, and Sanjit A. Seshia. Enforcing almost-sure reachability in POMDPs. In *CAV (2)*, volume 12760 of *LNCS*, pages 602–625. Springer, 2021.
- 30 Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.
- 31 Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *UAI*, pages 417–426. Morgan Kaufmann, 1999.
- 32 Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798, 2005.
- 33 Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Math. Oper. Res.*, 12(3):441–450, 1987.

- 34 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.
- 35 Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Algorithms for omega-regular games with imperfect information. *Log. Methods Comput. Sci.*, 3(3), 2007.
- 36 Jean-François Raskin and Ocan Sankur. Multiple-environment Markov decision processes. In *FSTTCS*, volume 29 of *LIPICs*, pages 531–543. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.
- 37 Marc Rigter, Bruno Lacerda, and Nick Hawes. Minimax regret optimisation for robust planning in uncertain Markov decision processes. In *AAAI*, pages 11930–11938. AAAI Press, 2021.
- 38 Marnix Suilen, Marck van der Vegt, and Sebastian Junges. A PSPACE algorithm for almost-sure Rabin objectives in multi-environment MDPs, 2024. [arXiv:2407.07006](https://arxiv.org/abs/2407.07006).
- 39 Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.
- 40 Marck van der Vegt, Nils Jansen, and Sebastian Junges. Robust almost-sure reachability in multi-environment MDPs. In *TACAS (1)*, volume 13993 of *LNCs*, pages 508–526. Springer, 2023.
- 41 Tobias Winkler, Sebastian Junges, Guillermo A. Pérez, and Joost-Pieter Katoen. On the complexity of reachability in parametric Markov decision processes. In *CONCUR*, volume 140 of *LIPICs*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

