30th EACSL Annual Conference on Computer Science Logic

CSL 2022, February 14–19, 2022, Göttingen, Germany (Virtual Conference)

Edited by

Florin Manea

Alex Simpson

LIPIcs – Vol. 216 – CSL 2022 www.dagstuhl.de/lipics

Editors

Florin Manea
University of Göttingen, Germany
florin.manea@informatik.uni-goettingen.de

Alex Simpson
University of Ljubljana, Slovenia
alex.simpson@fmf.uni-lj.si

ACM Classification 2012
Theory of computation → Logic

ISBN 978-3-95977-218-1

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/978-3-95977-218-1.

Publication date
February, 2022

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.

Digital Object Identifier: 10.4230/LIPIcs.CSL.2022.0

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.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

30th EACSL Annual Conference on Computer Science Logic (CSL 2022).
Editors: Florin Manea and Alex Simpson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at CSL 2022, the 30th meeting in the conference series *Computer Science Logic (CSL)*, the annual conference of the European Association for Computer Science Logic (EACSL). CSL 2022 was held from 14th to 19th February 2022. It was organised at the University of Göttingen, but took place as an on-line meeting due to the ongoing pandemic.

CSL started as a series of international workshops, and became an international conference in 1992. Previous instalments of CSL were held in Ljubljana (2022, on-line), Barcelona (2020), Birmingham (2018), Stockholm (2017), Marseille (2016), Berlin (2015), Vienna (2014), Torino (2013), Fontainebleau (2012), Bergen(2011), Brno (2010), Coimbra (2009), Bologna (2008), Lausanne (2007), Szeged (2006), Oxford (2005), Karpacz (2004), Vienna (2003), Edinburgh (2002), Paris (2001), Munich (2000), Madrid (1999), Brno (1998), Aarhus (1997), Utrecht (1996), Paderborn (1995), Kazimierz (1994), Swansea (1993) and San Miniato (1992).

CSL is an interdisciplinary conference, spanning both basic and application-oriented research in mathematical logic and computer science. It is a forum for the presentation of research on all aspects of logic and its applications, including automated deduction and interactive theorem proving, constructive mathematics and type theory, equational logic and term rewriting, automata and games, game semantics, modal and temporal logic, logical aspects of computational complexity, finite model theory, computational proof theory, logic programming and constraints, lambda calculus and combinatory logic, domain theory, categorical logic and topological semantics, database theory, specification, extraction and transformation of programs, logical aspects of quantum computing, logical foundations of programming paradigms, verification and program analysis, linear logic, higher-order logic, and non-monotonic reasoning.

The conference received 91 abstracts of which 75 were followed up by full-paper submissions. The programme committee selected 35 papers for presentation at the conference. Each paper was reviewed by at least three members of the programme committee, with the help of external reviewers. The submission and reviewing process, programme committee discussion, and author notifications were all handled by the Easychair conference management system. In addition to the contributed papers, there were five invited talks, by: Udi Boker (Interdisciplinary Center, Herzliya, Israel), Martín Escardó (University of Birmingham, UK), Rosalie Iemhoff (Utrecht University, the Netherlands), Karen Lange (Wellesley College, USA), and Annabelle McIver (Macquarie University, Australia). We thank the invited speakers for their stimulating talks and papers, which greatly contributed to the success of the conference.

One of the major regular events at CSL conferences is the presentation of the *Ackermann Award*: the annual EACSL award for an outstanding dissertation in the area of logic in computer science. The recipients of the award are selected by jury from a field of international nominees, and the recipients receive their award at a ceremony at which they give a prize lecture on their dissertation. This year, the jury elected to give the Ackermann Award 2021 to Marie Fortin for her thesis *Expressivity of first-order logic, star-free propositional dynamic logic and communicating automata* defended at ENS Paris-Saclay (France) in 2020, supervised by Paul Gastin and Benedikt Bollig and to Sandra Kiefer for her thesis *Power and Limits of the Weisfeiler-Leman Algorithm* defended at RWTH Aachen (Germany) in 2020, with examiners Martin Grohe, Pascal Schweitzer, and Neil Immerman. The awards were presented during the conference. The citation for the awards is included in the proceedings.

30th EACSL Annual Conference on Computer Science Logic (CSL 2022).
Editors: Florin Manea and Alex Simpson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A significant event at CSL 2022 was the presentation of the inaugural *Helena Rasiowa Award*, named after the eminent Polish mathematician and logician Helena Rasiowa (1917–1994) whose work had an essential impact on the emerging field of logic in computer science. The Helena Rasiowa Award is given to the best paper, as decided by the programme committee, that is written solely by students or to which students were the main contributors. This award, which was presented for the first time at CSL 2022, will be a regular feature of CSL conferences henceforth. There was a strong field of candidates for the inaugural award, with 10 of the accepted papers eligible. From these, the programme committee selected Antonio Casares as the recipient of the 2022 Helena Rasiowa Award, for his paper *On the Minimisation of Transition-Based Rabin Automata and the Chromatic Memory Requirements of Muller Conditions*. Antonio Casares is a PhD student at the University of Bordeaux under the supervision of Thomas Colcombet, Nathanaël Fijalkow and Igor Walukiewicz.

CSL 2022 also had two affiliated workshops: the 22nd International Workshop on Logic and Computational Complexity – LCC 2022 and the Logic Mentoring Workshop.

We are very grateful to all the members of the CSL 2022 programme committee and external reviewers for their careful and efficient evaluation of the papers submitted. We would like to thank also the members of the organisation committee Maria Kosche, Tore Koß, Patricia Nitzke, Viktoriya Pak, and Stefan Siemer, from the University of Göttingen, for taking care to ensure a smooth-running and enjoyable conference, a task that was complicated by organising a virtual meeting. The award-certificates offered at CSL 2022 to the winners of the Ackermann Award and the Helena Rasiowa Award were designed by Elena Lykiardopol, to whom we are very grateful. It was as always a pleasure to work with Thomas Schwentick who, as the EACSL president, provided excellent guidance. The proceedings of CSL 2022 are published as a volume in the LIPIcs series. We thank Michael Wagner, Michael Didas, and all the Dagstuhl/LIPIcs team for their ongoing support and for the high quality preparation of these proceedings. Last, but not least, we are very grateful to the University of Göttingen and to the German Research Foundation (DFG) for supporting the organisation of this conference.

Florin Manea and Alex Simpson 10th November 2021

Program Committee Members

Thorsten Altenkirch (Nottingham, UK)

Benedikt Bollig (Cachan, France)

Agata Ciabattoni (Vienna, Austria)

Liron Cohen (Ben-Gurion University, Israel)

Anupam Das (Birmingham, UK)

Claudia Faggian (Paris, France)

Francesco Gavazzo (Bologna, Italy)

Stefan Göller (Kassel, Germany)

Willem Heijltjes (Bath, UK)

Sandra Kiefer (Aachen, Germany)

Emanuel Kieronski (Wroclaw, Poland)

Bartek Klin (Warsaw, Poland)

Juha Kontinen (Helsinki, Finland)

Anthony Lin (Kaiserslautern, Germany)

Karoliina Lehtinen (Marseille, France)

Florin Manea (Göttingen, Germany, co-chair)

Fredrik Nordvall Forsberg (Strathclyde, UK)

Liat Peterfreund (Paris, France and Edinburgh, UK)

Daniela Petrisan (Paris, France)

Karin Quaas (Lepizig)

Alex Simpson (Ljubljana, Slovenia, co-chair)

Pawel Sobocinski (Tallin, Estonia)

Ana Sokolova (Salzburg, Austria)

Linda Brown Westrick (Connecticut, US)

30th EACSL Annual Conference on Computer Science Logic (CSL 2022).
Editors: Florin Manea and Alex Simpson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

30th EACSL Annual Conference on Computer Science Logic (CSL 2022).
Editors: Florin Manea and Alex Simpson

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Andreas Abel,
Matteo Acclavio,
Antonis Achilleos,
Andrea Aler Tubella,
Guillaume Allais,
Carlo Angiuli,
Clément Aubert,
Louwe B. Kuijer,
Patrick Baillot,
Thibaut Balabonski,
Philippe Balbiani,
Davide Barbarossa,
Pablo Barenbaum,
Bartosz Bednarczyk,
Steffen van Bergerem,
Ulrich Berger,
Mark Bickford,
James Brotherston,
Ulrik Buchholtz,
Damien Busatto-Gaston,
Simon Castellan,
David Cerna,
Bruno Courcelle,
Gianluca Curzi,
Francesco Dagnino,
Ugo Dal Lago,
Mariangiola Dezani-Ciancaglini,
Marko Doko,
Jannik Dreier,
Arnaud Durand,
Thomas Ehrhard,
Santiago Escobar,
Uli Fahrenberg,
David Fernández-Duque,
Marie Fortin,
Hadar Frenkel,
Jonas Frey,
Moses Ganardi,
Blaise Genest,
Marianna Girlando,
Erich Grädel,
Daniel Gratzer,
Giulio Guerrieri,
Christoph Haase,
Amar Hadzihasanovic,
Tuomas Hakoniemi,
Frederik Harwath,
Jules Hedges,
Lauri Hella,
Mathieu Hilaire,
Cameron Hill,
Petr Hlineny,
Stefan Hoffmann,
Ross Horne,
Jacob Howe,
Rasmus Ibsen-Jensen,
Farzad Jafarrahmani,
Tomi Janhunen,
Alex Kavvos,
Ariel Kellison,
Cynthia Kop,
Alexander Kozachinskiy,

Nicolai Kraus,
Stephan Kreutzer,
Dominique Larchey-Wendling,
Massimo Lauria,
Leonid Libkin,
Amaldev Manuel,
Sonia Marin,
Nicolas Markey,
Christoph Matheja,
Guy McCusker,
Arne Meier,
Edward Morehouse,
Muhammad Najib,
Chad Nester,
Lê Thành Dũng (Tito) Nguyễn,
Federico Olimpieri,
Paulo Oliva,
Paweł Parys,
Erik Paul,
Arno Pauly,
Thomas Powell,
Ian Pratt-Hartmann,
Andrei Rodin,
Adam Rogalewicz,
Simona Ronchi Della Rocca,
Benjamin Rossman,
Luca Roversi,
Jakub Rydval,
K. S. Thejaswini,
Alessio Santamaria.
Aleksy Schubert,
Peter Schuster,
Pascal Schweitzer,
Thomas Seiller,
Dmitry Shkatov,
Kan Shuanglong,
Michael Shulman,
Salomon Sickert,
David Sprunger,
Jonathan Sterling,
Olivier Stietel,
Lutz Straßburger,
Thomas Streicher,
Lidia Tendera,
Alwen Tiu,
Patrick Totzke,
Riccardo Treglia,
Daniel Turetsky,
Lionel Vaux Auclair,
Andrea Vezzosi,
Niels Voorneveld,
Paul Wilson,
Sarah Winter,
Xi Xiaoyong,
Chuangjie Xu,
Anna Zamansky,
Shufang Zhu,
Yoni Zohar.

The Ackermann Award 2021

by Michael Benedikt, Prakash Panangaden and Thomas Schwentick
For the Jury of the EACSL Ackermann Award

The 17th Ackermann Award is presented at CSL'22, which is held online and organised by the Fundamentals of Computer Science Group at University of Göttingen, Germany. The 2021 Ackermann Award was open to any PhD dissertation on any topic represented at the annual CSL and LICS conferences that were formally accepted by a degree-granting institution in fulfilment of the PhD degree between 1 January 2019 and 31 December 2020. The Jury received eight nominations for the 2021 Award. The candidates came from a number of different countries around the world. The institutions at which the nominees obtained their doctorates represent different countries in Europe and Asia.

Again this year, EACSL Ackermann Award is sponsored by the association *Alumni der Informatik Dortmund e.V.*[1]

The topics covered a wide range of areas in Logic and Computer Science as represented by the LICS and CSL conferences. All submissions were of a very high quality and contained significant contributions to their particular fields. The jury wish to extend their congratulations to all the nominated candidates for their outstanding work.

The wide range of excellent candidates presented the jury with an excruciating task. After an extensive discussion, the jury unanimously decided to award the **2021 Ackermann Award** to (in alphabetic order):

- Marie Fortin from France, for her thesis

  Expressivity of first-order logic, star-free propositional dynamic logicand communicating automata

  approved by *Université Paris-Saclay* in 2020,

  and

- Sandra Kiefer from Germany, for her thesis

  Power And Limits Of The Weisfeiler-Leman Algorithm

  approved by *RWTH Aachen* in 2020.

Citation for Marie Fortin

Marie Fortin shares the **2021 Ackerman Award** of the European Association of Computer Science Logic (EACSL) for her thesis

Expressivity of first-order logic, star-free propositional dynamic logic and communicating automata.

[1] `https://www.cs.tu-dortmund.de/nps/en/Alumni/index.html`

30th EACSL Annual Conference on Computer Science Logic (CSL 2022).
Editors: Florin Manea and Alex Simpson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The thesis studies the expressive power of first-order logic over ordered structures. This work has applications to message sequence charts, a fundamental structure that arises in the verification of concurrent systems and more generally to the connection between automata and logic. She proved a concurrent analogue of the classic Büchi-Elgot-Traktenbrot theorem for the relation between concurrent finite-state machines and message sequence charts. A second fundamental result obtained is the fact that first-order logic and FO restricted to three variables are equally expressive over message sequence charts.

Background to the thesis

Fortin's work belongs to the classic theme of relating automata to logics over suitable structures. The Büchi-Elgot-Traktenbrot theorem relates rfinite automata with monadic second-order logic over words. In the realm of concurrency a fundamental result of Mazurkiewicz relates his eponymous traces with asynchronous automata. For message sequence charts and concurrent finite-state automata it is much more complicated because this is a richer model. Earlier work by several authors established such connections in much more restricted cases. In message sequence charts there are three important order relations: $\leq$ the causal partial order which is generated by $\rhd$ and $\rightarrow_p$ which is the immediate precedence relation on a process p. The best existing result was a proof due to Bollig and Leuker that CFSM's are expressively equivalent to $EMSO(\rightarrow_p, \rhd)$, the existential fragment of second-order monadic logic over the two given relations. Capturing the causal relation was out of reach but essential to express many fundamental properties of concurrent computation.

Contributions of the thesis

Dr. Fortin made several fundamental contributions. First she showed that $EMSO^2(\rightarrow_p, \rhd, \leq)$, which is the fragment restricted to two variables is expressively equivalent to CFSM's. This was a proof requiring great technical skill and originality. Later she settled the question completely by showing that $EMSO(\rightarrow_p, \rhd, \leq)$ is expressively equivalent to CFSM's. In this proof she had to formulate a suitable logic of paths (a kind of propositional dynamic logic, PDL). As part of her overall proof she shows an equivalence between this logic and $FO(\rightarrow_p, \rhd, \leq)$ which is again a technically demanding contribution. A corollary of this result is that $FO(\rightarrow_p, \rhd, \leq)$ is expressively equivalent to $FO^3(\rightarrow_p, \rhd, \leq)$ over MSC's. Finally, in a single-authored paper at ICALP she shows that FO is expressively equivalent to FO^3 over a very general class of linear orders. It is very significant that she not only solves hard open problems but invents new tools to attack such problems; these are tools that can be applied to other problems as well so she has invigorated the subject with new techniques as well as new results.

Biographical sketch

Marie Fortin carried out her PhD (under the supervision of Paul Gastin and Benedikt Bollig) at ENS Paris-Saclay, Université Paris-Saclay. She was a winner of the EATCS Distinguished Dissertation Award 2021, a best paper award at CONCUR 2018 and a best student paper award at ICALP 2019 (Track B). She is currently a research associate at the University of Liverpool.

Citation for Sandra Kiefer

Sandra Kiefer shares the *2021 Ackerman Award* of the European Association of Computer Science Login (EACSL) for her thesis

Power And Limits Of The Weisfeiler-Leman Algorithm.

The Weisfeiler-Leman (WL) color refinement algorithm has long played a central role in computational graph theory and descriptive complexity theory and has come into new prominence recently via its role in machine learning over graphs. The thesis provides a plethora of new and deep insights on the power of WL. WL can be parameterized by the number of rounds required for termination and the number of pebbles required ("WL dimension"). Both parameters can be interpreted in terms of logic – the former corresponding to quantifier rank in an appropriate logic and the latter corresponding to number of variables. The thesis provides new bounds on both of these quantities, including a bound of 3 on the WL dimension for planar graphs.

The thesis is beautifully written, and makes use of a wide range of techniques; unusually for a thesis with close connections to graph theory and logic, it even makes use of empirical methods.

Background of the Thesis

The original version of the WL algorithm – now referred to as 1-dimensional WL – was introduced by Russian mathematicians Boris Weisfeiler and Andrei Leman in 1968. The algorithm evolved into its current, more general form, parameterized by a dimension k in the 1980s. It iteratively computes a colouring of the k-tuples of vertices of a graph. It can be seen in a number of lights: originally it was an approximate isomorphism test. Later it was found to be intimately connected to logic, with the distinguishing power of the WL test corresponding to the expressiveness of a canonical logic with counting. More recently it has been seen to play a fundamental rule in analysis of so-called Graph Neural Networks.

Contributions of the Thesis

Kiefer's dissertation provides a systematic analysis of the WL algorithm. It focuses on two natural parameters: the number of iterations the algorithm needs to converge, and the dimension it needs to recognize certain graph properties and identify certain graphs. For both quantities, the thesis provides new upper and lower bounds; previously, non-trivial results were only qualitative (bounded vs unbounded) or gave very rough estimates.

Notable results include bounds on the iteration number, making use of a combination of analytical and experimental methods. For the WL dimension it gives a complete classification of all graphs with WL dimension 1, along with an upper bound on dimension based on the tree width. Perhaps most importantly, the thesis provides a bound of 3 on the dimension for planar graphs.

The innovations in this thesis will be of interest to an unusually wide range of researchers, from computational graph theory to descriptive complexity theory to machine learning.

Biographical Sketch

Sandra Kiefer carried out her PhD (under the supervision of Martin Grohe and Pascal Schweitzer) at RWTH in Aachen Germany. For her project for female doctoral candidates, RWTH Aachen University awarded her the Brigitte Gilles Prize in 2019. After completing

her PhD, she took up a 1-year position as a postdoctoral research associate at the University of Warsaw, Poland. She is currently a postdoctoral research associate at RWTH Aachen University, Germany.

Jury

The jury for the **Ackermann Award 2021** consisted of eight members, two of them *ex officio*, namely, the president and the vice-president of EACSL. In addition, the jury also included a representative of SIGLOG (the ACM Special Interest Group on Logic and Computation).

The members of the jury were:

- Christel Baier (TU Dresden),
- Michael Benedikt (University of Oxford),
- Mikołaj Bojańczyk (University of Warsaw),
- Jean Goubault-Larrecq (ENS Paris-Saclay),
- Prakash Panangaden (McGill University),
- Simona Ronchi Della Rocca (University of Torino), the vice-president of EACSL,
- Thomas Schwentick (TU Dortmund University), the president of EACSL,
- Alexandra Silva, (University College London), ACM SigLog representative.

Previous winners

Previous winners of the Ackermann Award were

2005, Oxford:
Mikołaj Bojańczyk from Poland,
Konstantin Korovin from Russia, and
Nathan Segerlind from the USA.

2006, Szeged:
Balder ten Cate from the Netherlands, and
Stefan Milius from Germany.

2007, Lausanne:
Dietmar Berwanger from Germany and Romania,
Stéphane Lengrand from France, and
Ting Zhang from the People's Republic of China.

2008, Bertinoro:
Krishnendu Chatterjee from India.

2009, Coimbra:
Jakob Nordström from Sweden.

2010, Brno:
no award given.

2011, Bergen:
Benjamin Rossman from USA.

2012, Fontainebleau:
Andrew Polonsky from Ukraine, and
Szymon Toruńczyk from Poland.

2013, Turin:
Matteo Mio from Italy.

2014, Vienna:
Michael Elberfeld from Germany.

2015, Berlin:
Hugo Férée from France, and
Mickael Randour from Belgium.
2016, Marseille:
Nicolai Kraus from Germany
2017, Stockholm:
Amaury Pouly from France.
2018, Birmingham:
Amina Doumane from France.
2019, Barcelona (conference in 2020):
Antoine Mottet from France.
2020, Ljubljana (conference online in 2021)
Benjamin Kaminski from Germany.

Detailed reports on their work appeared in the CSL proceedings and are also available on the EACSL homepage.

Between Deterministic and Nondeterministic Quantitative Automata

Udi Boker
Reichman University, Herzliya, Israel

Abstract

There is a challenging trade-off between deterministic and nondeterministic automata, where the former suit various applications better, however at the cost of being exponentially larger or even less expressive. This gave birth to many notions in between determinism and nondeterminism, aiming at enjoying, sometimes, the best of both worlds. Some of the notions are yes/no ones, for example initial nondeterminism (restricting nondeterminism to allowing several initial states), and some provide a measure of nondeterminism, for example the ambiguity level.

We analyze the possible generalization of such notions from Boolean to quantitative automata, and suggest that it depends on the following key characteristics of the considered notion N– whether it is syntactic or semantic, and if semantic, whether it is word-based or language-based.

A syntactic notion, such as initial nondeterminism, applies as is to a quantitative automaton $\mathcal{A}$, namely $\mathsf{N}(\mathcal{A})$. A word-based semantic notion, such as unambiguity, applies as is to a Boolean automaton t-$\mathcal{A}$ that is derived from $\mathcal{A}$ by accompanying it with some threshold value $t \in \mathbb{R}$, namely $\mathsf{N}(t\text{-}\mathcal{A})$. A language-based notion, such as history determinism, also applies as is to t-$\mathcal{A}$, while in addition, it naturally generalizes into two different notions with respect to $\mathcal{A}$ itself, by either: i) taking the supremum of $\mathsf{N}(t\text{-}\mathcal{A})$ over all thresholds t, denoted by $\mathsf{Threshold\text{-}N}(\mathcal{A})$; or ii) generalizing the basis of the notion from a language to a function, denoted simply by $\mathsf{N}(\mathcal{A})$. While in general $\mathsf{N}(\mathcal{A}) \Rightarrow \mathsf{Threshold\text{-}N}(\mathcal{A}) \Rightarrow \mathsf{N}(t\text{-}\mathcal{A})$, we have for some notions $\mathsf{N}(\mathcal{A}) \equiv \mathsf{Threshold\text{-}N}(\mathcal{A})$, and for some not. (For measure notions, $\Rightarrow$ stands for $\geq$ with respect to the nondeterminism level.)

We classify numerous notions known in the Boolean setting according to their characterization above, generalize them to the quantitative setting and look into relations between them. The generalized notions open new research directions with respect to quantitative automata, and provide insights on the original notions with respect to Boolean automata.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Quantitative Automata, Measure of Nondeterminism, Determinism

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.1

Category Invited Talk

Acknowledgements We thank Karoliina Lehtinen for stimulating discussions on prelimiary versions of the paper, and specifically for suggesting the all-thresholds generalization of the notions.

1 Introduction

The tension between determinism and nondeterminism is at the core of computer science, most notably evident in the P vs. NP epic. In automata theory, determinism often suits certain applications, such as synthesis, better and allows for efficient decision algorithms, while nondeterminism allows for exponential succinctness, and sometimes higher expressiveness.

As a result, there is intensive research in automata theory on notions in between determinism and nondeterminism, starting in the 1970s [41, 40], and actively continuing until these days. (See, for example, the recent breakthrough on a superpolynomial lower bound for the state complexity involved in complementing unambiguous finite automata [50, 30]). Numerous such notions have developed over the years, many of which we classify in Section 3.

Some of them are yes/no ones, for example *determinizability by pruning*, and some provide a measure of nondeterminism, for example *deviation*. Deterministic automata have in the former case value "yes", and in the latter case value 0 or 1, depending on the notion.

We shall discuss the notions with respect to automata on finite and infinite words, while they often apply also to automata over richer input structures, such as trees and graphs. Further, while the notions are usually considered with respect to nondeterministic automata, they often also apply to alternating automata, setting limitations on both nondeterministic and universal transitions. (See, for example, [20, 13, 39])

Over the past decade, there is a growing interest in quantitative verification and synthesis (e.g., [2, 9, 24, 18, 25, 6, 15, 3, 36, 27]), and in *quantitative*[1] *automata* [17] as their underlying computational model, for addressing the requirements of contemporary systems – the traditional Boolean perspective of verification and synthesis falls short of many aspects of contemporary systems, concerning performance, robustness, and resource-constraints, according to which one system is often preferred over another, even though they are both correct, since one is, for example, faster than the other, or, if they are both incorrect, one misbehaves less frequently than the other.

Since quantitative automata keep the *choice* meaning of nondeterminism, notions of limited nondeterminism can be naturally generalized from Boolean to quantitative automata, which is indeed starting to take place in recent years [19, 4, 36, 26, 14]. In [14], the current author and Karoliina Lehtinen have considered three such notions – *determinizability by pruning*, *history determinism*, and *good-for-gameness* – and suggested to look into three different generalizations of each of them to the setting of quantitative automata.

In this paper, we look into the question of how to generalize to the quantitative setting the numerous limited-nondeterminism notions that exist for Boolean automata. We suggest that it depends on the following key characteristics of the considered notion N – whether it is syntactic or semantic, and if semantic, whether it is word-based or language-based.

A syntactic notion (Sections 3.1 and 4.1) applies as is to a quantitative automaton $\mathcal{A}$, namely $\mathsf{N}(\mathcal{A})$ – it only relates to the automaton's structure, indifferent to whether the semantics of the automaton is based on a Boolean acceptance condition or a value function. Among these notions are initial nondeterminism, tree width, structural ambiguity, and determinism in the limit.

A word-based semantic notion (Sections 3.2 and 4.2) applies as is to a Boolean automaton $t\text{-}\mathcal{A}$ that is derived from the quantitative automaton $\mathcal{A}$ by accompanying it with some threshold value $t \in \mathbb{R}$, namely $\mathsf{N}(t\text{-}\mathcal{A})$. Among these notions are ambiguity, branching, guessing, amount of nondeterminism, trace, and deviation.

A language-based notion (Sections 3.3 and 4.3) allows for the three natural generalizations suggested in [14]: i) applying it as is to $t\text{-}\mathcal{A}$; ii) applying it to $\mathcal{A}$ itself by taking the supremum of $\mathsf{N}(t\text{-}\mathcal{A})$ over all thresholds t, denoted by Threshold-$\mathsf{N}(\mathcal{A})$; and iii) applying it to $\mathcal{A}$ itself by generalizing the language basis of the notion to a function basis, denoted simply by $\mathsf{N}(\mathcal{A})$.

Interestingly, while in general $\mathsf{N}(\mathcal{A}) \Rightarrow$ Threshold-$\mathsf{N}(\mathcal{A}) \Rightarrow \mathsf{N}(t\text{-}\mathcal{A})$, it might be that $\mathsf{N}(\mathcal{A}) \not\Leftarrow$ Threshold-$\mathsf{N}(\mathcal{A})$, even though Threshold-$\mathsf{N}(\mathcal{A})$ is defined with respect to *all* thresholds $t \in \mathbb{R}$, as is the case with history determinism (see Section 4.3.1). (For measure notions, $\Rightarrow$ stands for $\geq$ with respect to the nondeterminism level.)

[1] Quantitative automata generally suit verification needs better than weighted automata, since they keep the *choice* meaning of nondeterminism and realize functions to the totally-ordered domain of real numbers. (See Section 2 for the definition of nondeterministic quantitative automata and [8] for an analysis of the differences between quantitative and weighted automata.)

Among the language-based notions are residuality, history determinism, good-for-gameness, good-for-MDPness, determinizability by pruning, and k-tokens games.

We provide in Section 3 partially-formal definitions of numerous notions known in the Boolean setting, classified according to the above characterization. In Section 4 we analyze their generalization to the quantitative setting, and in Section 5 we look into their interrelations, observing that they refine the interrelations known for the Boolean setting.

The notion generalization provides novel definitions of automata classes, and opens up many research directions on limited nondeterminism in each of the various quantitative automata types (limit-average automata, discounted-sum automata, etc.): What advantages of determinism are preserved (complexity of decision problems, suitability for various applications, etc.)? What advantages of nondeterminism are preserved (expressiveness, succinctness, etc.)? What are the closure properties of the resulting class of automata? What is the complexity of its membership check? etc.

Furthermore, the generalized notions provide insights on the original notions with respect to Boolean automata. For example, the refined notion interrelations, whereby history determinism and good-for-gameness are not equivalent, suggests that the two notions, which are known to be equivalent with respect to ω-regular automata, might not be equivalent with respect to other types of Boolean automata. Another example is the implication between history determinism and pre-residuality, which is known to hold for ω-regular automata, but turns out to depend on the suffix monotonicity of the acceptance condition.

2 Preliminaries

Words. An *alphabet* Σ is a finite nonempty set of letters. A finite (resp. infinite) *word* $u = \sigma_0 \dots \sigma_k \in \Sigma^*$ (resp. $w = \sigma_0\sigma_1 \dots \in \Sigma^\omega$) is a finite (resp. infinite) sequence of letters from Σ. A *language* is a set of words.

Boolean automata. Many notions of limited nondeterminism apply to various types of Boolean automata, namely to automata that define a *language* by accepting or rejecting words, such as finite automata on finite words (NFAs), pushdown automata, ω-regular automata, two-way automata, and more.

In our partially-formal presentation of the notions, we relate to a nondeterministic Boolean automaton as a structure $\mathcal{A} = (\Sigma, Q, I, \delta, \alpha)$, where Σ is an alphabet; Q is a finite nonempty set of states; $I \subseteq Q$ is a set of initial states; $\delta\colon Q \times \Sigma \to 2^Q$ is a transition function, and α is an acceptance condition. This basic structure can be extended for various automata types.

We denote by $\mathcal{A}^q$ the automaton that is derived from $\mathcal{A}$ by changing its initial set of states to be $\{q\}$. An automaton is deterministic if I is a singleton, and for every state q and letter σ, we have $|\delta(q, \sigma)| \leq 1$.

A run of the automaton is a path over its states, starting with an initial state and continuing along the transition function. A run is accepting if it satisfies the acceptance condition; a word is accepted by $\mathcal{A}$ if there is an accepting run on it; and the language that $\mathcal{A}$ recognizes, denoted by $L(\mathcal{A})$, is the set of words that it accepts. Two automata $\mathcal{A}$ and $\mathcal{A}'$ are equivalent, denoted by $\mathcal{A} \equiv \mathcal{A}'$, if they recognize the same language.

In the case of NFAs, which define regular languages, the acceptance condition α is a subset of Q, and a run is accepting if it ends in a state in α. In the case of Büchi automata, which define ω-regular languages, α is also a subset of Q, and a run is accepting if it visits a state in α infinitely often. (More on acceptance conditions of ω-regular automata can be found, for example, in [7].)

Quantitative Automata. A *nondeterministic quantitative*[2] *automaton* on words is a tuple $\mathcal{A} = (\Sigma, Q, I, \delta)$, where Σ is an alphabet; Q is a finite nonempty set of states; $I \subseteq Q$ is a set of initial states; and $\delta\colon Q \times \Sigma \to 2^{(\mathbb{R}\times Q)}$ is a transition function over weight-state pairs.

A *transition* is a tuple $(q, \sigma, x, q') \in Q\times\Sigma\times\mathbb{R}\times Q$, also written $q \xrightarrow{\sigma:x} q'$. (Note that there might be several transitions with different weights over the same letter between the same pair of states[3].) We write $\gamma(t) = x$ for the weight of a transition $t = (q, \sigma, x, q')$.

We require that the automaton $\mathcal{A}$ is *complete*, namely that for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at least one state q' and a transition $q \xrightarrow{\sigma:x} q'$.

A *run* of the automaton on a word w is a sequence $\rho = q_0 \xrightarrow{w[0]:x_0} q_1 \xrightarrow{w[1]:x_1} q_2 \dots$ of transitions where $q_0 \in I$ and $q_{i+1} \in \delta(q_i, w[i])$. As each transition t_i carries a weight $\gamma(t_i) \in \mathbb{R}$, the sequence ρ provides a weight sequence $\gamma(\pi) = \gamma(t_0)\gamma(t_1)\gamma(t_2)\dots$.

A Val automaton (for example a Sum automaton) is one equipped with a *value function* $\mathsf{Val} : \mathbb{R}^* \to \mathbb{R}$ or $\mathsf{Val} : \mathbb{R}^\omega \to \mathbb{R}$, which assigns real values to runs of $\mathcal{A}$. The value of a run π is $\mathsf{Val}(\gamma(\pi))$. The value of $\mathcal{A}$ on a word w is the supremum of $\mathsf{Val}(\pi)$ over all runs π of $\mathcal{A}$ on w.

Automata $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent*, denoted by $\mathcal{A} \equiv \mathcal{A}'$, if they realize the same function.

Value functions. We list below some of the common value functions used for quantitative automata on finite/infinite words, defined over sequences of real weights.

For finite sequences $v = v_0 v_1 \dots v_{n-1}$:

- $\mathsf{Sum}(v) = \sum_{i=0}^{n-1} v_i$
- $\mathsf{Avg}(v) = \frac{1}{n}\sum_{i=0}^{n-1} v_i$
- $\mathsf{Prod}(v) = \prod_{i=0}^{n-1} v_i$

For finite and infinite sequences $v = v_0 v_1 \dots$:

- $\mathsf{Inf}(v) = \inf\{v_n \mid n \geq 0\}$
- $\mathsf{Sup}(v) = \sup\{v_n \mid n \geq 0\}$
- For a discount factor $\lambda \in \mathbb{Q} \cap (0,1)$, $\lambda\text{-}\mathsf{DSum}(v) = \sum_{i\geq 0} \lambda^i v_i$

For infinite sequences $v = v_0 v_1 \dots$:

- $\mathsf{LimInf}(v) = \lim_{n\to\infty} \inf\{v_i \mid i \geq n\}$
- $\mathsf{LimSup}(v) = \lim_{n\to\infty} \sup\{v_i \mid i \geq n\}$
- $\mathsf{LimInfAvg}(v) = \mathsf{LimInf}(\mathsf{Avg}(v_0), \mathsf{Avg}(v_0, v_1), \mathsf{Avg}(v_0, v_1, v_2), \dots)$
- $\mathsf{LimSupAvg}(v) = \mathsf{LimSup}(\mathsf{Avg}(v_0), \mathsf{Avg}(v_0, v_1), \mathsf{Avg}(v_0, v_1, v_2), \dots)$

($\mathsf{LimInfAvg}$ and $\mathsf{LimSupAvg}$ are also called $\underline{\mathsf{MeanPayoff}}$ and $\overline{\mathsf{MeanPayoff}}$.)

3 Notion Classification

We consider notions that limit the nondeterminism of a Boolean automaton $\mathcal{A}$. The notions may either be yes/no ones (always having "yes" for deterministic automata) or provide a measure of nondeterminism, having a fixed value $k \in \mathbb{N}$, dependency on $\mathcal{A}$'s size (e.g., polynomial), *finite*, or *infinite* value (and always have value 0 or 1 for deterministic automata).

[2] We speak of "quantitative" rather than "weighted" automata, following the distinction made in [8] between the two.

[3] This extra flexibility of "parallel" transitions with different weights is often omitted (e.g., in [16]) since it is redundant for some value functions while important for others.

We classify the notions into syntactic and semantic ones, then classify the semantic ones into word-based and language-based, and finally further classify the language-based ones.

Due to the extent of the different notions, we define them only partially formally.

3.1 Syntactic Notions

These notions only relate to the structure of $\mathcal{A}$, unrelated to the language it recognizes.

- Initially nondeterministic($\mathcal{A}$) holds if $\mathcal{A}$ is deterministic, except for possibly having several initial states. Considering NFAs, such an automaton is called "deterministic finite automaton with multiple initial states" (MDFA). See [37, 33]. (Such a Büchi automaton is called in [52, Section 2] "semi deterministic", however this name is currently more in use for a "deterministic in the limit" automaton, as defined below.)
- Tree width$(\mathcal{A}, w)$ = number of runs of $\mathcal{A}$ on a word w; Tree width$(\mathcal{A})$ = sup{tree width$(\mathcal{A}, w)$} over all words w. It is also called leaf size. See [35].
- Structural ambiguity$(\mathcal{A}) = k$ if there is a single initial state q_0 in $\mathcal{A}$, and for every word w and state q of $\mathcal{A}$, there are at most k runs of $\mathcal{A}$ over w that end in q. When $k = 1$, $\mathcal{A}$ is structurally unambiguous. See [45].
- General structural ambiguity$(\mathcal{A}) = k$ if for every word w and states q and q' of $\mathcal{A}$, there are at most k paths from q to q' over w. When $k = 1$, $\mathcal{A}$ is generally structurally unambiguous. See [45].

We also consider in this class notions that do not relate to $\mathcal{A}$'s language, but do relate to its state labeling or transition labeling.

- Deterministic in the limit$(\mathcal{A})$ holds if $\mathcal{A}^q$ is deterministic for every accepting state q of $\mathcal{A}$ (considering only the transitions and states reachable from q). See [23, page 34]. It is also called limit determinism and semi determinism.

3.2 Semantic Word-Based Notions

A notion N in this class is derived from a word measure $\mathsf{N}(\mathcal{A}, w)$, by setting its value for an automaton $\mathcal{A}$ to be $\mathsf{N}(\mathcal{A}) = \sup\{\mathsf{N}(\mathcal{A}, w) \mid w \in L(\mathcal{A})\}$, namely the supremum over all words that the automaton *accepts*. (When it is clear from the context what $\mathcal{A}$ is, we write $\mathsf{N}(w)$ instead of $\mathsf{N}(\mathcal{A}, w)$.)

The most common such notion is of *ambiguity*, defined by

- Ambiguity(w) = the number of accepting runs of $\mathcal{A}$ on w. When ambiguity$(\mathcal{A}) = 1$, $\mathcal{A}$ is said to be unambiguous. See a survey in [21].

There are various additional notions in the class, whose measure is based on *aggregating the nondeterminism along runs*. They differ by the aggregation function, and by whether they consider all runs, the best run, or the worst run on the word w.

- Amount of nondeterminism(w) = the minimal number of nondeterministic choices that occur in an accepting run of $\mathcal{A}$ on w. See [42].
- Branching(w) = the minimum branching of an accepting run of $\mathcal{A}$ on w, where branching of a run r is the product of the nodes' degrees along r in the computation tree of $\mathcal{A}$ on w. See [29].
- Guessing$(w) = \log($branching$(w))$, namely summing up the logarithm of the nodes' degrees rather than multiplying the degrees. See [29].
- Trace(w) = the maximum branching of an accepting run of $\mathcal{A}$ on w. See [49].
- Deviation(w) is almost the same as guessing(w), just considering for each node its degree minus one rather than its degree. See [28].

3.3 Semantic Language-Based Notions

Notions within this class are not based on a word measure, but rather relate, explicitly or implicitly, to language equality. We further classify them into the following subclasses.

3.3.1 Determinism Embedding

A nondeterministic such automaton embeds an equivalent deterministic automaton, which can be derived from it by just pruning transitions or by more complicated manipulations, such as merging and pruning. A well known example of merging is the standard minimization procedure of DFAs, merging states within the same Myhill Nerode equivalence class.

Consider automata $\mathcal{A} = (\Sigma, Q, I, \delta, \alpha)$ and $\mathcal{A}' = (\Sigma, Q', I', \delta', \alpha')$. We say that $\mathcal{A}'$ is derived from $\mathcal{A}$ by *pruning* if $Q' \subseteq Q$, $I' \subseteq I$ and $\delta' \subseteq \delta$. We further say that $\mathcal{A}'$ is derived from $\mathcal{A}$ by *merging and pruning* if $Q' \subseteq Q$ and there exists a partition $P \subseteq 2^Q$ of Q and a mapping $\mu : P \to Q$, such that i) for every $S \in P, \mu(S) \in S$; and ii) for every transition $p' \xrightarrow{\sigma:x} q' \in \delta'$ we have states $p \in \mu^{-1}(p')$ and $q \in \mu^{-1}(q')$, such that there is a transition $p \xrightarrow{\sigma:x} q \in \delta$. (We've already added transition weights, which can be ignored when irrelevant.)

- Determinizable by pruning($\mathcal{A}$) holds if there exists an automaton $\mathcal{A}'$ that is equivalent to $\mathcal{A}$ and derived from it by pruning. See [4, 10].
- Determinizable by merging and pruning($\mathcal{A}$) holds if there exists an automaton $\mathcal{A}'$ that is equivalent to $\mathcal{A}$ and derived from it by merging and pruning. See [47, Section 2.1].

3.3.2 Language Similarity between States

The most common such notion requires all target states of transitions from the same state over the same letter to have the same residual language. Pre-residuality loosens it by requiring the existence of a target state whose residual language contains the languages of all other target states.

- Residual($\mathcal{A}$) holds if for every state q, letter σ, and states $q', q'' \in \delta(q, \sigma)$, we have $L(\mathcal{A}^{q'}) = L(A^{q''})$. See [5, Definition 15]. It is called semantic determinism in [1].
- Pre-residual($\mathcal{A}$) holds if $\mathcal{A}$ can be pruned into an equivalent residual automaton. See [5, Definition 15].

3.3.3 Restricted Choice Function

A nondeterministic automaton has, in each of its choices, "unlimited view" of the entire history of the word prefix read so far and of the entire future of the word suffix to be read. Two natural restrictions are to only allow a (partial) view of the history or to only allow a (partial) view of the future. An automaton is *history deterministic* if it can only view the history of the word, and it has k *lookahead* if it can only see the next k letters. History determinism is refined or made coarse by two orthogonal measures, the first considering the size of the memory required to store the relevant part of the history, and the second allowing to maintain several runs, one of which should always be correct.

- History deterministic($\mathcal{A}$) holds if there exists a strategy $s : Q \times \Sigma^* \to \delta$ to resolve $\mathcal{A}$'s nondeterminism. See [19, 13]. It is formally defined via a *letter game* played on $\mathcal{A}$ (see Section 3.3.4), whereby $\mathcal{A}$ is history deterministic if Eve has a winning strategy in it.
- History deterministic with memory M($\mathcal{A}$) holds if there exists a history-determinism strategy for $\mathcal{A}$ that uses memory M, where M can be a fixed size, dependent on the size of $\mathcal{A}$, *finite*, or *infinite*. Notice that when $M = 0$, $\mathcal{A}$ is determinizable by pruning (see Section 3.3.1).

- History determinism$(\mathcal{A}) = k$ if there exist k strategies that can be used in parallel to resolve $\mathcal{A}$'s nondeterminism. (When $k = 1$, $\mathcal{A}$ is history deterministic.) It is formally defined via a *k-runs letter game*, in which Eve should have a winning strategy (see Section 3.3.4). The notion is originally defined in [48], where it is called the width of $\mathcal{A}$.
- Lookahead$(\mathcal{A}) = k$ if there exists a strategy $s : Q \times \Sigma^{\leq k+1} \to \delta$ to resolve $\mathcal{A}$'s nondeterminism, where $\Sigma^{\leq k+1}$ stands for the current and next up to k letters of the word read. See [51, 46].

3.3.4 Winning Letter Games on $\mathcal{A}$

The *letter game* formalizes the history determinism notion (see Section 3.3.3). In the game, Adam produces a word w letter by letter, and Eve should resolve the nondeterminism. Eve wins a play if her run is accepting or $w \notin L(\mathcal{A})$. Then, $\mathcal{A}$ is history deterministic if Eve has a winning strategy in the letter game. In the *k-runs letter game*, Eve has k runs to maintain, one of which should be accepting if $w \in L(\mathcal{A})$.

Notice that resolving the winner of a letter game, for deciding whether or not a given automaton is history deterministic, is generally a difficult task, due to the complicated winning condition of the game – it depends on w's membership in $L(\mathcal{A})$, which relates to all (possibly uncountably many) runs of $\mathcal{A}$ on w. In an attempt to overcome this difficulty, Bangol and Kuperberg defined in [5] the *k-token game*, which replaces the $w \in L(\mathcal{A})$ part of the winning condition by a requirement from Adam to provide an evidence for the membership of w in $\mathcal{A}$ – Adam should also resolve the nondeterminism along k runs, and Eve wins a play if her run is accepting or all of Adam's runs are rejecting.

If Eve wins the letter game on $\mathcal{A}$ then she obviously wins the k-tokens game on $\mathcal{A}$ for any k. As for the converse, the 1-token game is generally easier for Eve than the letter game [5], but it is open whether or not the 2-tokens game is equivalent to the letter game, known as the "G2 conjecture" [5, Conclusions]. It was proved correct so far for Büchi and coBüchi automata [5, 11], and it is still open for stronger acceptance conditions.

There are various variants of the k-tokens game in the literature, depending on how exactly Adam is allowed to conduct his runs, one of which is the *joker game* [43].

- Letter game$(\mathcal{A})$: In each round of a play, Adam chooses a letter $\sigma \in \Sigma$ and then Eve chooses a corresponding transition $t \in \delta$. In the limit, a play consists of a word w generated by Adam and a run r on w generated by Eve. Eve wins if r is accepting or $w \notin L(\mathcal{A})$. See [32, 13].
- k-runs letter game$(\mathcal{A})$: As the letter game, except that Eve maintains k runs, by choosing at each round k corresponding transitions, and she wins if at least one of her runs is accepting or $w \notin L(\mathcal{A})$. See [48].
- k-tokens game$(\mathcal{A})$: In each round of a play, Adam chooses a letter $\sigma \in \Sigma$, then Eve chooses a corresponding transition $t \in \delta$, and then Adam chooses k corresponding transitions. In the limit, a play consists of a word w generated by Adam, a run r on w generated by Eve, and k runs on w generated by Adam. Eve wins if r is accepting or all of Adam's runs are rejecting. See [5].

3.3.5 Good For (composition with) Entities

An automaton $\mathcal{A}$ is intuitively good for some entities, e.g., games or Markov decision processes (MDPs), if the composition of $\mathcal{A}$ with every entity E whose definition is based on $\mathcal{A}$'s language yields an entity $E \times \mathcal{A}$ that is equivalent to E.

- Good for games($\mathcal{A}$) holds if for every game G with Σ-labeled transitions and winning condition $L(\mathcal{A})$, we have that G and $G \times \mathcal{A}$ have the same winner. See [32, 13].
- Good for small (n) games($\mathcal{A}$) holds if $\mathcal{A}$ is good for games with up to n positions. See [22, Definition 8] and [44].
- Good for trees($\mathcal{A}$) is equivalent to $\mathcal{A}$ being good for one-player games [13, Lemma 15]. See [10, Section 2.3] for the original definition.
- Good for automata($\mathcal{A}$) holds if for every alternating automaton $\mathcal{B}$ with Σ-labeled transitions (or states) and acceptance condition $L(\mathcal{A})$, we have $\mathcal{B} \times \mathcal{A}$ is equivalent to $\mathcal{B}$. See [20] and [13, Definition 6].
- Good for MDPs($\mathcal{A}$) holds if for every MDP M with Σ-labeled transitions and the objective that its runs generate words in $L(\mathcal{A})$, we have that M and $M \times \mathcal{A}$ have the same value to their optimal strategies. See [31, Definition 1].

4 Generalizing the Notions to the Quantitative Setting

We follow below the structure of Section 3, which classified the different notions, and explain how the notions of each class can be generalized to the quantitative setting. When relevant, we provide the (partially formal) definition of each of the considered generalized notions.

4.1 Syntactic Notions

Notions unrelated to the automaton's language and acceptance labeling, as described in Section 3.1, apply as is to quantitative automata – they only relate to the automaton's structure, which is indifferent to whether the semantics of the automaton relates to Boolean or quantitative values. This is the case, in particular, with initial nondeterminsm, tree width, structural ambiguity, and general structural ambiguity.

As for determinisim in the limit, which does not relate to a language, but does relate to labeling of states or transitions, it can analogously apply to quantitative automata, requiring for example that an automaton is deterministic after every transition with some designated weight. It might be relevant only to quantitative automata of certain types, as it is only relevant to Büchi automata among the standard ω-regular automata.

4.2 Semantic Word-Based Notions

A quantitative automaton $\mathcal{A}$ can be naturally turned into a Boolean one, by accompanying it with a threshold value $t \in \mathbb{R}$, and defining that a word w is accepted by the *threshold automaton* t-$\mathcal{A}$ iff $\mathcal{A}(w) \geq t$ (or $\mathcal{A}(w) > t$).

Semantic word-based notions, as described in Section 3.2, measure for each *accepted* word the level of nondeterminism in its runs. Since these measures only consider the nondeterminism involved in the runs, they apply as is to threshold automata – Once an acceptance criterion is set, the semantic word-based measures are analogous to the syntactic measures considered in Section 3.1.

Looking for generalized notions that apply to $\mathcal{A}$ itself, and not only to t-$\mathcal{A}$, as will be defined in Section 4.3 for language-based notions, we currently do not see natural such generalizations: i) Taking the supremum of the notion's value on t-$\mathcal{A}$ over all thresholds $t \in \mathbb{R}$ will eliminate the semantic meaning of the notion, making it just a syntactic notion as in Section 3.1. For example, ambiguity will be the same as tree width. ii) Replacing the Boolean acceptance criterion with some function that relates to the value of the automaton on a word might be an interesting notion for some types of quantitative automata, but it changes the original meaning of the notion and does not seem to provide a general notion between determinism and nondeterminism.

4.3 Semantic Language-Based Notions

A notion N that is based, explicitly or implicitly, on language equality, as described in Section 3.3, can be naturally generalized to the quantitative setting in the following three ways.

Approach I. $\mathsf{N}(t\text{-}\mathcal{A})$. Applying the notion as is to a threshold automaton t-$\mathcal{A}$, as described in Section 4.2 above for word-based notions.

Approach II. Threshold-$\mathsf{N}(\mathcal{A})$. Defining the notion's value on $\mathcal{A}$ to be the worst (supremum) value of $\mathsf{N}(t\text{-}\mathcal{A})$ over all thresholds $t \in \mathbb{R}$.

Approach III. $\mathsf{N}(\mathcal{A})$. Directly generalizing the notion to apply to $\mathcal{A}$ by replacing the language basis of N with a function basis. For example, rather then requiring that $\mathcal{A}$ and some related automaton $\mathcal{A}'$ recognize the same language, we require that the quantitative automata $\mathcal{A}$ and $\mathcal{A}'$ realize the same function.

By definition, if Threshold-$\mathsf{N}(\mathcal{A})$ holds for some automaton $\mathcal{A}$ than $\mathsf{N}(t\text{-}\mathcal{A})$ holds with respect to every threshold $t \in \mathbb{R}$. (For a notion that provides a nondeterminism measure rather than a yes/no value, we have Threshold-$\mathsf{N}(\mathcal{A}) \geq \mathsf{N}(t\text{-}\mathcal{A})$).

We also generally have $\mathsf{N}(\mathcal{A}) \Rightarrow$ Threshold-$\mathsf{N}(\mathcal{A})$ (or $\mathsf{N}(\mathcal{A}) \geq$ Threshold-$\mathsf{N}(\mathcal{A})$), since morally the former requires exact equality and the latter requires equality with respect to thresholds. Interestingly, it might be that $\mathsf{N}(\mathcal{A}) \not\Leftarrow$ Threshold-$\mathsf{N}(\mathcal{A})$, even though Threshold-$\mathsf{N}(\mathcal{A})$ is defined with respect to *all* thresholds $t \in \mathbb{R}$, as is the case with history determinism (see Section 4.3.1).

Notice that when speaking of $\mathsf{N}(t\text{-}\mathcal{A})$, we consider the same definition of N as in the Boolean case. Further, the definition of Threshold-N is directly derived from it. Yet, the definition of N in $\mathsf{N}(\mathcal{A})$, for a quantitative automaton $\mathcal{A}$, is more general than its definition for a Boolean automaton, as its basis is generalized from languages to functions.

For each notion N listed in Section 3.3, we provide below its generalization $\mathsf{N}(\mathcal{A})$ to a quantitative automaton $\mathcal{A}$. Observe that if N is explicitly based on automata equivalence or containment, then it can be directly generalized to quantitative automata, in which setting automata equivalence stands for realizing the same function and an automaton $\mathcal{A}'$ is contained in $\mathcal{A}$ if for every word w, $\mathcal{A}'(w) \leq \mathcal{A}(w)$. However, if N is only implicitly based on language equality, its generalization requires some subtlety.

Determinism Embedding. Such notions, and in particular determinizability by pruning and determinizability by merging and pruning, consider the *equivalence* of $\mathcal{A}$ and an embedded automaton $\mathcal{A}'$, and are thus directly generalized to quantitative automata.

Language Similarity between States. These notions, and in particular residuality and pre-residuality also directly speak of automata equivalence or containment, and can thus directly apply to quantitative automata. Yet, it is natural to extend them in two aspects:

I. Since in quantitative automata there are weights on transitions, the first transition can make a difference and should be considered. Thus, the definition should be that $\mathsf{Residual}(\mathcal{A})$ holds, for a Val automaton $\mathcal{A}$, if for every state q, letter σ, and transition $t = q \xrightarrow{\sigma:x} q'$, we have that $\sup\{\mathsf{Val}(\pi) \mid \pi \text{ is a run of } \mathcal{A}^q \text{ on } w \text{ starting with } t\} = \mathcal{A}^q(w)$.

II. The intended meaning of residual transitions is that it is "safe" to follow them. However, while this is the case when the value function only depends on the future, like in ω-regular automata, or when it is monotonic with respect to suffixes (cf. "suffix-monotonic functions" [14, Definition 18]), it need not be the case with general value functions. (See more about it in Section 5.) For making the notion relevant also to more general value functions, one can define transitions to be "safe" if taking them allows to get the best value to every suffix, following *every prefix* (cf. "cautious strategies" [14, Definition 14]).

Restricted choice function. These notions, and in particular lookahead and the different variants of history determinism, put limitations on the transition function of $\mathcal{A}$, which stand equally for Boolean and quantitative automata, and require the automaton $\mathcal{A}'$ that is obtained from $\mathcal{A}$ by these limitations to be equivalent to $\mathcal{A}$, which again directly generalizes to quantitative automata.

Notice that history determinism can still be formally defined via the letter game, whose generalization to the quantitative setting is described below.

Winning Letter Games on $\mathcal{A}$. These notions do not explicitly speak of language equality, and thus need a bit more delicate handling. The games are played exactly as in the Boolean setting and the only change is in the definition of Eve's winning condition – In the letter game, instead of requiring her run to accept if the generated word w is in $L(\mathcal{A})$, we require that the *value* of her run on w is exactly $\mathcal{A}(w)$; In the k-tokens game, instead of requiring her run to accept if some of Adam's runs are accepting, we require that the *value* of her run is at least as high as all of Adam's runs. See [14, Definition 2] (where the notions are formally defined also with respect to alternating automata).

Good For (composition with) Entities. These notions, differently from all of the previously discussed notions, consider also external entities – A "good for some entities" notion N holds for a Boolean automaton $\mathcal{A}$ if $\mathcal{A}$ properly composes with every entity whose definition is based on $\mathcal{A}$'s language.

Hence, for generalizing N to apply to quantitative automata, which realize functions from words to real numbers, we should first see if there is a relevant generalization of the external entities to be based on functions to real numbers rather than on languages.

- Good-for-gameness. The generalization of the notion relates to composition with *zero-sum games*, which generalize the win-lose games used for Boolean automata. A quantitative automaton $\mathcal{A}$ is good for games iff for every *zero-sum* game G whose transitions are Σ-labeled and the *payoff* (value) of a play generating a word w is $\mathcal{A}(w)$, we have that G and $G \times \mathcal{A}$ have the same *value*. See [14, Definition 1].
- Good-for-automataness. The generalization of the notion relates to composition with *alternating quantitative automata*, which generalize alternating Boolean ones.
  Notice that in the composition $\mathcal{B} \times \mathcal{A}$ of (Boolean or quantitative) automata $\mathcal{A}$ and $\mathcal{B}$, the transitions or states of $\mathcal{B}$ should be labeled with the alphabet of $\mathcal{A}$. (See [20] and [13, Definition 2]). A quantitative automaton $\mathcal{B}$ formally has transition labels (weights) in $\mathbb{R}$ (usually in $\mathbb{Q}$), implying that the alphabet Σ of $\mathcal{A}$ should be contained in $\mathbb{R}$. Yet, as there are finitely many weight labels in $\mathcal{B}$, we may consider an arbitrary alphabet Σ for $\mathcal{A}$, where each letter $\sigma \in \Sigma$ stands for a "name" to a weight from $\mathcal{B}$'s labeling.
  Then, a quantitative automaton $\mathcal{A}$ is good for automata iff for every alternating quantitative automaton $\mathcal{B}$ with Σ-labeled transitions (or states) and *value function* $\mathcal{A}$, we have that $\mathcal{B} \times \mathcal{A}$ is equivalent to $\mathcal{B}$.
- Good-for-MDPness. The generalization is with respect to MDPs whose objective is to maximize $\mathcal{A}$'s value on the generated word; that is, instead of having an objective function "$f(w) = 1$ if $w \in L(\mathcal{A})$ and 0 otherwise", the MDP has the objective function "$f(w) = \mathcal{A}(w)$". Then, a quantitative automaton $\mathcal{A}$ is good for MDPs iff for every MDP M with Σ-labeled transitions and the objective function $\mathcal{A}$, we have that M and $M \times \mathcal{A}$ have the same value to their optimal strategies. (In the product $M \times \mathcal{A}$ with a Val automaton $\mathcal{A}$, the transitions are labeled with the weight of the corresponding transition in $\mathcal{A}$, and the objective of the MDP is to maximize $\mathsf{Val}(r)$, where r is the sequence of weights generated along a run of the MDP.)

4.3.1 Notions vs. Threshold-Notions

While generally we have Notion $\Rightarrow$ Threshold-Notion for the language-based notions, it depends on the notion whether or not there is also a converse implication.

When Notions $\equiv$ Threshold-Notions. This is the case with notions that are "based on direct values", for example residuality: If residual($\mathcal{A}$) does not hold, there is a transition $t = q \xrightarrow{\sigma:x} q'$ and a word w, such that $v_1 = \sup\{\mathsf{Val}(\pi) \mid \pi \text{ is a run of } \mathcal{A}^q \text{ on } w \text{ starting with } t\} < \mathcal{A}^q(w) = v_2$. Then the threshold automaton t-$\mathcal{A}$, for $t = (v_1 - v_2)/2$, is not residual.

Notice that this is, however, not the case with pre-residuality, which is not based on direct values, as it allows for a pruning phase.

Good-for-some-entities notions can also have equivalence between the two generalized versions of the notion, since intuitively if an automaton $\mathcal{A}$ is not good for some entities then there is some specific entity E having a specific value v with respect to which $\mathcal{A}$ is not good, implying that t-$\mathcal{A}$, with $t = v$, is also not good for E. This is indeed the case with good-for-gamenesss [14, Lemma 5] and good-for-automataness (see Section 5).

When Notions $\not\equiv$ Threshold-Notions. The two variants are generally different for notions that are based on strategies, since Notion requires the same strategy for all values, while Threshold-Notion allows for a different strategy to each threshold.

It is shown in [14, Lemma 10] and demonstrated in [14, Figure 3], which we repeat in Figure 1, to be the case for history determinism and determinizability by pruning, and the same reasoning applies also to determinizability by merging and pruning, pre-residuality, lookahead, letter game, k-runs letter game, and k-tokens game.

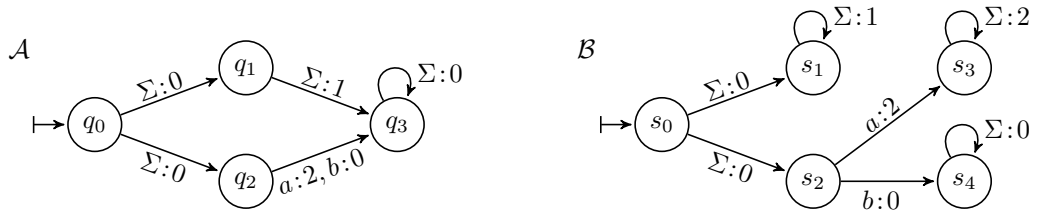


Figure 1 From [14, Figure 3]. Nondeterministic automata that demonstrate the case of Notion $\not\equiv$ Threshold-Notion with respect to the notions listed above. The automaton $\mathcal{A}$ demonstrates it with respect, for example, to the Sum/DSum/Sup value functions, and $\mathcal{B}$ with respect, for example, to the Avg/LimSup/LimInf/LimSupAvg/LimInfAvg value functions. Consider, for instance, determinizability by pruning – $\mathcal{A}$ is not determinizable by pruning, but for every threshold t it is: going only from q_0 to q_1 for a threshold $t \leq 1$ and going only from q_0 to q_2 for a threshold $t > 1$.

5 Relations between Notions

Some of the notions are related to each other by means of implication or equivalence. In some cases, their generalization to the quantitative setting does not change their interrelations, while in other cases it strictly refines them.

Syntactic notions and word-based notions. The syntactic notions do not change when generalized to the quantitative setting, and the semantic word-based notions also remain as is, just relating to threshold automata. Hence, the basic relations between these notions are as

in the Boolean setting. (See, for example, [34, 49, 38].) In addition, relations between notions that depend on the automaton type can be further researched with respect to threshold automata of various quantitative automata types.

All-Thresholds generalization of language-based notions. Considering Threshold-$\mathsf{N}(\mathcal{A})$ notions, they *morally* relate to each other as in the Boolean setting. For example, notions N_1 and N_2 that are equivalent with respect to Boolean automata (i.e., for every Boolean automaton $\mathcal{A}$, we have $\mathsf{N}_1(\mathcal{A}) = \mathsf{N}_2(\mathcal{A})$) are also equivalent with respect to threshold automata (i.e., for every quantitative automaton $\mathcal{A}$ and threshold $t \in \mathbb{R}$, we have $\mathsf{N}_1(t\text{-}\mathcal{A}) = \mathsf{N}_2(t\text{-}\mathcal{A})$, as the latter are also Boolean automata. Thus, we also have Threshold-$\mathsf{N}_1(\mathcal{A})$ = Threshold-$\mathsf{N}_2(\mathcal{A})$).

Yet, observe that general Boolean automata need not be regular or ω-regular, and notions that are equivalent with respect to (ω-)regular automata need not be equivalent with respect to Boolean automata that are derived from quantitative automata. For example, over ω-regular automata, history determinism implies pre-residuality, while the implication does not hold for all Boolean automata nor for all threshold quantitative automata. (See Section 5).

Basis Generalization of Language-Based Notions

Generalizing the basis of notions from language equality to function equality refines the relations between them: Notions that are different with respect to Boolean (ω-)regular automata are also different with respect to quantitative automata, as the former are special cases of the latter, however notions "known to be equivalent" might turn inequivalent in the quantitative setting, as is the case for example with history determinism and good-for-gameness.

As in the Boolean setting.

- Determinizability by pruning $\Rightarrow^1$ history determinism $\Rightarrow^2$ good-for-gameness $\equiv^3$ good-for-automataness.

  1. Determinizability by pruning is a special case of history determinism, restricting the history-determinism strategy to be positional.
  2. A history-determinism strategy for an automaton $\mathcal{A}$ can be combined with an optimal strategy in a game G, giving an optimal strategy in the game $G \times \mathcal{A}$ [14, Theorem 4].
  3. A game is a special case of an alternating automaton $\mathcal{B}$, giving the implication from right to left, while for every specific word w, as in the Boolean setting [13, Theorem 16], the value of $\mathcal{B}$ on w is viewed as a game, giving the implication from left to right.

- History determinism $\Rightarrow$ good-for-MDPness. As in the Boolean setting [31], a history-determinism strategy for an automaton $\mathcal{A}$ can be combined with an optimal strategy for an MDP M, ensuring an optimal strategy in $M \times \mathcal{A}$.
- The G2 conjecture. Two-token games (G2) characterize history determinism for Büchi and coBüchi automata [5, 11], and it is conjectured in [5, Conclusions] that this characterization also holds for parity automata (and thus for all ω-regular automata). It is shown in [12] that G2 also characterizes history determinism for various quantitative automata, and it is open whether this is the case for all quantitative automata.

Different from the Boolean setting.

- Good-for-gameness $\not\Rightarrow$ history determinism. While the two notions are equivalent and often mixed in the ω-regular setting, they turn out to be inequivalent in the quantitative setting, having good-for-gameness $\equiv$ Threshold-history-determinism $\not\rightleftarrows$ history determinism [14, Theorem 4]. (The left equivalence assumes that the letter game on $\mathcal{A}$ is determined [14, Lemma 7].) For example, the automata in Figure 1 are good for games but not history deterministic.
- History determinism $\not\Rightarrow$ pre-residuality. In the ω-regular setting, the implication holds [43, Lemma 18] (and there is no back implication [1, Theorem 1]). Yet, the implication is based on the implicit assumption that the value function depends on suffixes: a history deterministic automaton $\mathcal{A}$ all of whose transitions are used by some history deterministic strategy is guaranteed, in this case, to be residual, as otherwise once Eve chooses a non-residual transition, Adam will choose a suffix that witnesses its non-residuality, leading to Eve's lose. However, this need not be the case if the value function also depends on prefixes, as demonstrated in Figure 2.

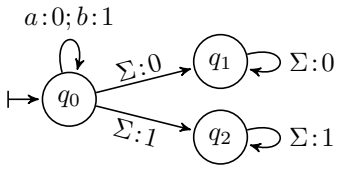


Figure 2 A Val automaton $\mathcal{A}$ on finite words with the value function $\mathsf{Val}(\rho) = 1$ if ρ has both even and odd values, and 0 otherwise. Notice that $\mathcal{A}$ is history deterministic but not pre-residual.

6 Conclusions

We have analyzed how notions of limited nondeterminism can be generalized from Boolean to quantitative automata; first observing that it depends on whether the notion is syntactic, semantic word-based, or semantic language-based, and then analyzing the possible notion generalizations within each of these three classes.

Our analysis results with generalized notions that define novel classes of quantitative automata, and opens up new research directions, which can contribute to the advance of quantitative verification and synthesis. For example, results on history deterministic and good for games quantitative automata directly relate to possible new solutions to quantitative synthesis [14].

In addition, the interrelation between the notions in the quantitative setting is shown to refine their known interrelation with respect to ω-regular automata, providing better understanding of the notions also with respect to Boolean automata whose behavior need not be regular.

References

1 Bader Abu Radi, Orna Kupferman, and Ofer Leshkowitz. A hierarchy of nondeterminism. In *Proc. of MFCS*, volume 202 of *LIPIcs*, pages 85:1–85:21, 2021.
2 Shaull Almagor, Udi Boker, and Orna Kupferman. Formalizing and reasoning about quality. *Journal of the ACM*, 63(3):24:1–24:56, 2016.
3 Shaull Almagor, Orna Kupferman, Jan Oliver Ringert, and Yaron Velner. Quantitative assume guarantee synthesis. In *Proc. of CAV*, pages 353–374. Springer, 2017.

4 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 6(2):28:1–28:36, 2010.

5 Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *Proc. of FSTTCS*, page 16, 2018.

6 Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. of CAV*, pages 140–156, 2009.

7 Udi Boker. Why these automata types? In *Proc. of LPAR*, pages 143–163, 2018.

8 Udi Boker. Quantitative vs. weighted automata. In *Proc. of RP*, pages 1–16, 2021.

9 Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM Trans. Comput. Log.*, 15(4):27:1–27:25, 2014.

10 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Proc. of ICALP*, pages 89–100, 2013.

11 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *arXiv:2002.07278*, 2020.

12 Udi Boker and Karoliina Lehtinen. Token games and history deterministic quantitative automata. Submitted.

13 Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *Proc. of CONCUR*, pages 19:1–19:16, 2019.

14 Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In *Proc. of FSTTCS*, pages 35:1–35:20, 2021.

15 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Proc. of LATA*, pages 3–23, 2016.

16 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *Proceedings of FCT*, pages 3–13, 2009.

17 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010.

18 Krishnendu Chatterjee, Thomas A Henzinger, Barbara Jobstmann, and Rohit Singh. Quasy: Quantitative synthesis tool. In *Proc. of TACAS*, pages 267–271. Springer, 2011.

19 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proc. of ICALP*, pages 139–150, 2009.

20 Thomas Colcombet. Fonctions régulières de coût. *Habilitation à diriger les recherches, École Doctorale de Sciences Mathématiques de Paris Centre*, 2013.

21 Thomas Colcombet. Unambiguity in automata theory. In *Proc. of DCFS*, pages 3–18, 2015.

22 Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In *Proc. of FOSSACS*, pages 1–26, 2019.

23 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

24 Laurent Doyen. Games and automata: From Boolean to quantitative verification. Habilitation Thesis, École Normale Supérieure de Cachan, 2011.

25 Marco Faella, Axel Legay, and Mariëlle Stoelinga. Model checking quantitative linear time logic. *Electr. Notes Theor. Comput. Sci.*, 220(3):61–77, 2008.

26 Emmanuel Filiot, Ismaël Jecker, Nathan Lhote, Guillermo A. Pérez, and Jean-François Raskin. On delay and regret determinization of max-plus automata. In *LICS*, pages 1–12, 2017.

27 Emmanuel Filiot, Christof Löding, and Sarah Winter. Synthesis from weighted specifications with partial domains over finite words. In *FSTTCS*, pages 46:1–46:16, 2020.

28 Daniel Goč and Kai Salomaa. Computation width and deviation number. In *Descriptional Complexity of Formal Systems*, pages 150–161, 2014.

29 Jonathan Goldstine, Chandra M. R. Kintala, and Detlef Wotschke. On measuring nondeterminism in regular languages. *Inf. Comput.*, 86(2):179–194, 1990.

30 Mika Göös and Stefan Kiefer. Lower bounds on unambiguous automata complementation and separation via communication complexity. *CoRR*, abs/2109.09155, 2021. `arXiv:2109.09155`.

31 Ernst Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Good-for-MDPs automata for probabilistic analysis and reinforcement learning. In *Proc. of TACAS*, pages 306–323, 2020.

32 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proc. of CSL*, pages 395–410, 2006.

33 Markus Holzer, Kai Salomaa, and Sheng Yu. On the state complexity of k-entry deterministic finite automata. *J. Autom. Lang. Comb.*, 6(4):453–466, 2001.

34 Juraj Hromkovič, Juhani Karhumäki, Hartmut Klauck, Georg Schnitger, and Sebastian Seibert. Measures of nondeterminism in finite automata. In *Proc. of ICALP*, pages 199–210, 2000.

35 Juraj Hromkovic, Sebastian Seibert, Juhani Karhumäki, Hartmut Klauck, and Georg Schnitger. Communication complexity method for measuring nondeterminism in finite automata. *Inf. Comput.*, 172(2):202–217, 2002.

36 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Reactive synthesis without regret. *Acta Informatica*, 54(1):3–39, 2017.

37 Martin Kappes. Descriptional complexity of deterministic finite automata with multiple initial states. *Journal of Automata, Languages and Combinatorics*, 5:269–278, January 2000.

38 Chris Keeler and Kai Salomaa. Branching measures and nearly acyclic NFAs. *Int. J. Found. Comput. Sci.*, 30(6-7):1135–1155, 2019.

39 Chris Keeler and Kai Salomaa. Alternating finite automata with limited universal branching. In *Proc. of LATA*, pages 196–207, 2020.

40 Chandra M. R. Kintala. Refining nondeterminism in context-free languages. *Math. Syst. Theory*, 12:1–8, 1978.

41 Chandra M. R. Kintala and Patrick C. Fischer. Computations with a restricted number of nondeterministic steps (extended abstract). In *Proc. of STOC*, pages 178–185, 1977.

42 Chandra M. R. Kintala and Detlef Wotschke. Amounts of nondeterminism in finite automata. *Acta Informatica*, 13:199–204, 1980.

43 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Proc. of ICALP*, pages 299–310, 2015.

44 Karoliina Lehtinen and Udi Boker. Register games. *Log. Methods Comput. Sci.*, 16(2), 2020.

45 Hing Leung. Structurally unambiguous finite automata. In *Proc. of CIAA*, pages 198–207, 2006.

46 Christof Löding and Stefan Repke. Decidability results on the existence of lookahead delegators for NFA. In *Proc. of FSTTCS*, pages 327–338, 2013.

47 Christof Löding and Andreas Tollkötter. State space reduction for parity automata. In *Proc. of CSL*, pages 27:1–27:16, 2020.

48 Anirban Majumdar and Denis Kuperberg. Computing the width of non-deterministic automata. *Logical Methods in Computer Science*, 15, 2019.

49 Alexandros Palioudakis, Kai Salomaa, and Selim Akl. Comparisons between measures of nondeterminism on finite automata. In *Proc. of DCFS*, pages 217–228, 2013.

50 Mikhail A. Raskin. A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In *Proc. of ICALP*, pages 138:1–138:11, 2018.

51 Bala Ravikumar and Nicolae Santean. On the existence of lookahead delegators for NFA. *Int. J. Found. Comput. Sci.*, 18(5):949–973, 2007.

52 M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. of LICS*, pages 332–344, 1986.

How to Develop an Intuition for Risk... and Other Invisible Phenomena

Natasha Fernandes
School of Engineering and IT, UNSW Canberra, Australia

Annabelle McIver
Department of Computing, Macquarie University, Sydney, Australia

Carroll Morgan
School of Computer Science and Engineering, UNSW, Sydney, Australia

Abstract

The study of quantitative risk in security systems is often based around complex and subtle mathematical ideas involving probabilities. The notations for these ideas can pose a communication barrier between collaborating researchers even when those researchers are working within a similar framework.

This paper describes the use of geometrical representation and reasoning as a way to share ideas using the minimum of notation so as to build intuition about what kinds of properties might or might not be true. We describe a faithful geometrical setting for the channel model of quantitative information flow (QIF) and demonstrate how it can facilitate "proofs without words" for problems in the QIF setting.

2012 ACM Subject Classification Security and privacy → Formal methods and theory of security

Keywords and phrases Geometry, Quantitative Information Flow, Proof, Explainability, Privacy

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.2

Category Invited Talk

1 Introduction

The analysis of security- and privacy vulnerabilities continues to be a challenging and important problem. Most practitioners acknowledge that online activity will never be absolutely safe, and so rigorous scrutiny on the basis of models and proof has a significant role to play in explaining and evaluating the severity of the threats that remain. But understanding *risk* is generally a fraught process: not only must it contend with opinions about what constitutes risky behaviour, but understanding must necessarily accommodate "invisible" extrinisic influences. For example nothing about the four digit integer 6174 suggests that it would be risky to send it in an email unless we find out that it is someone's PIN and could therefore put their life's savings in jeopardy. [1]

Assuming that evaluating the impact of risk in a given scenario is something that is still a useful thing to do, the following questions arise. Can we provide quantitative measurements that provide some sense of severity of a discovered vulnerability? How can we elaborate qualitative explanations for any numerical measurements of vulnerability that we might compute? How can we advance and share our knowledge of security and privacy defences when those explanations can be highly technical and abstract?

[1] Actually 6174 turns out to have a very curious intrinsic property of being invariant under Kaprekar's operation.

In this paper we describe our experience of addressing those questions using geometrical visualisations of the Quantitative Information Flow (QIF) framework [1]. Our development of this style of "geometrical reasoning" came about through a collaboration between teams who were working on the similar problems in quantification of security risks, using (as it turned out) the same basic mathematical principles, but with very different notations and somewhat differing objectives. We discovered that using visualisations helped enormously in sharing mathematical ideas and building intuition amongst the team members. The visualisations helped confirm facts that we knew in various forms, and enabled us not only to refute conjectures we thought might be true, but also to suggest to us conjectures that we had not discovered for ourselves when we were using purely formal notations. Subsequent rigorous proof was then able to remove the geometrical hunch, providing new theorems in privacy (described below) [8].

Thus a completely unexpected benefit of our "geometry of risk" was its facilitation of "notation-free" communication of fundamental but complex ideas providing summaries of proof steps in the form of geometrical constructions. Equipped with a sense of certainty endowed by the geometry allowed us to formulate and prove formally new theorems thus advancing our understanding of QIF and how it could be used in security and privacy. In this we find we are in agreement with Henri Poincaré's insights on the benefit of pictorial representations to promote intuition and communication of complex mathematical ideas between scientitsts:

> "I have already had occasion to insist on the place intuition should hold in the teaching of the mathematical sciences. Without it young minds could not make a beginning in the understanding of mathematics; they could not learn to love it and would see in it only a vain logomachy; above all, without intuition they would never become capable of applying mathematics. But now I wish before all to speak of the role of intuition in science itself. If it is useful to the student, it is still more so to the creative scientist."
>
> Extract from *Intuition and Logic in mathematics* appearing in *La valeur de la science*, Henri Poincaré, 1905.

1.1 Related work

There are many examples in mathematics of visualisations used to provide explanation and insight for formal reasoning. The earliest instance is of course Euclid's elements for reasoning about spacial relationships; Oliver Byrne's 1847 treatise [3] is a masterful account of how diagrams can be used optimally to convey complex geometrical ideas. Roger Nelsen's *Proofs without words* [11] promotes the use of visualisations to explain mathematical ideas in a range of topics from algebra to calculus, and theorems about sequences and series. And one of the shortest papers ever written consisted essentially of two figures, as the explanation of a mathematical result [6].

The geometry underlying the study of Quantitative Information Flow first appeared in Alvim [1]; this also includes Morgan's "overlapping triangles" demonstration that the at-least-as-secure-as partial order on information-flow channels is not (alas) a lattice as well. Fernandes' application of geometrical ideas to study universally-optimal utility mechanisms for differential privacy appears in [8].

In this paper therefore we emphasise the role played by geometrical ideas in supporting the communication via "proofs without words" for quantifying security and privacy risks using quantitative information flow.

2 Quantitative Information Flow Basics

The informal idea of a secret is that it is something about which there is some uncertainty, and the greater the uncertainty the more difficult it is to discover exactly what the secret is. For example, one's 4-digit PIN should be kept secret, but if the last two digits are discovered to be 7 and 4, then it becomes much easier to guess the rest of it. That is, when *any* information about a secret becomes available to an observer (often referred to as an adversary) the uncertainty is reduced, allowing the property, or even exact value of the secret, to be more accurately inferred. When that happens, we say that information (about the secret) has leaked.

Quantitative Information Flow (QIF) makes the above intuition mathematically precise. Given a range of possible secret values of (finite) type $\mathcal{X}$, we model a secret as a discrete probability distribution of type $\mathbb{D}\mathcal{X}$, because it ascribes "probabilistic uncertainty" to the secret's exact value. Given some distribution $\pi\colon \mathbb{D}\mathcal{X}$, we write π_x for the probability that π assigns to $x\colon \mathcal{X}$, with the idea that the more likely it is that the real value is some specific x, then the closer π_x will be to 1. Usually the uniform distribution over $\mathcal{X}$ models a secret which could equally well take any one of the possible values drawn from its type and we might say that, beyond the existence of the secret, nothing else is known. There could, of course, be many reasons for using some other distribution: for example if the secret were the height of an individual then a normal distribution might be more realistic. In any case, once we have a secret, we are interested in analysing whether an algorithm, or protocol, that uses it might leak some information about it. To do that we define a measure for uncertainty, and use it to compare the uncertainty of the secret before and after executing the algorithm. If we find that the two measurements are different then we can say that there has been an information leak.

The idea of measuring security risk in terms of quantitative information flow (and that name) was pioneered by Clarke et al. [4]. That and other early QIF analyses of information leaks in computer systems [5, 4] used Shannon entropy [12] to measure uncertainty because it captures the idea that more uncertainty implies "more secrecy" – and indeed the uniform distribution corresponds to maximum Shannon entropy (corresponding to maximum "Shannon uncertainty"). More recent treatments have shown however that Shannon entropy is not *always* the best way to measure uncertainty in security contexts: precisely because of its beautiful generality, it might not model scenarios relevant to the goals of a particular adversary. Indeed there are some circumstances where a Shannon analysis actually gives a more favourable assessment of security than is actually warranted, when the adversary's motivation is taken into account [13].

Alvim et al. [2] proposed a notion of uncertainty, more general than Shannon, based on "gain functions". In this paper we will use the equivalent formulation of loss functions. [2] A *loss function* measures a secret's uncertainty according to how it affects an adversary's actions within his context. We write $\mathcal{W}$ for a (usually finite) set of actions available to an adversary corresponding to an "attack scenario" where the adversary tries to infer something (e.g. some property, but perhaps its actual value) about the secret. For a given secret $x\colon \mathcal{X}$, an adversary's choice of action $w\colon \mathcal{W}$ results in the adversary's losing something beneficial to his objective. That loss can vary depending on the adversary's action (w) and the exact value of the secret (x). The more effective is the adversary's choice in how to act, the more he is able to overcome any uncertainty concerning the secret's value.

[2] Shannon Entropy is a special case: it can be defined using a loss function.

▶ **Definition 1.** *Given a type $\mathcal{X}$ of secrets, a (real valued) loss function $\ell\colon \mathcal{W}\times\mathcal{X}\to\mathbb{R}$ is such that $\ell(w,x)$ determines the loss to an adversary if he chooses w and the secret is x.*

A simple example of a loss function is br (for "Bayes Risk", as explained below), where $\mathcal{W} := \mathcal{X}$ so that the available actions are simply "guess the value of x", and thus [3]

$$\mathsf{br}(w,x) \;:=\; 0 \;\; \textit{if}\;\; w{=}x \;\; \textit{else}\;\; 1\ . \tag{1}$$

For this scenario, the adversary's goal is to determine the exact value of the secret, so he loses nothing if he correctly guesses the value of a secret (a good outcome for him), and otherwise he loses \$1 (bad).

Elsewhere the great utility and expressivity of loss functions for measuring various attack scenarios relevant to security –far beyond just guessing the secret's value – have been thoroughly explored [1]. Given a loss function we define the *uncertainty* of a secret in $\mathbb{D}\mathcal{X}$ relative to the scenario the loss function describes: it is the minimum *average* loss to an adversary. More explicitly, for each action w, the adversary's average loss relative to some distribution π of the secret is $\sum_{x\in\mathcal{X}} \ell(w,x)\times\pi_x$; thus his minimum average loss is the action that yields that minimal average.

▶ **Definition 2.** *Let $\ell\colon \mathcal{W}\times\mathcal{X}\to\mathbb{R}$ be a loss function, and $\pi\colon\mathbb{D}\mathcal{X}$ be a secret. The* uncertainty *$U_\ell[\pi]$ of the secret wrt. ℓ is given by*

$$U_\ell[\pi] \;:=\; \min_{w\colon\mathcal{W}} \sum_{x\colon\mathcal{X}} \ell(w,x)\times\pi_x\ .$$

For a secret $\pi\colon\mathbb{D}\mathcal{X}$, the uncertainty wrt. the loss function br is $U_{\mathsf{br}}[\pi] := 1 - \max_{x\colon\mathcal{X}} \pi_x$, that is the deficit of the maximum probability assigned by π to possible values of x. The adversary's best strategy for optimising his loss would therefore be to choose the value x that corresponds to the maximum probability under π. This uncertainty U_{br} is called *Bayes' Risk*.

We now define a *mechanism* to be an abstract model of a protocol or algorithm that uses secrets. As the mechanism executes we assume that there are a number of observables, that is outputs it might produce, that can depend on the actual value of the secret it is processing: we write $\mathcal{Y}$ for the type of those observables. The *model* of the mechanism therefore assigns a probability that $y\colon\mathcal{Y}$ might be observed given that the secret is x. Such observables could be sample timings in a timing analysis in cryptography, for example.

▶ **Definition 3.** *A* mechanism *is a stochastic channel*[4] *$C\colon\mathcal{X}\times\mathcal{Y}\to[0,1]$. The value C_{xy} is the probability that y is observed given that the secret is x.*

Given a (prior) secret $\pi\colon\mathbb{D}\mathcal{X}$ and mechanism C we write $\pi{\triangleright}C$ for the joint distribution *in $\mathbb{D}(\mathcal{X}\times\mathcal{Y})$ defined*

$$(\pi{\triangleright}C)_{xy} \;:=\; \pi_x\times C_{xy}\ .$$

For each $y\colon\mathcal{Y}$, the marginal probability *that y is observed is $p_y := \sum_{x\colon\mathcal{X}} (\pi{\triangleright}C)_{xy}$.* [5]

For each observable y, the corresponding posterior probability *of the secret is the conditional $\pi|^y$ in $\mathbb{D}\mathcal{X}$ defined $(\pi|^y)_x := (\pi{\triangleright}C)_{xy}/p_y$.* [6]

---

[3] We write $:=$ for "is defined to be".

[4] "Stochastic" means that the rows sum to 1, i.e. that $\sum_{y\colon\mathcal{Y}} C_{xy} = 1$ for each x.

[5] Equivalently that is $\sum_x \pi_x C_{xy}$.

[6] We assume for convenience that when we write p_y the terms C, π and y are understood from the context. Notation suited for formal calculation would need to incorporate C and π explicitly.

Given a prior secret π and mechanism C, it's clear that the entry $\pi_x \times C_{xy}$ of the joint distribution $\pi \triangleright C$ is the probability that the actual secret value is x and the observation is y. That joint distribution contains two important pieces of information, which we single out: the probability p_y of observing y and the corresponding posterior $\pi|^y$ which represents the adversary's updated view about the uncertainty of the secret's value. If the uncertainty of the posterior increases, then information about the secret has leaked and a rational adversary *might* use it to decrease his loss by changing how he acts, i.e. by altering his choice of action w. The adversary's average overall loss, taking the observations into account, is defined to be the average posterior uncertainty (i.e. the average loss of each posterior distribution, weighted according to their respective marginals):

$$U_\ell[\pi \triangleright C] := \sum_{y \in \mathcal{Y}} p_y \times U_\ell[\pi|^y] \ , \qquad \text{where } p_y \text{ and } \pi|^y \text{ are defined at Def. 3.} \tag{2}$$

The "rationality" of the adversary is inside the U_ℓ, expressed by the $\min_{w:\mathcal{W}}$ there (Def. 2).

2.1 Hyper-distributions summarise the risk

In the above model the key structure used to compute the posterior loss is $[\pi \triangleright C]$ – which can be represented (more abstractly) as a *hyper-distribution*, that is a distribution of type $\mathbb{D}^2\mathcal{X}$ where the outer probability is p^y, the marginal probability of an observation and the inner distribution is the posterior corresponding to that y.

An advantage of that abstraction is that there is then a partial order on $\mathbb{D}^2\mathcal{X}$ which allows the robust comparison of channels wrt. their information flow properties. It is the relation ($\sqsubseteq$), defined

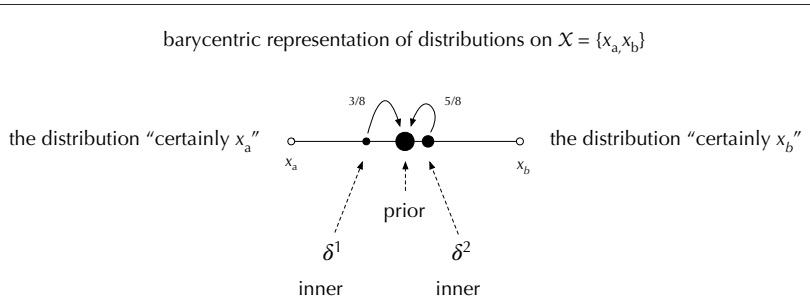
▶ **Definition 4.** *Let C, D be channels. We say that $C \sqsubseteq D$ if for all loss functions ℓ and prior distributions π we have $U_\ell[\pi \triangleright C] \leq U_\ell[\pi \triangleright D]$.*

That is, if D refines C then we can be sure that the adversary always loses no less with D than with C in any scenario that can be defined by some π and ℓ of the correct type: that is, for a defender D is at least as secure as C. As we noted above, Shannon Entropy is a special case of an ℓ-uncertainty – but with infinitely many actions– and we call its loss function se, so that Shannon entropy is U_{se}.

2.2 Reasoning geometrically

The basic model of QIF – set out mathematically above – turns out to have an appealing geometrical interpretation, one that we can use to visualise the relationships between channels and also loss functions. The first step is to visualise hyper-distributions using a *barycentric* representation of $\mathbb{D}\mathcal{X}$ [1][chapter 12]. Recall from §2.1 above that a hyper-distribution is of type $\mathbb{D}^2\mathcal{X}$, equivalently $\mathbb{D}(\mathbb{D}\mathcal{X})$ or a "distribution of distributions". The "inner $\mathbb{D}$" defines what we call "inners", distribution on $\mathcal{X}$ directly, that correspond to posteriors associated with the observations – and when $\mathcal{X}$ is finite the inners are (non-negative) 1-summing vectors in $\mathbb{R}^{|\mathcal{X}|}$. Thus a hyper-distribution corresponds to a convex sum of 1-summing vectors. The "outer $\mathbb{D}$" is the "outer" that gives the marginal probabilities associated with each inner, the weights in that convex sum.

Suppose first that $\mathcal{X} := \{x_a, x_b\}$ and that $\mathsf{u}: \mathbb{D}\mathcal{X}$ is the uniform prior. As explained above u corresponds to a 1-summing vector, so that as expected in this case $\mathsf{u} := (1/2, 1/2)$. Here the first component is the probability that the secret is x_a and the second component is the



The barycentric representation of $\mathbb{D}\{x_a, x_b\}$ is the horizontal line between x_a and x_b (corresponding to the 1-summing vectors $(1,0)$ and $(0,1)$). The two inners are labelled δ^1 and δ^2 with their positions relative to the $x_a(x_b)$ labels corresponding to the probabilities $\delta^1_{x_a}(\delta^1_{x_b})$ and $\delta^2_{x_a}(\delta^2_{x_b})$. The outers are indicated as the relative distance between the δ's and their weighted average to obtain the original prior u.

Figure 1 Barycentric representation of $[\mathsf{u} \triangleright C]$.

probability that the secret is x_b – since u is uniform, those probabilities are the same. Next let the observations $\mathcal{Y}{:=}\{y_1, y_2\}$ be and consider the channel $C \in \mathcal{X}{\times}\mathcal{Y}$, and its corresponding hyper-distribution $[\mathsf{u}{\triangleright}C]$ when the prior is u.

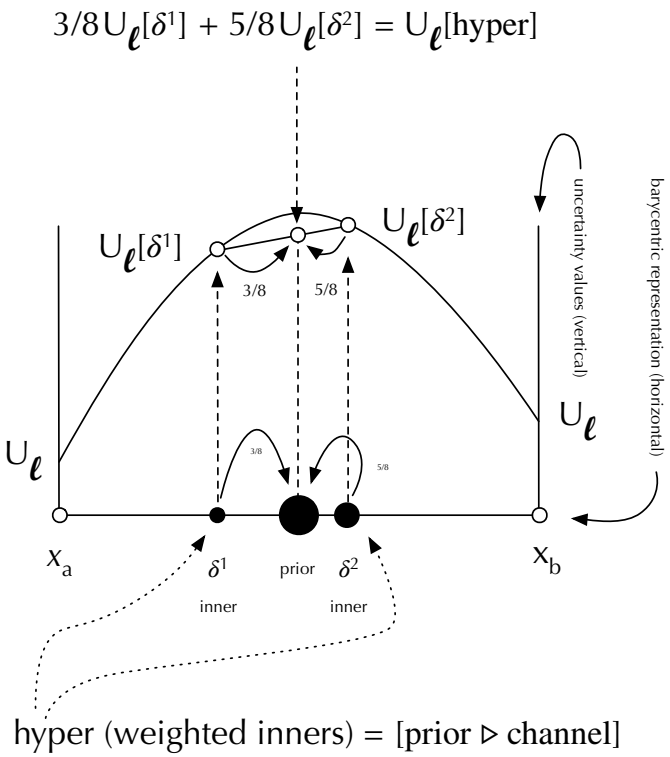
$$C \;:=\; \begin{array}{c} \\ x_a \\ x_b \end{array} \begin{array}{c} \begin{array}{cc} y_1 & y_2 \end{array} \\ \left(\begin{array}{cc} 1/2 & 1/2 \\ 1/4 & 3/4 \end{array}\right) \end{array} \qquad [\mathsf{u}{\triangleright}C] \;:=\; \begin{array}{c|cc} & 3/8 & 5/8 \\ \hline x_a & 2/3 & 2/5 \\ x_b & 1/3 & 3/5 \end{array}$$

As we can see, the inners (i.e. posteriors) are $\delta^1{:=}(2/3, 1/3)$ and $\delta^2{:=}(2/5, 3/5)$, and their corresponding "outers" the marginal probabilities $3/8, 5/8$.

Since $\mathcal{X}$ has only two elements, the barycentric representation of $\mathbb{D}\mathcal{X}$ is one-dimensional, running from distribution "certainly x_a" at left to "certainly x_b" at right, as in Fig. 1. A point on that line represents a linear combination of those two extremes, and the larger the probability the distribution assigns to x_a, say, the closer its representing point is to the left-hand side. This barycentric representation therefore locates inners $\delta^{1,2}$ on that horizontal line, with for example δ^1 lying closer to the left-hand side because it assigns greater probability to x_a.

Fig. 1 also shows the prior (as a point in the middle of the line, because it's uniform), and the points representing $\delta^{1,2}$ are given sizes corresponding to the *outers* associated with them. If linearly combined with those sizes as coefficients, they will give the prior at the position shown (and with "size" 1). That is a property of the construction $[\pi{\triangleright}C]$ for any prior π (uniform or not) and channel C.

Next, given a loss function ℓ we can plot its associated uncertainty $U_\ell(\pi)$ on the vertical axis above the barycentric representation, and in fact (for all ℓ) it will determine a concave and continuous curve there. Fig. 2 also shows an uncertainty U_ℓ as a function from inners to reals. Here a particular loss $\ell(w, \cdot)$ becomes a tangent to the curve U_ℓ, and therefore the curve shown is the envelope of all those w-determined tangents. Locating δ^1, δ^2 on the (horizontal) barycentric axis, we can easily read off $U_\ell[\delta^1]$ and $U_\ell[\delta^2]$ as the height of the curve U_ℓ above them.



The barycentric axis is the horizontal labelled x_a, x_b, and the inners (posteriors) of $[\mathsf{u}\triangleright C]$ are indicated as δ^1, δ^2. The circles' sizes represent their respective outer (i.e. marginal) probabilities. The curve denotes U_ℓ as a function of points (distributions) on the horizontal barycentric axis. Thus $U_\ell[\delta^1])$ is the height of the vertical line from δ^1 up to the curve U_ℓ.

The average of the inners δ^1 and δ^2 weighted according to their respective outer probabilities in $[\mathsf{u}\triangleright C]$ yields the prior u again; but the (same) average of the expected losses is $U_\ell[\mathsf{u}\triangleright C]$, whose value is the height of the vertical line from the prior u (in this case) to the line joining the two points on the U_ℓ curve.

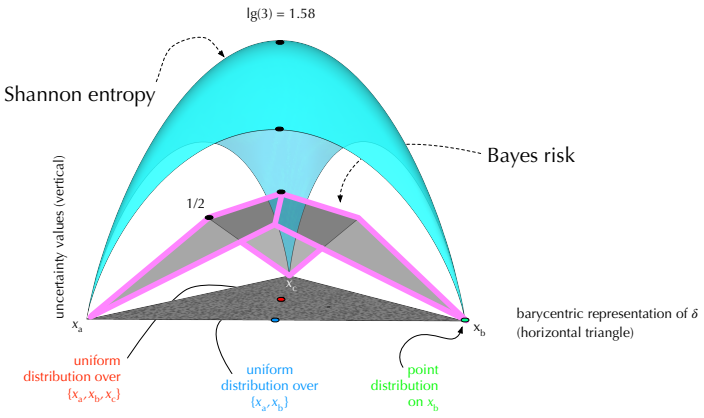
Figure 2 Barycentric representation of uncertainties and loss functions.

With all that done, it is easy to compute $U_\ell[\mathsf{u}\triangleright C]$ as that same weighted average of the heights of U_ℓ above them: and that is done geometrically by connecting those two points on $U_\ell[\mathsf{u}\triangleright C]$ with a straight line, and noting its height *above the prior*. That is, we simply take the weighted average $3/8\times U_\ell[\delta^1] + 5/8\times U_\ell[\delta^2]$, which is depicted on the figure as the point at which the vertical line from u meets the line joining the uncertainties at $U_\ell[\delta^{1,2}]$.

Because U_ℓ is concave, we now have a "proof without words" [11] of the well known Jensen's inequality. [7] [8]

[7] For more than two inners, the same "proof" generalises; but the combinations have to be done two-by-two.

[8] As has been noted by many authors, a "proof without words" is not a *proof*. We use it here to mean that it is a suggestion for a proof strategy – of course the actual proof must be demonstrated with appropriate words.



The barycentric representation of distributions over three values $\{x_a, x_b, x_c\}$ is a (two-dimensional) triangle, rather than a straight line as it was in Fig. 2 above when $\mathcal{X}$ had just two values: it's always one dimension smaller than the size of $\mathcal{X}$, because the constraint that the probabilities sum to 1 loses one degree of freedom.

Figure 3 Barycentric representations on three points.

In the following sections we use pictures and constructions given above as a way to build and share intuition about the behaviour of channels. Our demonstrations can be thought of as providing hints to explain a complex argument; here we suppress the accompanying formal arguments leaving the constructions as "proofs with very few words".

3 Some QIF proofs without *many* words

3.1 Refinement seen geometrically

In Def. 4 the *refinement* partial order is defined between hyper-distributions, that the loss with respect to *any* uncertainty must increase. But the *Coriaceous Theorem* [10] gives an equivalent geometric definition. One hyper – an (outer) distribution over (its inner) distributions – is a refinement of another just when the more refined's inners can be realised as a weighted merge of the less refined's inners. In Fig. 2 for example, the hyper represented by the two smaller dots can be refined to another – to many others – by "carving off" pieces of the inners and merging them according to their respective weights. [9] And the proof of the Coriaceous Theorem is *itself* inspired geometrically, because it relies on the Separating-Hyperplane Lemma, where the convex region represents the more refined hyper, and the separating plane's normal gives the coefficients of the loss function that satisfies the original Def. 4.

The geometric view tells us immediately that a more-refined hyper's inners must lie (non-strictly) within the convex hull of the less-refined hyper, and helps us (in §3.2) to see – again geometrically – whether the refinement partial order is a lattice. The following stronger fact (requiring a full proof) allows us to make stronger geometric arguments [1].

[9] In the extreme case, they can be refined to the singleton hyper whose sole inner is the original prior, as illustrated by the arrows in that figure.

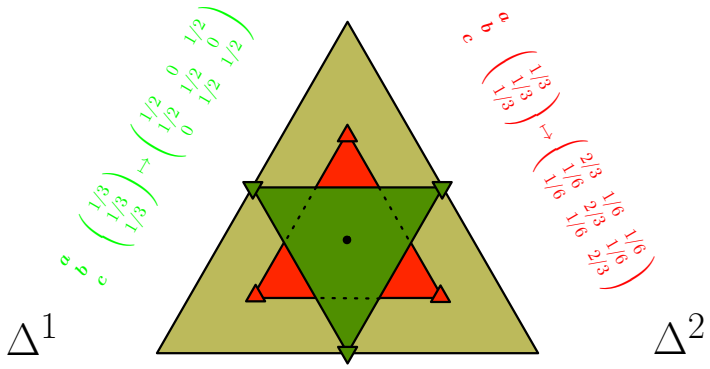


Figure 4 Two triangular hypers $\Delta^{1,2}$ on $\mathcal{X} = \{x_a, x_b, x_c\}$.

▶ **Lemma 5.** *Let the (finite) state space $\mathcal{X}$ have N elements, and let some hyper Δ have N linearly independent inners. Then any other hyper Δ' all of whose inners lie within the convex hull of Δ (and that is derived from the same prior) is a refinement of Δ, no matter how many inners Δ' might have.*

The original "qualitative" *Lattice of Information* [9] is (by its very title) a lattice; but in the quantitative case (here), it turns out that it is not a lattice.

3.2 The refinement order is not a lattice

In Fig. 3 we show how our barycentric representation of distributions appears when $\mathcal{X}$ is $\{x_a, x_b, x_c\}$, no longer a line but now an equilateral triangle, shown in grey at the bottom. We will reason about hypers in that triangle.

Fig. 4 shows two hypers $\Delta^{1,2}$ (green and red resp.) over a state space $\mathcal{X} = \{x_a, x_b, x_c\}$. They are both generated from the uniform prior $(1/3, 1/3, 1/3)$, shown as a black dot at the centre of the barycentric triangle, and each of the hypers is a (smaller) equilateral triangle itself, having three inners (each) all three with outer probability $1/3$. All refinements of Δ^1 must lie within the green triangle; and all refinements of Δ^2 must lie within the red triangle; and so all refinements of *both* must lie within the yellow hexagon.

And so from Lem. 5 we know that *any* hyper (with the same prior) in the yellow hexagon must refine both $\Delta^{1,2}$, because they have only three inners (each).

Now consider three more hypers, each with only two inners (each with outer $1/2$), named $\Delta^{3,4,5}$ as shown in Fig. 5. Each of $\Delta^{3,4,5}$ is a refinement of each of $\Delta^{1,2}$, as noted just above, which fact we write compactly as $\Delta^{1,2} \sqsubseteq \Delta^{3,4,5}$. We show (after the following lemma) that there is no hyper Δ satisfying

$$\Delta^{1,2} \quad \sqsubseteq \quad \Delta \quad \sqsubseteq \quad \Delta^{3,4,5} \quad . \tag{3}$$

▶ **Lemma 6.** *If hypers Δ', Δ'' (with the same prior) lie (non-strictly) within the yellow hexagon of Fig. 5, and $\Delta' \sqsubseteq \Delta''$, then the outer probability of any of the (six) hexagon vertices in Δ'' cannot exceed the outer probability of that same vertex in Δ'.* [10]

That is because refinement interpolates inners, and interpolation of any *inners of Δ' (which, remember, are non-strictly within the hexagon) cannot increase the outer of one of the hexagon's vertices, because the hexagon is convex and its vertices are its extreme points.*

[10] If some vertex of the hexagon is not "actually" one of the inners of the hyper considered, we just consider its outer to be zero.

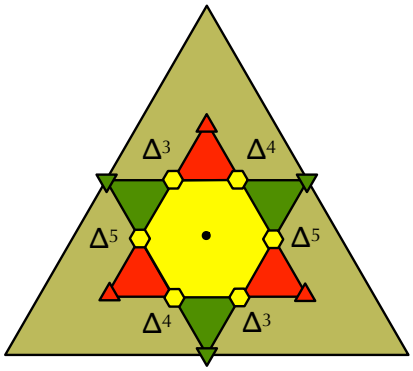


Figure 5 Three more two-inner hypers $\Delta^{3,4,5}$ on $\mathcal{X} = \{a, b, c\}$.

Now from Lem. 6 we have immediately that there is *no* Δ satisfying (3), because any Δ that refines both $\Delta^{1,2}$ must lie within the hexagon; and by Lem. 6 if Δ additionally is refined by all of $\Delta_{3,4,5}$ then the outer of Δ at all six vertices be at least $1/2$, impossible because those outers must sum to 1.

And so $(\sqsubseteq)$ on $\mathcal{X} = \{x_a, x_b, x_c\}$ is not a lattice: for both the join $\Delta^1 \sqcup \Delta^2$ and the meet $\Delta^3 \sqcap \Delta^4 \sqcap \Delta^5$, if they existed, would as Δ satisfy (3) – which Lem. 6 showed was impossible. And so neither exists.

3.3 Channel composition is not idempotent. . . unless it is deterministic

Channel composition (or parallel) composition is defined to be the channel obtained by independent executions of two channels, taking both their observations into account.

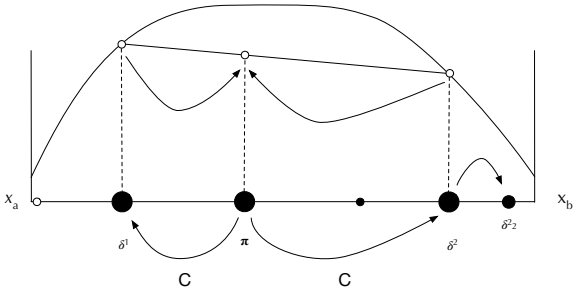
▶ **Definition 7.** *Let* $C{:}\mathcal{X}{\times}\mathcal{Y} \to [0,1]$, $D{:}\mathcal{X}{\times}\mathcal{Z} \to [0,1]$. *The parallel composition* $C\|D{:}\mathcal{X}{\times}(\mathcal{Y}{\times}\mathcal{Z}) \to [0,1]$ *of* C, D *is defined as the product space of observations which are now drawn from* $\mathcal{Y}{\times}\mathcal{Z}$*:*

$$(C\|D)_{x(yz)} := C_{xy} \times D_{xz} \ .$$

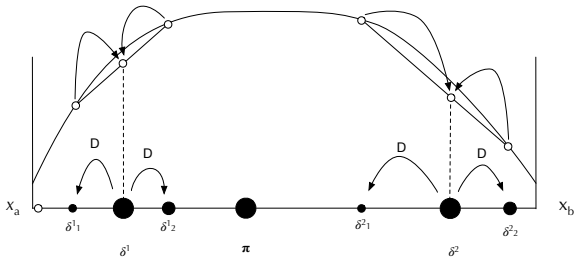
Landauer [9] showed that parallel composition of deterministic channels are idempotent; this suggests the question of whether parallel composition more generally is *idempotent*. Can it be the case that $C\|C = C$ when C is not deterministic?

We show that $C\|C \neq C$ for properly probabilistic channels by the geometric constructions shown in Fig. 2 and Fig. 6. First Fig. 2 shows that whatever the prior π, when a properly probabilistic channel with two observations is applied to it, there will be two inners, averaging to the original channel. But now Fig. 6 shows how to apply that construction twice: first C is applied tp the original prior, yielding two inners $\delta^{1,2}$ and then, because of the independence, D is now applied to each of $\delta^{1,2}$, where these new corresponding inners average to the δ's.

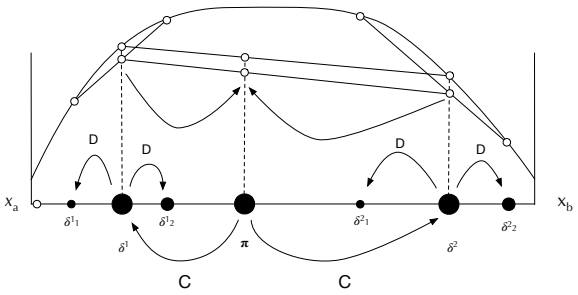
Using now the uncertainty U_ℓ we can visualise how the information flow must be different between C and $C \parallel D$: we apply the construction in Fig. 2 twice, each time averaging: first to compute $U_\ell[\delta^{1,2} \triangleright D]$ must be strictly less than each of $U_\ell[\delta^{1,2}]$; and then again to show that $U_\ell[\pi \triangleright (C\|D)]$. Our question is now answered by setting D to C.



Channel C makes inners $\delta^{1,2}$ from prior $\boldsymbol{\pi}$



Those inners $\delta^{1,2}$ become priors for D



The expected loss for $[\pi \triangleright C \| D]$ is determined by applying the construction from Fig. 2 twice. First $[\pi \triangleright C]$ is represented by its two inners $\delta^{1,2}$ which average to π; then the same construction is applied to $[\delta^1 \triangleright D]$ and $[\delta^2 \triangleright D]$ as shown. The average $U_\ell[\delta^1 \triangleright D]$ and $U_\ell[\delta^2 \triangleright D]$ are then weighted averages as described in Fig. 2. The expected loss $U_\ell[\pi \triangleright C \| D]$ is then computed by averaging those averages.

Figure 6 Construction for $C \| D$ in 2 dimensions using Shannon Entropy.

We note finally that this geometric construction assumes that the inners in $[\pi \triangleright C]$ are spread either side of π. This does not happen when C is deterministic i.e. does not have any values except 1's and 0's. For deterministic channels, the secrets are separated into equivalence classes and the information flow yields exactly which class a secret is in. Geometrically this means that when the support of an inner δ lies entirely within an equivalent class the inners $[\delta \triangleright C]$ are actually δ.

3.4 Which is better, the Laplace- or Geometric mechanism for implementing differential privacy?

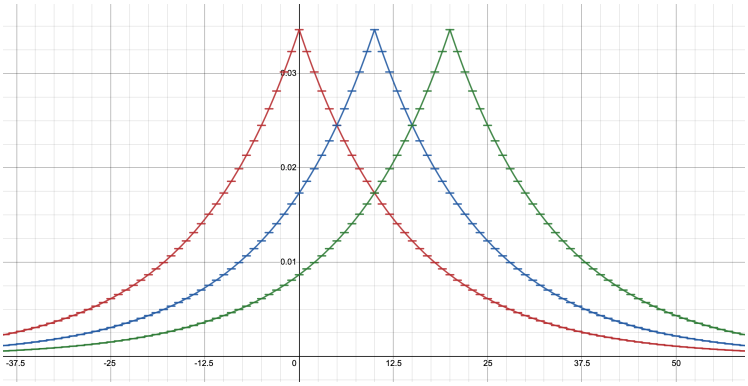


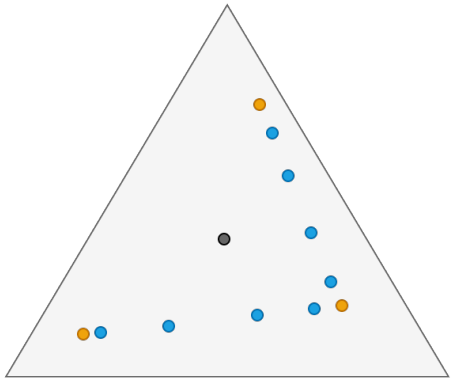
Figure 7 The Laplace (continuous pdf) and Geometric (discrete lines) noise mechanisms.

Differential privacy [7] is a technique for for providing individuals' data some measure of privacy when that data is shared through e.g. a query (to the database containing it). The idea is that rather than reporting the result of a raw query, instead some random noise (chosen according to a parameter ϵ) is added to the result and the noisy answer is then reported. Different methods of adding random noise have different properties of course – and those that are in keeping with the spirit of differential privacy can guarantee to make similar query results "indistinguishable in output" so that in practice an observer cannot tell apart the outputs of inputs that are already similar (in the raw), even when those raw results are distinguishable enough to risk a privacy breach.

Two popular methods of randomisation are based on the Geometric and Laplace probability distributions leading to the definition of the corresponding Geometric and Laplace mechanisms. Given the output of a query is some number d (consisting of e.g. the count of data entries satisfying a condition, or some average value) instead of outputting the raw d, the Geometric mechanism would output $d+c$ where c is distributed according to a geometric distribution; similarly the Laplace mechanism would output $d+e$ where e is distributed according to the Laplace distribution. The two different methods of randomisation are depicted in Fig. 7.

Interestingly, although they have broadly similar shapes (Fig. 7) albeit the Laplace gives a continuous"probability density function" and the Geometric a discrete number of outputs, there is no obvious way to compare the properties of these mechanisms in terms of how their privacy properties work. Perhaps they leak the same, or entirely different amounts of information when used as randomisers. It turns out that when viewed in terms of QIF channels we find that, for the same ϵ parameter we can say definitively that the Geometric mechanism leaks more information than does the Laplace mechanism, and thus the Laplace mechanism is strictly more private.

Our proof with very few words in illustrated in Fig. 8 which depicts the barycentric visualisation of the hyper-distributions corresponding to the Geometric- and Laplace mechanisms when applied to three secret values (i.e. potential raw query results) and represented as QIF channels. As explained above, the inners of the corresponding hyper-distributions can be located as points on the plane in three dimensions. Curiously we see that the Geometric mechanism produces three inners (orange dots in Fig. 8) which are linearly independent because they do not lie on a line. Even more curiously the inners from the Laplace mechanism (blue dots in Fig. 8) lie within the convex hull of the Geometric's inners, and therefore by Lem. 5 the Laplace perforce refines the Geometric, which from Def. 4 means that the Geometric mechanism always leaks more information about the secret than does the Laplace mechanism. This observation, discovered purely by this visualisation led to the fully formal proof of universal optimality of the Laplace mechanism for continuous inputs [8].



The Geometric mechanism on 3 secrets parametrised by ϵ consists of 3 linearly independent posteriors (orange). The Laplace mechanism parametrised by the same ϵ (and on the same domain) consists of the blue posteriors sitting on the edges – ie. in the convex hull – defined by the Geometric posteriors. It follows that the Laplace mechanism is a refinement of the Geometric mechanism.

Figure 8 Construction of Geometric (orange) and Laplace (blue) hypers in 3 dimensions.

4 Conclusions

In this paper we have demonstrated how to use geometrical ideas to explain complex ideas within the framework of quantitative information flow. Although they do not represent full formal proofs of these results they have proved to be useful for sharing ideas between different groups of collaborators and therefore in developing the field.

References

1 M. S. Alvim, K. Chatzikokolakis, A.K. McIver, C.C. Morgan, C. Palamidessi, and G. Smith. *The Science of Quantitative Information Flow*. Information Security and Cryptography. Springer International Publishing, 2020.

2 Mário S. Alvim, Kostas Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF 2012)*, pages 265–279, June 2012.

3 Oliver Byrne, Bruce Rogers, and Euclid. *The first six books of the elements of Euclid: in which coloured diagrams and symbols are used instead of letters for the greater ease of learners / by Oliver Byrne*. William Pickering London, 1847.

4 David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. *Electr. Notes Theor. Comput. Sci.*, 59(3):238–251, 2001.

5 Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005), 20-22 June 2005, Aix-en-Provence, France*, pages 31–45, 2005.

6 J. Conway and A. Soifer. Can $n^2 + 1$ unit equilateral trianges cover an equilateral triangle of side $> n$, say $n + \epsilon$? *The American Mathematical Monthly*, 18(143), 2005.

7 Cynthia Dwork. Differential privacy. In *Proc. 33rd International Colloquium on Automata, Languages, and Programming (ICALP 2006)*, pages 1–12, 2006.

8 Natasha Fernandes, Annabelle McIver, and Carroll Morgan. The laplace mechanism has optimal utility for differential privacy over continuous queries. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–12. IEEE, 2021.

9 Jaisook Landauer and Timothy Redmond. A lattice of information. In *Proc. 6th IEEE Computer Security Foundations Workshop (CSFW'93)*, pages 65–70, June 1993.

10 Annabelle McIver, Carroll Morgan, Geoffrey Smith, Barbara Espinoza, and Larissa Meinicke. Abstract channels and their robust information-leakage ordering. 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 83–102. Springer, 2014. doi:10.1007/978-3-642-54792-8_5.

11 Roger Nelsen. *Proofs Without Words: Exercises in Visual Thinking*, volume 1. MAA Press, 1993.

12 C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

13 Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *Proc. 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS '09)*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302, 2009.

Simulation by Rounds of Letter-To-Letter Transducers

Antonio Abu Nassar
Computer Science Department, Technion, Haifa, Israel

Shaull Almagor
Computer Science Department, Technion, Haifa, Israel

Abstract

Letter-to-letter transducers are a standard formalism for modeling reactive systems. Often, two transducers that model similar systems differ locally from one another, by behaving similarly, up to permutations of the input and output letters within "rounds". In this work, we introduce and study notions of simulation by rounds and equivalence by rounds of transducers. In our setting, words are partitioned to consecutive subwords of a fixed length k, called rounds. Then, a transducer $\mathcal{T}_1$ is k-round simulated by transducer $\mathcal{T}_2$ if, intuitively, for every input word x, we can permute the letters within each round in x, such that the output of $\mathcal{T}_2$ on the permuted word is itself a permutation of the output of $\mathcal{T}_1$ on x. Finally, two transducers are k-round equivalent if they simulate each other.

We solve two main decision problems, namely whether $\mathcal{T}_2$ k-round simulates $\mathcal{T}_1$ (1) when k is given as input, and (2) for an existentially quantified k.

We demonstrate the usefulness of the definitions by applying them to process symmetry: a setting in which a permutation in the identities of processes in a multi-process system naturally gives rise to two transducers, whose k-round equivalence corresponds to stability against such permutations.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Concurrency; Theory of computation → Abstraction

Keywords and phrases Transducers, Permutations, Parikh, Simulation, Equivalence

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.3

Related Version *Full Version*: `https://arxiv.org/abs/2105.01512`

Funding *Shaull Almagor*: European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837327.

1 Introduction

Reactive systems interact with their environment by receiving inputs, corresponding to the state of the environment, and sending outputs, which describe actions of the system. Finite-state reactive systems are often modeled by *transducers* – finite-state machines over alphabets Σ_I and Σ_O of inputs and outputs, respectively, which read an input letter in Σ_I, and respond with an output in Σ_O. Such transducers are amenable to automatic verification of certain properties (e.g., LTL model-checking), and are therefore useful in practice. Nonetheless, modeling complex systems may result in huge transducers, which make verification procedures prohibitively expensive, and makes understanding the constructed transducers difficult.

A common approach to gain a better understanding of a transducer (or more generally, any system) is *simulation* [19], whereby a transducer $\mathcal{T}_1$ is simulated by a "simpler" transducer $\mathcal{T}_2$ in such a way that model checking is easier on $\mathcal{T}_2$, and the correctness of the desired property is preserved under the simulation. Usually, "simpler" means smaller, as in standard simulation [19] and fair simulation [13], but one can also view e.g., linearization of concurrent programs [14] as a form of simulation by a simpler machine.

In this work, we introduce and study new notions of simulation and of equivalence for transducers, based on *k-rounds*: consider an input word $x \in \Sigma_I^*$ whose length is $k \cdot R$ for some $k, R > 0$. We divide the word into R disjoint infixes of length k, called k-rounds. We then say that two words $x, x' \in \Sigma_I^{kR}$ are k-round equivalent, denoted $x' \asymp_k x$, if x' is obtained from x by permuting the letters within each round of x. For example *abcabc* and *cbaacb* are 3-round equivalent. We now say that a transducer $\mathcal{T}_1$ is *k-round simulated* by a transducer $\mathcal{T}_2$, denoted $\mathcal{T}_1 \prec_k \mathcal{T}_2$, if for every[1] input $x \in \Sigma_I^{kR}$ we can find some input $x' \asymp_k x$, such that the outputs of $\mathcal{T}_1$ on x and $\mathcal{T}_2$ on x', denoted y, y' respectively, are also equivalent: $y' \asymp_k y$. Intuitively, $\mathcal{T}_1 \prec_k \mathcal{T}_2$ means that every behaviour of $\mathcal{T}_1$ is captured by $\mathcal{T}_2$, up to permutations within each k-round.

The benefit of k-round simulation is twofold: First, it may serve as an alternative simulation technique for reducing the state space while maintaining the correctness of certain properties. Second, we argue that k-round simulation is in and of itself a design concern. Indeed, in certain scenarios we can naturally design a transducer $\mathcal{T}_2$ that performs a certain task in an ideal, but not realistic, way, and we want to check that an existing design, namely $\mathcal{T}_1$, is simulated by this ideal. In particular, this is useful when dealing with systems that naturally work in rounds, such as schedulers (e.g., Round Robin, cf. Example 3), arbiters, and other resource allocation systems.

We start with an example demonstrating both benefits.

▶ **Example 1.** Consider a monitor M for the fairness of a distributed system with 10 processes $\mathcal{P} = \{1, \ldots, 10\}$. At each timestep, M receives as input the ID of the process currently working. The monitor then verifies that in each round of 10 steps, every process works exactly once. As long as this holds, the monitor keeps outputting `safe`, otherwise `error`.

M can be modeled by a transducer $\mathcal{T}_1$ that keeps track of the set of processes that have worked in the current round. Thus, the transducer has at least 2^{10} states, as it needs to keep track of the subset of processes that have been seen.

It is not hard to see that $\mathcal{T}_1$ is 10-round simulated by an "ideal" transducer $\mathcal{T}_2$ which expects to see the processes in the order $1, \ldots, 10$. This transducer needs roughly 10 states, as it only needs to know the index of the next process it expects to see.

Now, suppose we want to verify some correctness property which is invariant to permutations of the processes within each 10-round, such as "if there is no `error`, then Process 3 works at least once every 20 steps". Then we can verify this against the much smaller $\mathcal{T}_2$. ⌟

The notion of k-round simulation arises naturally in the setting of *process symmetry*. There, the input and output alphabets are $\Sigma_I = 2^I$ and $\Sigma_O = 2^O$ respectively, where $I = \{i_1, \ldots, i_m\}$ and $O = \{o_1, \ldots, o_m\}$ represent signals corresponding to m processes. Process symmetry addresses the scenario where the identities of the processes may be scrambled. For example, if the input $\{i_1, i_2\}$ is generated, the system might actually receive an input $\{i_7, i_4\}$. A system exhibits process symmetry if, intuitively, its outputs are permuted in a similar way to the inputs. Unfortunately, deterministic systems that are process symmetric are extremely naive, as process symmetry is too restrictive for them. While this can be overcome using probabilistic systems, as studied by the second author in [1], it is also desirable to find a definition that is suited for deterministic systems. As we show in Section 6, k-round simulation provides such a definition.

The main contributions of this work are as follows. We introduce the notion of k-round simulation and k-round equivalence, and define two decision problems pertaining to them: in *fixed round simulation* we need to decide whether $\mathcal{T}_1 \prec_k \mathcal{T}_2$ for a given value of k, and

[1] Our formal definition allows to also restrict the input to some regular language $\Lambda \subseteq \Sigma_I^*$, see Section 3.

in *existential round simulation* we need to decide whether there exists some value of k for which $\mathcal{T}_1 \prec_k \mathcal{T}_2$ holds. In fact, we consider a somewhat more elaborate setting, by also allowing the inputs to $\mathcal{T}_1$ to be restricted to some regular language Λ. We solve the first problem by reducing it to the containment of two nondeterministic automata. For the second problem, things become considerably more difficult, and the solution requires several constructions, as well as tools such as Presburger Arithmetic and Parikh's theorem. In addition, we demonstrate the usefulness of the definitions in relation to process symmetry.

Related Work. Simulation relations between systems are a well studied notion. We refer the reader to [7, Chapter 13] and references therein for an exposition. The connection of our notion with standard simulation is only up to motivation, as our measure is semantic and does not directly relate to the state space.

Technically, our work is closely related to *commutative automata* [4] and *jumping automata* [10, 18] – models of automata capable of reading their input in a discontinuous manner, by jumping from one letter to another. Indeed, our notion of round simulation essentially allows the simulating transducer to read the letters within rounds in a discontinuous manner. This similarity is manifested implicitly in Section 5.2, where we encounter similar structures as e.g., [15] (although the analysis here has a different purpose).

Finally, the initial motivation for this work comes from *process symmetry* [1, 6, 9, 16, 17]. We demonstrate the connections in depth in Section 6.

Paper organization. In Section 2 we present some basic definitions used throughout the paper. In Section 3 we introduce k-round simulation and equivalence, define the relevant decision problems, and study some fundamental properties of the definitions. In Section 4 we solve fixed round simulation, while developing some technical tools and characterizations that are reused later. Section 5 is our main technical result, where we develop a solution for existential round simulation. In particular, in Section 5.1 we give an overview of the solution, before going through the technical details in Section 5.2. In Section 5.3 we give lower bounds for the existential setting. In Section 6 we use round simulation to obtain a definition of process symmetry for deterministic transducers. Finally, in Section 7, we discuss some variants and open problems.

Due to lack of space, some proofs are omitted and can be found in the full version.

2 Preliminaries

Automata. A *deterministic finite automaton* (DFA) is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, and $F \subseteq Q$ is the set of accepting states.

The run of $\mathcal{A}$ on a word $w = \sigma_0 \cdot \sigma_2 \cdots \sigma_{n-1} \in \Sigma^*$ is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_{i+1} = \delta(q_i, \sigma_i)$ for all $0 \leq i < n$. The run is *accepting* if $q_n \in F$. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if the run of $\mathcal{A}$ on w is accepting. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We also consider *nondeterministic automata* (NFA), where $\delta : Q \times \Sigma \to 2^Q$. Then, a run of $\mathcal{A}$ on a word $w \in \Sigma^*$ as above is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $0 \leq i < n$. The language of $\mathcal{A}$ is defined analogously to the deterministic setting. We denote by $|\mathcal{A}|$ the number of states of $\mathcal{A}$.

As usual, we denote by δ^* the transition function lifted to words. For states q, q' and $w \in \Sigma^*$, we write $q \xrightarrow{w}_{\mathcal{A}} q'$ if $q' \in \delta^*(q, w)$. That is, if there is a run of $\mathcal{A}$ from q to q' while reading w.

An NFA $\mathcal{A}$ can be viewed as a morphism from Σ^* to the monoid $\mathbb{B}^{Q\times Q}$ of $Q \times Q$ Boolean matrices, where we associate with a letter $\sigma \in \Sigma$ its *type* $\tau_{\mathcal{A}}(\sigma) \in \mathbb{B}^{Q\times Q}$ defined by $(\tau_{\mathcal{A}}(\sigma))_{q,q'} = 1$ if $q \xrightarrow{\sigma}_{\mathcal{A}} q'$, and $(\tau_{\mathcal{A}}(\sigma))_{q,q'} = 0$ otherwise. We extend this to Σ^* by defining, for a word $w = \sigma_1 \cdots \sigma_n \in \Sigma^*$, its type as $\tau_{\mathcal{A}}(w) = \tau_{\mathcal{A}}(\sigma_1) \cdots \tau_{\mathcal{A}}(\sigma_n)$ where the concatenation denotes Boolean matrix product. It is easy to see that $(\tau_{\mathcal{A}}(w))_{q,q'} = 1$ iff $q \xrightarrow{w}_{\mathcal{A}} q'$.

Transducers. Consider two sets Σ_I and Σ_O, representing Input and Output alphabets, respectively. A Σ_I/Σ_O *transducer* is $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \boldsymbol{\ell}\rangle$ where Q, $q_0 \in Q$, and $\delta : Q \times \Sigma_I \to Q$ are as in a DFA, and $\boldsymbol{\ell} : Q \to \Sigma_O$ is a labelling function. For a word $w \in \Sigma_I^*$, consider the run $\rho = q_0, \ldots, q_n$ of $\mathcal{T}$ on a word w. We define its output $\boldsymbol{\ell}(\rho) = \boldsymbol{\ell}(q_1) \cdots \boldsymbol{\ell}(q_n) \in \Sigma_O^*$, and we define the output of $\mathcal{T}$ on w to be $\mathcal{T}(w) = \boldsymbol{\ell}(\rho)$. Observe that we ignore the labelling of the initial state in the run, so that the length of the output matches that of the input.

For a transducer $\mathcal{T}$ and a state s, we denote by $\mathcal{T}^s$ the transducer $\mathcal{T}$ with initial state s.

Words and Rounds. Consider a word $w = \sigma_0 \cdots \sigma_{n-1} \in \Sigma^*$. We denote its length by $|w|$, and for $0 \le i \le j < |w|$, we define $w[i : j] = \sigma_i \cdots \sigma_j$. For $k > 0$, we say that w is a *k-round word* if $|w| = kR$ for some $R \in \mathbb{N}$. Then, for every $0 \le r < R$, we refer to $w[rk : r(k+1) - 1]$ as the *r-th round* in w, and we write $w = \gamma_0 \cdots \gamma_{R-1}$ where γ_r is the r-th round. We emphasize that k signifies the length of each round, not the number of rounds.

In particular, throughout the paper we consider k-round words $(x, y) \in (\Sigma_I^k \times \Sigma_O^k)^*$. In such cases, we sometimes use the natural embedding of $(\Sigma_I^k \times \Sigma_O^k)^*$ in $(\Sigma_I \times \Sigma_O)^*$ and in $\Sigma_I^* \times \Sigma_O^*$, and refer to these sets interchangeably.

Parikh Vectors and Permutations. Consider an alphabet Σ. For a word $w \in \Sigma^*$ and a letter $\sigma \in \Sigma$, we denote by $\#_\sigma(w)$ the number of occurrences of σ in w. The Parikh map[2] $\mathfrak{P} : \Sigma^* \to \mathbb{N}^\Sigma$ maps every word $w \in \Sigma^*$ to a *Parikh vector* $\mathfrak{P}(w) \in \mathbb{N}^\Sigma$, where $\mathfrak{P}(w)(\sigma) = \#_\sigma(w)$. We lift this to languages by defining, for $L \subseteq \Sigma^*$, $\mathfrak{P}(L) = \{\mathfrak{P}(w) : w \in L\}$.

For $\boldsymbol{p} \in \mathbb{N}^\Sigma$ (in the following we consistently denote vectors in $\mathbb{N}^\Sigma$ by bold letters) we write $|\boldsymbol{p}| = \sum_{\sigma\in\Sigma} \boldsymbol{p}(\sigma)$. In particular, for a word $w \in \Sigma^*$ we have $|\mathfrak{P}(w)| = |w|$.

By Parikh's Theorem [21], for every NFA $\mathcal{A}$ we have that $\mathfrak{P}(L(\mathcal{A}))$ is a *semilinear set*, namely a finite union of sets of the form $\{\boldsymbol{p} + \lambda_1 \boldsymbol{s_1} + \ldots + \lambda_1 \boldsymbol{s_m} : \lambda_1, \ldots, \lambda_m \in \mathbb{N}\}$ where $\boldsymbol{p}, \boldsymbol{s_1}, \ldots, \boldsymbol{s_m} \in \mathbb{N}^d$ (and the translation is effective).

Consider words $x, y \in \Sigma^*$, we say that x is a *permutation* of y if $\mathfrak{P}(x) = \mathfrak{P}(y)$ (indeed, in this case y can be obtained from x by permuting its letters). Note that in particular this implies $|x| = |y|$.

3 Round Simulation and Round Equivalence

Consider two k-round words $x, y \in \Sigma^{kR}$ with the same number of rounds R, and denote their rounds by $x = \alpha_0 \cdots \alpha_{R-1}$ and $y = \beta_0 \cdots \beta_{R-1}$. We say that x and y are *k-round equivalent*, denoted $x \asymp_k y$ (or $x \asymp y$, when k is clear from context)[3], if for every $0 \le r < R$ we have that $\mathfrak{P}(\alpha_r) = \mathfrak{P}(\beta_r)$. That is, $x \asymp y$ iff the r-th round of y is a permutation of the r-th round of x, for every r. Clearly $\asymp$ is indeed an equivalence relation.

[2] We use $\mathbb{N}^\Sigma$ rather than $\mathbb{N}^{|\Sigma|}$ to emphasize that the vector's indices are the letters in Σ.
[3] Conveniently, our symbol for round equivalence is a rounded equivalence.

▶ **Example 2** (Round-equivalence for words)**.** Consider the words $x = abaabbabbbaa$ and $y = baabbaabbaba$ over the alphabet $\Sigma = \{a, b\}$. Looking at the words as 3-round words, one can see in Table 1 that the 3-rounds in y are all permutations of those in x, which gives $x \asymp_3 y$. However, looking at x, y as 4-round words, the number of occurrences of b already in the first 4-round of x and of y is different, so $x \not\asymp_4 y$, as illustrated in Table 2. ⌟

Table 1 x and y are 3-round equivalent.

x	aba	abb	abb	baa
y	baa	bba	abb	aba

Table 2 x and y are not 4-round equivalent.

x	*abaa*	bbab	bbaa
y	*baab*	baab	baba

Let Σ_I and Σ_O be input and output alphabets, let $\Lambda \subseteq \Sigma_I^*$ be a regular language, and let $k > 0$. Consider two Σ_I/Σ_O transducers $\mathcal{T}_1$ and $\mathcal{T}_2$. We say that $\mathcal{T}_2$ *k-round simulates* $\mathcal{T}_1$ *restricted to* Λ, denoted $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, if for every k-round word $x \in \Lambda$ there exists a k-round word $x' \in \Sigma_I^*$ such that $x \asymp_k x'$ and $\mathcal{T}_1(x) \asymp_k \mathcal{T}_2(x')$.

Intuitively, $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ if for every input word $x \in \Lambda$, we can permute each k-round of x to obtain a new word x', such that the k-rounds of the outputs of $\mathcal{T}_1$ on x and of $\mathcal{T}_2$ on x' are permutations of each other. Note that the definition is not symmetric: the input x for $\mathcal{T}_1$ is universally quantified, while x' is chosen according to x. We illustrate this in Example 4.

If $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ and $\mathcal{T}_2 \prec_{k,\Lambda} \mathcal{T}_1$ we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *k-round equivalent restricted to* Λ, denoted $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$. In the special case where $\Lambda = \Sigma_I^*$ (i.e., when we require the simulation to hold for every input), we omit it from the subscript and write e.g., $\mathcal{T}_1 \prec_k \mathcal{T}_2$.

▶ **Example 3** (Round Robin)**.** We consider a simple version of the Round Robin scheduler for three processes $\mathcal{P} = \{0, 1, 2\}$. In each time step, the scheduler outputs either a singleton set containing the ID of the process whose request is granted, or an empty set if the process whose turn it is did not make a request. Depending on the ID $i \in \{0, 1, 2\}$ of the first process, we model the scheduler as a $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer $\mathcal{T}_i = \langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, Q, q_{(i-1)\%3}, \delta, \boldsymbol{\ell} \rangle$ depicted in Figure 1, where % is the mod operator, $Q = \{q_0, q_1, q_2, q_0', q_1', q_2'\}$, $\delta(q_i, \sigma) = q_{(i+1)\%3}$ if $i + 1 \in \sigma$ and $\delta(q_i, \sigma) = q'_{(i+1)\%3}$ otherwise, $\boldsymbol{\ell}(q_i) = \{i\}$ and $\boldsymbol{\ell}(q_i') = \emptyset$.

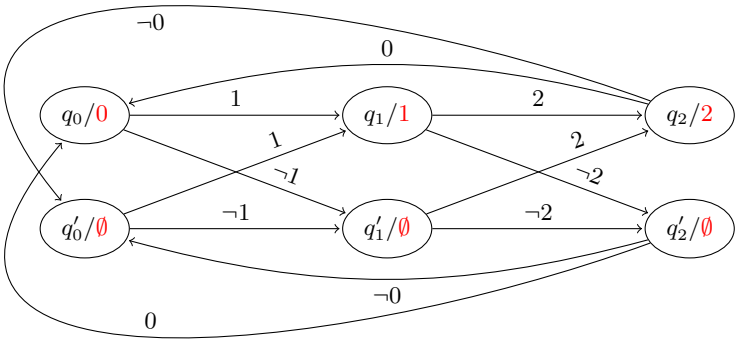


Figure 1 The transducer $\mathcal{T}_i$ for Round Robin, initial state omitted. The input letters σ and $\neg\sigma$ mean all input letters from $2^{\mathcal{P}}$ that, respectively, contain or do not contain σ. The labels are written in red on the states, singleton brackets omitted (e.g., 1 means $\{1\}$).

Technically, the initial state changes the behaviour of $\mathcal{T}_i$ significantly (e.g. $\mathcal{T}_0(\{0\}\{2\}\{1\}) = \{0\}\emptyset\emptyset$ whereas $\mathcal{T}_1(\{0\}\{2\}\{1\}) = \emptyset\{2\}\emptyset$). Conceptually, however, changing the initial state does not alter the behaviour, as long as the requests are permuted accordingly. This is captured by round equivalence, as follows.

We argue that, if we allow reordering of the input letters, then the set of processes whose requests are granted in each round is independent of the start state. This is equivalent to saying $\mathcal{T}_0 \equiv_k \mathcal{T}_j$ for $j \in \{1, 2\}$, which indeed holds: if $j = 1$ then we permute all rounds of the form $\sigma_0\sigma_1\sigma_2$ to $\sigma_1\sigma_2\sigma_0$, and similarly if $j = 2$ then we permute all rounds to $\sigma_2\sigma_0\sigma_1$. It is easy to see that the run of $\mathcal{T}_i$ on the permuted input grants outputs that are permutations of the output of $\mathcal{T}_0$ on the non-permuted input. ⌟

▶ **Example 4** (Round simulation is not symmetric)**.** Consider the Σ_I/Σ_O transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over the alphabet $\Sigma_I = \{a, b\}$ and $\Sigma_O = \{0, 1\}$, depicted in Figure 2. We claim that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$

Figure 2 Transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) illustrate the asymmetry in the definition of round equivalence (see Example 4).

but $\mathcal{T}_2 \not\prec_2 \mathcal{T}_1$. Starting with the latter, observe that $\mathcal{T}_2(ab) = 00$, but $\mathcal{T}_1(ab) = \mathcal{T}_1(ba) = 01$. Since $00 \not\succ_2 01$, we have $\mathcal{T}_2 \not\prec_2 \mathcal{T}_1$.

We turn to show that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$. Observe that for every input word of the form $x \in (ab+ba)^m$, we have $\mathcal{T}_1(x) = (01)^m$, and $x \succ_2 (ba)^m$. So in this case we have that $\mathcal{T}_2((ba)^m) = (10)^m \succ_2 (01)^m$. Next, for $x \in (ab+ba)^m \cdot bb \cdot w$ for some $w \in \Sigma_I^*$ we have $\mathcal{T}_1(x) = (01)^m 011^{|w|}$ and $x \succ_2 (ba)^m \cdot bb \cdot w$, for which $\mathcal{T}_2((ba)^m \cdot bb \cdot w) = (01)^m 101^{|w|} \succ_2 \mathcal{T}_1(x)$. The case where $x \in (ab + ba)^m \cdot aa \cdot w$ is handled similarly. We conclude that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$. ⌟

Round simulation and round equivalence give rise to the following decision problems:

- In *fixed round simulation* (resp. *fixed round equivalence*) we are given transducers $\mathcal{T}_1, \mathcal{T}_2$, an NFA for the language Λ, and $k > 0$, and we need to decide whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ (resp. whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$).
- In *existential round simulation* (resp. *existential round equivalence*) we are given transducers $\mathcal{T}_1, \mathcal{T}_2$ and an NFA for the language Λ, and we need to decide whether there exists $k > 0$ such that $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ (resp. $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$).

In the following we identify Λ with an NFA (or DFA) for it, as we do not explicitly rely on its description.

We start by showing that deciding equivalence (both fixed and existential) is reducible, in polynomial time, to the respective simulation problem.

▶ **Lemma 5.** *Fixed (resp. existential) round equivalence is reducible in polynomial time to fixed (resp. existential) round simulation.*

Proof. First, we can clearly reduce fixed round equivalence to fixed round simulation: given an algorithm that decides, given $\mathcal{T}_1, \mathcal{T}_2$, Λ and $k > 0$, whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, we can decide whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$ by using it twice to decide whether both $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ and $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ hold.

A slightly more careful examination shows that the same approach can be taken to reduce existential round equivalence to existential round simulation, using the following observation: if $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, then for every $m \in \mathbb{N}$ it holds that $\mathcal{T}_1 \prec_{mk,\Lambda} \mathcal{T}_2$. Indeed, we can simply group every m rounds of length k and treat them as a single mk-round.

Now, given an algorithm that decides, given $\mathcal{T}_1, \mathcal{T}_2$ and Λ, whether there exists $k > 0$ such that $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, we can decide whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$ by using the algorithm twice to decide whether there exists k_1 such that $\mathcal{T}_1 \prec_{k_1,\Lambda} \mathcal{T}_2$ and k_2 such that $\mathcal{T}_1 \prec_{k_2,\Lambda} \mathcal{T}_2$ hold. If there are no such k_1, k_2, then clearly $\mathcal{T}_1 \not\equiv_{k,\Lambda} \mathcal{T}_2$. However, if there are such k_1, k_2, then by the observation above we have $\mathcal{T}_1 \equiv_{k_1 k_2,\Lambda} \mathcal{T}_2$ (we can also take $\mathrm{lcm}(k_1, k_2)$ instead of $k_1 k_2$). ◀

By Lemma 5, for the purpose of upper-bounds, we focus henceforth on round simulation.

4 Deciding Fixed Round Simulation

In this section we show decidability of fixed round simulation (and, by Lemma 5, fixed round equivalence). The tools we develop will be used in Section 5 to handle the existential variant.

Let Σ_I and Σ_O be input and output alphabets. Consider two Σ_I/Σ_O transducers $\mathcal{T}_1$ and $\mathcal{T}_2$, and let $\Lambda \subseteq \Sigma_I^*$ and $k > 0$. In order to decide whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, we proceed as follows. First, we cast the problem to a problem about deterministic automata. Then, we translate k-rounds into letters, by working over the alphabets Σ_I^k and Σ_O^k. We construct an NFA, dubbed the *permutation closure*, for each transducer $\mathcal{T}$, that captures the behaviour of $\mathcal{T}$ on words and their permutations. Intuitively, the NFA takes as input a word $(x, y) \in (\Sigma_I^k \times \Sigma_O^k)^*$, guesses a round-equivalent word $x' \asymp x$, and verifies that $\mathcal{T}(x') \asymp \mathcal{T}(x)$. We then show that round-simulation amounts to deciding the containment of these NFAs. We now turn to give the details of the construction.

The Trace DFA. Consider a transducer $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \ell \rangle$, we define its *trace* DFA $\mathtt{Tr}(\mathcal{T}) = \langle \Sigma_I \times \Sigma_O, Q \cup \{q_\perp\}, q_0, \eta, Q \rangle$ where for $q \in Q$ and $(\sigma, \sigma') \in \Sigma_I \times \Sigma_O$ we define $\eta(q, (\sigma, \sigma')) = \delta(q, \sigma)$ if $\mathcal{T}^q(\sigma) = \sigma'$ and $\eta(q, (\sigma, \sigma')) = q_\perp$ otherwise. $q_\perp$ is a rejecting sink.

$\mathtt{Tr}(\mathcal{T})$ captures the behaviour of $\mathcal{T}$ in that $L(\mathtt{Tr}(\mathcal{T})) = \{(x, y) \in (\Sigma_I \times \Sigma_O)^* : \mathcal{T}(x) = y\}$.

The Permutation-Closure NFA. Consider an NFA $\mathcal{N} = \langle \Sigma_I \times \Sigma_O, S, s_0, \eta, F \rangle$, and let $k > 0$. We obtain from $\mathcal{N}$ an NFA $\mathtt{Perm}_k(\mathcal{N}) = \langle \Sigma_I^k \times \Sigma_O^k, S, s_0, \mu, F \rangle$ where the alphabet is $\Sigma_I^k \times \Sigma_O^k$, and the transition function μ is defined as follows. For a letter $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ and a state $s \in S$, we think of (α, β) as a word in $(\Sigma_I \times \Sigma_O)^*$. Then we have

$$\mu(s, (\alpha, \beta)) = \bigcup \{\eta^*(s, (\alpha', \beta')) : \mathfrak{P}(\alpha') = \mathfrak{P}(\alpha) \wedge \mathfrak{P}(\beta) = \mathfrak{P}(\beta')\}. \tag{1}$$

That is, upon reading (α, β), $\mathtt{Perm}_k(\mathcal{N})$ can move to any state s' that is reachable in $\mathcal{N}$ from s by reading a permutation of α, β (denoted α', β'). Recall that for two words x, x' we have that $x \asymp_k x'$ if for every two corresponding k-rounds α, α' in x and x' we have $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha')$. Thus, we have the following.

▶ **Observation 6.** $L(\mathtt{Perm}_k(\mathcal{N})) = \{(x, y) \in \Sigma_I^* \times \Sigma_O^* : \exists x' \asymp_k x, y' \asymp_k y, (x', y') \in L(\mathcal{N}) \wedge |x| = |y| = kR \text{ for some } R \in \mathbb{N}\}$

Since the transition function of $\mathtt{Perm}_k(\mathcal{N})$ is only defined using permutations of its input letters, we have the following property, which we refer to as *permutation invariance*:

▶ **Observation 7** (Permutation Invariance). *For every state $s \in S$ and letters $(\alpha, \beta), (\alpha', \beta') \in \Sigma_I^k \times \Sigma_O^k$, if $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha')$ and $\mathfrak{P}(\beta) = \mathfrak{P}(\beta')$ then $\mu(s, (\alpha, \beta)) = \mu(s, (\alpha', \beta'))$.*

Given a transducer $\mathcal{T}$, we apply the permutation closure to the trace DFA of $\mathcal{T}$. In order to account for $\Lambda \subseteq \Sigma_I^*$, we identify it with $\Lambda \subseteq \Sigma_I^* \times \Sigma_O^*$ by simply ignoring the Σ_O component. We remind that Λ denotes both a language and a corresponding DFA or NFA.

▶ **Lemma 8.** *Consider transducers $\mathcal{T}_1, \mathcal{T}_2$, an* NFA Λ *and $k > 0$. Let $\mathcal{A}_1^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ (where the intersection is obtained by the product* NFA*) and $\mathcal{A}_2^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$, then*

$$L(\mathcal{A}_1^k) = \{(x,y) \in \Sigma_I^* \times \Sigma_O^* \,:\, \exists x' \asymp_k x,\ \mathcal{T}_1(x') \asymp_k y \ \wedge\ |x| = |y| = kR \text{ where } R \in \mathbb{N} \wedge x' \in \Lambda\}.$$
$$L(\mathcal{A}_2^k) = \{(x,y) \in \Sigma_I^* \times \Sigma_O^* \,:\, \exists x' \asymp_k x,\ \mathcal{T}_2(x') \asymp_k y \ \wedge\ |x| = |y| = kR \text{ where } R \in \mathbb{N}\}.$$

Proof. Recall that $\mathtt{Tr}(\mathcal{T})$ accepts a word (x', y') iff $\mathcal{T}(x') = y'$. The claim then follows from Observation 6, by replacing the expression $y \asymp y' \wedge (x', y') \in L(\mathtt{Tr}(\mathcal{T}))$ with the equivalent expression $\mathcal{T}(x') \asymp_k y$. ◀

We now reduce round simulation to the containment of permutation-closure NFAs.

▶ **Lemma 9.** *Consider transducers $\mathcal{T}_1, \mathcal{T}_2$, an* NFA Λ *and $k > 0$. Let $\mathcal{A}_1^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ and $\mathcal{A}_2^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$, then, $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ iff $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$.*

Proof. For the first direction, assume $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, and let $(x,y) \in L(\mathcal{A}_1^k)$. By Lemma 8, x and y are k-round words, and there exists a word $x' \in \Lambda$ such that $x \asymp x'$ and $\mathcal{T}_1(x') \asymp y$. Since $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, then applying the definition on x' yields that there exists a k-round word x'' such that $x' \asymp x''$ and such that $\mathcal{T}_1(x') \asymp \mathcal{T}_2(x'')$. Since $\asymp$ is an equivalence relation, it follows that $x \asymp x''$ and $\mathcal{T}_2(x'') \asymp y$, so again by Lemma 8 we have $(x,y) \in L(\mathcal{A}_2^k)$.

Conversely, assume $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$, we wish to prove that for every k-round word $x \in \Lambda$ there exists a word x' such that $x \asymp x'$ and $\mathcal{T}_1(x) \asymp \mathcal{T}_2(x')$. Let $x \in \Lambda$ be a k-round word, and let $y = \mathcal{T}_1(x)$, then clearly $(x,y) \in L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$ (since $x \asymp x$, $\mathcal{T}_1(x) = y \asymp y$ and $x \in \Lambda$). By Lemma 8, there exists x' such that $x \asymp x'$ and $\mathcal{T}_2(x') \asymp y = \mathcal{T}_1(x)$, so $\mathcal{T}_2(x') \asymp \mathcal{T}_1(x)$, thus concluding the proof. ◀

▶ Remark 10. The proof of Lemma 9 can be simplified by using instead of $\mathcal{A}_1^k$, the augmentation of $\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda$ to k-round words. However, such a DFA is not permutation invariant, which is key to our solution for existential round simulation. Since this simplification does not reduce the overall complexity, we use a uniform setting for both solutions.

Lemma 9 shows that deciding fixed round equivalence amounts to deciding containment of NFAs. By analyzing the size of the NFAs, we obtain the following.

▶ **Theorem 11.** *Given transducers $\mathcal{T}_1, \mathcal{T}_2$, an* NFA Λ*, and $k > 0$ in unary, the problem of deciding whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ is in* PSPACE.

Proof. Let $\mathcal{A}_1^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ and $\mathcal{A}_2^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$. By Lemma 9, deciding whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ amounts to deciding whether $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$. Looking at the dual problem, recall that for two NFAs $\mathcal{N}_1, \mathcal{N}_2$ we have that $L(\mathcal{N}_1) \not\subseteq L(\mathcal{N}_2)$ iff there exists $w \in L(\mathcal{N}_2) \setminus L(\mathcal{N}_1)$ with $|w| \leq |\mathcal{N}_1| \cdot 2^{|\mathcal{N}_2|}$ (this follows immediately by bounding the size of an NFA for $L(\mathcal{N}_1) \cap \overline{L(\mathcal{N}_2)}$). Thus, we can decide whether $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$ by guessing a word w over $\Sigma_I^k \times \Sigma_O^k$ of single-exponential length (in the size of $\mathcal{A}_1^k$ and $\mathcal{A}_2^k$), and verifying that it is accepted by $\mathcal{A}_1^k$ and not by $\mathcal{A}_2^k$.

Observe that to this end, we do not explicitly construct $\mathcal{A}_1^k$ nor $\mathcal{A}_2^k$, as their alphabet size is exponential. Rather, we evaluate them on each letter of w based on their construction from $\mathcal{T}$. At each step we keep track of a counter for the length of w, a state of $\mathcal{A}_1^k$, and a set of states of $\mathcal{A}_2^k$. Since the number of states in $\mathcal{A}_1^k$ and $\mathcal{A}_2^k$ is the same as that of $\mathcal{T}_1$ and $\mathcal{T}_2$, this requires polynomial space.

By Savitch's theorem we have that coNPSPACE = PSPACE, and the proof is concluded. ◀

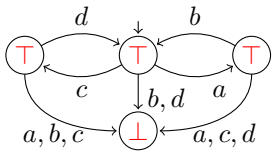


Figure 3 The transducer $\mathcal{T}_1$ in the proof of Theorem 12.

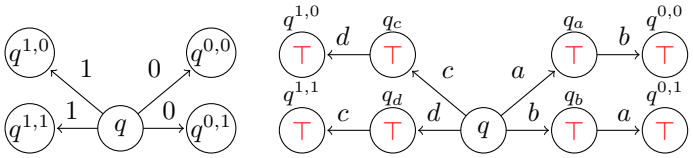


Figure 4 Every state and its 4 transitions in $\mathcal{N}$ (left) turn into 8 transitions in $\mathcal{T}_2$ (right). All transitions not drawn in the right figure lead to $q_\perp$, a sink state labelled $\perp$.

We now give a PSPACE-hardness lower bound, thus concluding the problem is PSPACE-complete. By Lemma 5, we give a stronger lower bound already for round-equivalence.[4]

▶ **Theorem 12.** *The problem of deciding, given transducers $\mathcal{T}_1, \mathcal{T}_2$, whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$, is* PSPACE*-hard, even for $k = 2$ and for a fixed Λ (given as a 4-state* DFA*).*

Proof sketch. We show a reduction from the universality problem for NFAs over alphabet $\{0,1\}$ where all states are accepting and the degree of nondeterminism is at most 2, to round-equivalence with $k = 2$ and with Λ given as a DFA of constant size. See the full version for a proof of PSPACE-hardness of the former problem, and for the full reduction.

Consider an NFA $\mathcal{N} = \langle Q, \{0,1\}, \delta, q_0, Q\rangle$ where $|\delta(q,\sigma)| \leq 2$ for every $q \in Q$ and $\sigma \in \{0,1\}$.

We construct two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over input and output alphabets $\Sigma_I = \{a,b,c,d\}$ and $\Sigma_O = \{\top, \perp\}$ and $\Lambda \subseteq \Sigma_I^*$, such that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$.

Set $\Lambda = (ab + cd)^*$. Intuitively, our reduction encodes $\{0,1\}$ over $\{a,b,c,d\}$ by identifying 0 with ab and with ba, and 1 with cd and with dc. Then, $\mathcal{T}_1$ (Figure 3) keeps outputting $\top$ for all inputs in Λ, thus mimicking a universal language in $\{0,1\}^*$. We then construct $\mathcal{T}_2$ so that every nondeterministic transition of $\mathcal{N}$ on e.g., 0 is replaced by two deterministic branches on ab and on ba (see Figure 4). Hence, when we are allowed to permute ab and ba by round equivalence, we capture the nondeterminism of $\mathcal{N}$. The outputs in $\mathcal{T}_2$ are all $\top$, except a sink state $q_\perp$ labelled $\perp$, which is reached upon any undefined transition (including transitions from states of $\mathcal{N}$ that do not have an outgoing 0 or 1 transition).

We show that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$, by showing that $\mathcal{T}_2 \prec_{2,\Lambda} \mathcal{T}_1$ always holds, and that for the converse, namely $\mathcal{T}_1 \prec_{2,\Lambda} \mathcal{T}_2$, permuting an input word $w \in \Lambda$ essentially amounts to choosing an accepting run of $\mathcal{N}$ on the corresponding word in $\{0,1\}^*$. ◀

▶ **Corollary 13.** *Given transducers $\mathcal{T}_1, \mathcal{T}_2$, an* NFA *$\Lambda$, and $k > 0$ in unary, the problem of deciding whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ is* PSPACE*-complete.*

5 Deciding Existential Round Simulation

We turn to solve existential round simulation. That is, given $\mathcal{T}_1, \mathcal{T}_2$ and Λ, we wish to decide whether there exists $k > 0$ such that $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$. By Lemma 9, this is equivalent to deciding whether there exists $k > 0$ such that $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$, as defined therein.

[4] The reduction in Lemma 5 is a Turing reduction. Nonetheless, our PSPACE-hardness proof actually explicitly shows the hardness of both simulation and equivalence.

5.1 Intuitive Overview

We start with an intuitive explanation of the solution and its challenges. For simplicity, assume for now $\Lambda = \Sigma_I^*$, so it can be ignored. The overall approach is to present a small-model property for k: in Theorem 14, the main result of this section, we give an upper bound on the minimal $k > 0$ for which $\mathcal{T}_1 \prec_k \mathcal{T}_2$. In order to obtain this bound, we proceed as follows. Observe that for a transducer $\mathcal{T}$ and for $0 < k \neq k'$ the corresponding permutation closure NFAs $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}))$ are defined on the same state space, but differ by their alphabet ($\Sigma_I^k \times \Sigma_O^k$ vs $\Sigma_I^{k'} \times \Sigma_O^{k'}$). Thus, by definition, these NFAs form infinitely many distinct automata. Nonetheless, there are only finitely many possible types of letters (indeed, at most $|\mathbb{B}^{Q \times Q}| = 2^{|Q|^2}$). Therefore, there are only finitely many *type profiles* for NFAs (namely the set of letter types occurring in the NFA), up to multiplicities of the letter types.

Recall that by Lemma 9, we have that $\mathcal{T}_1 \prec_k \mathcal{T}_2$ iff $L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1))) \subseteq L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2)))$. Intuitively, one could hope that if $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_i))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_i))$ have the same type profile, for each $i \in \{1, 2\}$, then $L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1))) \subseteq L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2)))$ iff $L(\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_1))) \subseteq L(\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_2)))$. Then, if one can bound the index k after which no further type profiles are encountered, then the problem reduces to checking a finite number of containments.

Unfortunately, this is not the case, the reason being that the mapping of letters induced by the equal type profiles between $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_1))$ may differ from the one between $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_2))$, and thus one cannot translate language containment between the two pairs. We overcome this difficulty, however, by working from the start with product automata that capture the structure of both $\mathcal{T}_1$ and $\mathcal{T}_2$ simultaneously, and thus unify the letter mapping.

We are now left with the problem of bounding the minimal k after which all type profiles have been exhausted. In order to provide this bound, we show that for every type profile, the set of indices in which it occurs is semilinear. Then, by finding a bound for each type profile, we attain the overall bound. The main result of this section is the following.

▶ **Theorem 14.** *Given transducers $\mathcal{T}_1, \mathcal{T}_2$ and Λ, we can effectively compute $K_0 > 0$ such that if $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ for some $k \in \mathbb{N}$, then $\mathcal{T}_1 \prec_{k',\Lambda} \mathcal{T}_2$ for some $k' \leq K_0$.*

Which by Lemma 9 immediately entails the following.

▶ **Corollary 15.** *Existential round simulation is decidable.*

We prove Theorem 14 in Section 5.2, organized as follows. We start by lifting the definition of *types* in an NFA to Parikh vectors, and show how these relate to the NFA (Lemma 16). We then introduce Presburger Arithmetic and its relation to Parikh's theorem. In Lemma 17 we show that the set of Parikh vectors that share a type τ is definable in Presburger arithmetic, which provides the first main step towards our bound.

We then proceed to define the "redundant products", which are the product automata mentioned above, that serve to unify the types between $\mathcal{T}_1$ and $\mathcal{T}_2$. In Observations 18 and 19 we formalize the connection of these products to the transducers $\mathcal{T}_1, \mathcal{T}_2$. We then define the type profiles mentioned above, and prove in Lemma 20 that they exhibit a semilinear behaviour. Finally, in Lemma 21 we prove that when two redundant-product automata have the same type profile, then the containment mentioned above can be shown. We conclude by combining these results to obtain Theorem 14.

5.2 Proof of Theorem 14

Type Matrices of Parikh Vectors. Consider the alphabet $\Sigma_I^k \times \Sigma_O^k$ for some $k > 0$. Recall that by Observation 7, permutation closure NFAs are permutation invariant, and from Section 2, the *type* of a word in an NFA is the transition matrix it induces. In particular, for permutation-invariant NFAs, two letters $(\alpha, \beta), (\alpha', \beta') \in \Sigma_I^k \times \Sigma_O^k$ with $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha')$ and $\mathfrak{P}(\beta) = \mathfrak{P}(\beta')$ have the same type.

Following this, we now lift the definition of types to Parikh vectors. Consider an NFA $\mathcal{N} = \langle \Sigma_I \times \Sigma_O, S, s_0, \eta, F \rangle$, and let $\boldsymbol{p} \in \mathbb{N}^{\Sigma_I}, \boldsymbol{o} \in \mathbb{N}^{\Sigma_O}$ be Parikh vectors with $|\boldsymbol{p}| = |\boldsymbol{o}| = k$. We define the type $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) \in \mathbb{B}^{S \times S}$ to be $\tau_{\mathtt{Perm}_k(\mathcal{N})}(\alpha, \beta)$ where $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ are such that $\mathfrak{P}(\alpha) = \boldsymbol{p}$ and $\mathfrak{P}(\beta) = \boldsymbol{o}$. By permutation invariance, this is well-defined, i.e. is independent of the choice of α and β.

Note that the definition of types is "non-uniform", since we use different automata to extract the type of words of different length. We obtain a more uniform description as follows (see the full version for the proof).

▶ **Lemma 16.** *In the notations above, for every $s_1, s_2 \in S$, we have that $(\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}))_{s_1, s_2} = 1$ iff there exist $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ with $\mathfrak{P}(\alpha) = \boldsymbol{p}$ and $\mathfrak{P}(\beta) = \boldsymbol{o}$ such that $s_1 \xrightarrow{(\alpha,\beta)}_{\mathcal{N}} s_2$.*

Presburger Arithmetic. The first ingredient in the proof of Theorem 14 is to characterize the set of Parikh vectors whose type is some fixed matrix $\tau \in \mathbb{B}^{Q \times Q}$. For this characterization, we employ the first-order theory of the naturals with addition and order $\mathrm{Th}(\mathbb{N}, 0, 1, +, <, =)$, commonly known as Presburger Arithmetic (PA). We do not give a full exposition of PA, but refer the reader to [12] (and references therein) for a survey. In the following we briefly cite the results we need.

For our purposes, a PA formula $\varphi(x_1, \ldots, x_d)$, where $x_1, \ldots, x_d$ are free variables, is evaluated over $\mathbb{N}^d$, and *defines* the set $\{(a_1, \ldots, a_d) \in \mathbb{N}^d : (a_1, \ldots, a_d) \models \varphi(x_1, \ldots, x_d)\}$. For example, the formula $\varphi(x_1, x_2) := x_1 < x_2 \wedge \exists y. x_1 = 2y$ defines the set $\{(a, b) \in \mathbb{N}^2 : a < b \wedge a \text{ is even}\}$.

A fundamental result about PA states that the definable sets in PA are exactly the semilinear sets. In particular, by Parikh's theorem we have that for every NFA $\mathcal{A}$, $\mathfrak{P}(L(\mathcal{A}))$ is PA definable. In fact, by [22], one can efficiently construct a linear-sized existential PA formula for $\mathfrak{P}(L(\mathcal{A}))$. We can now show that the set of Parikh vectors whose type is τ is PA definable.

▶ **Lemma 17.** *Consider an* NFA $\mathcal{N} = \langle \Sigma_I \times \Sigma_O, S, s_0, \eta, F \rangle$, *and a type $\tau \in \mathbb{B}^{S \times S}$, then the set $\{(\boldsymbol{p}, \boldsymbol{o}) \in \mathbb{N}^{\Sigma_I} \times \mathbb{N}^{\Sigma_O} : \tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau\}$ is* PA *definable.*

Proof. Let $\tau \in \mathbb{B}^{S \times S}$, and consider a Parikh vector $(\boldsymbol{p}, \boldsymbol{o}) \in \mathbb{N}^{\Sigma_I} \times \mathbb{N}^{\Sigma_O}$ with $k = |\boldsymbol{p}| = |\boldsymbol{o}|$. By Lemma 16, we have that $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff the following holds for every $s_1, s_2 \in S$: we have $\tau_{s_1, s_2} = 1$ iff there exists a letter $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ such that $\mathfrak{P}(\alpha) = \boldsymbol{p}, \mathfrak{P}(\beta) = \boldsymbol{o}$, and $s_1 \xrightarrow{(\alpha,\beta)}_{\mathcal{N}} s_2$.

Consider $s_1, s_2 \in S$ and define $\mathcal{N}_{s_2}^{s_1}$ to be the NFA obtained from $\mathcal{N}$ by setting the initial state to be s_1 and a single accepting state s_2. Then, we have $s_1 \xrightarrow{(\alpha,\beta)}_{\mathcal{N}} s_2$ iff $(\alpha, \beta) \in L(\mathcal{N}_{s_2}^{s_1})$.

Thus, $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff for every $s_1, s_2 \in S$ we have that $\tau_{s_1, s_2} = 1$ iff there exist a word (α, β) with $\mathfrak{P}(\alpha') = \boldsymbol{p}$ and $\mathfrak{P}(\beta') = \boldsymbol{o}$ such that $(\alpha, \beta) \in L(\mathcal{N}_{s_2}^{s_1})$. Equivalently, we have $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff for every $s_1, s_2 \in S$ we have that $\tau_{s_1, s_2} = 1$ iff $(\boldsymbol{p}, \boldsymbol{o}) \in \mathfrak{P}(L(\mathcal{N}_{s_2}^{s_1}))$.

By Parikh's Theorem, for every $s_1, s_2 \in S$ we can compute a PA formula ψ_{s_1,s_2} such that $(\boldsymbol{p}, \boldsymbol{o}) \models \psi_{s_1,s_2}$ iff $(\boldsymbol{p}, \boldsymbol{o}) \in \mathfrak{P}(L(\mathcal{N}^{s_1}_{s_2}))$. Now we can construct a PA formula Ψ_τ such that $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff $(\boldsymbol{p}, \boldsymbol{o}) \models \Psi_\tau$, as follows:

$$\Psi_\tau := \bigwedge_{s_1,s_2 \,:\, \tau_{s_1,s_2}=1} \psi_{s_1,s_2} \wedge \bigwedge_{s_1,s_2 \,:\, \tau_{s_1,s_2}=0} \neg\psi_{s_1,s_2}.$$

Finally, observe that Ψ_τ defines the set in the premise of the lemma, so we are done. ◀

The Redundant Product Construction. As mentioned in Section 5.1, for the remainder of the proof we want to reason about the types of $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ and $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$ simultaneously. In order to so, we present an auxiliary product construction.

Let $\mathcal{T}_1, \mathcal{T}_2$ be transducers, $\Lambda \subseteq \Sigma_I^*$ be given by an NFA, and let $\mathcal{D}_1 = \mathtt{Tr}(\mathcal{T}_1) \cap \Lambda$ and $\mathcal{D}_2 = \mathtt{Tr}(\mathcal{T}_2)$. We now consider the product automaton of $\mathcal{D}_1$ and $\mathcal{D}_2$, and endow it with two different acceptance conditions, capturing that of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Formally, for $i \in \{1, 2\}$, denote $\mathcal{D}_i = \langle \Sigma_I \times \Sigma_O, S_i, s^i_0, \eta_i, F_i \rangle$, then the product automaton is defined as $\mathcal{B}_i = \langle \Sigma_I \times \Sigma_O, S_1 \times S_2, (s^1_0, s^2_0), \eta_1 \times \eta_2, G_i \rangle$, where $G_1 = F_1 \times Q_2$ and $G_2 = Q_1 \times F_2$, and $\eta_1 \times \eta_2$ denotes the standard product transition function, namely $\eta_1 \times \eta_2((s_1, s_2), (\sigma, \sigma')) = (\eta_1(s_1, (\sigma, \sigma')), \eta_2(s_2, (\sigma, \sigma')))$. Thus, $\mathcal{B}_i$ tracks both $\mathcal{D}_1$ and $\mathcal{D}_2$, but has the same acceptance condition as $\mathcal{D}_i$. This seemingly "redundant" product construction has the following important properties, which are crucial for our proof:

▶ **Observation 18.** *In the notations above, we have the following:*

1. $L(\mathcal{B}_1) = L(\mathcal{D}_1)$ *and* $L(\mathcal{B}_2) = L(\mathcal{D}_2)$.
2. *For every letter* $(\sigma, \sigma') \in \Sigma_I \times \Sigma_O$*, we have* $\tau_{\mathcal{B}_1}(\sigma, \sigma') = \tau_{\mathcal{B}_2}(\sigma, \sigma')$.

Indeed, Item 1 follows directly from the acceptance condition, and Item 2 is due to the identical transition function of $\mathcal{B}_1$ and $\mathcal{B}_2$.

By Observation 6, $L(\mathtt{Perm}_k(\mathcal{D}_i))$ depends only on $L(\mathcal{D}_i)$. We thus have the following.

▶ **Observation 19.** *For every* $k > 0$ *we have* $L(\mathtt{Perm}_k(\mathcal{B}_1)) = L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda))$ *and* $L(\mathtt{Perm}_k(\mathcal{B}_2)) = L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2)))$.

Type Profiles. We now consider the set of types induced by the redundant product constructions $\mathcal{B}_1$ and $\mathcal{B}_2$ on Parikh vectors of words of length k. By Item 2 of Observation 18, it's enough to consider $\mathcal{B}_1$.

For $k > 0$, we define the *k-th type profile* of $\mathcal{B}_1$ to be $\Upsilon(\mathcal{B}_1, k) = \{\tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha), \mathfrak{P}(\beta)) : (\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k\}$, i.e., the set of all types of Parikh vectors $(\boldsymbol{p}, \boldsymbol{o})$ with $|\boldsymbol{p}| = |\boldsymbol{o}| = k$ that are induced by $\mathcal{B}_1$. Clearly there is only a finite number of type profiles, as $\Upsilon(\mathcal{B}_1, k) \subseteq \mathbb{B}^{S' \times S'}$, where S' is the state space of $\mathcal{B}_1$. Therefore, as k increases, after some finite K_0, every distinct type profile will have been encountered. We now place an upper bound on K_0.

▶ **Lemma 20.** *We can effectively compute* $K_0 > 0$ *such that for every* $k > 0$ *there exists* $k' \leq K_0$ *with* $\Upsilon(\mathcal{B}_1, k') = \Upsilon(\mathcal{B}_1, k)$.

Proof. Consider a type τ, and let Ψ_τ be the PA formula constructed as per Lemma 17 for the NFA $\mathcal{B}_1$. Observe that for a Parikh vector $(\boldsymbol{p}, \boldsymbol{o})$ and for $k > 0$, the expression $|\boldsymbol{p}| = |\boldsymbol{o}| = k$ is PA definable. Indeed, writing $\boldsymbol{p} = (x_1, \ldots, x_{|\Sigma_I|})$ and $\boldsymbol{q} = (y_1, \ldots, y_{|\Sigma_O|})$, the expression is defined by $x_1 + \ldots + x_{|\Sigma_I|} = k \wedge y_1 + \ldots + y_{|\Sigma_O|} = k$.

Let $T \subseteq \mathbb{B}^{S' \times S'}$ be a set of types (i.e., a potential type profile). We define a PA formula $\Theta_T(z)$ over a single free variable z such that $k \models \Theta_T(z)$ iff $\Upsilon(\mathcal{B}_1, k) = T$, as follows.

$$\Theta_T(z) = \left(\forall \boldsymbol{p}, \boldsymbol{o}, |\boldsymbol{p}| = |\boldsymbol{o}| = z \rightarrow \bigvee_{\tau \in T} \Psi_\tau(\boldsymbol{p}, \boldsymbol{o}) \right) \wedge \left(\bigwedge_{\tau \in T} \exists \boldsymbol{p}, \boldsymbol{o}, |\boldsymbol{p}| = |\boldsymbol{o}| = z \wedge \Psi_\tau(\boldsymbol{p}, \boldsymbol{o}) \right)$$

Intuitively, $\Theta_T(z)$ states that every Parikh vector $(\boldsymbol{p}, \boldsymbol{o})$ with $|\boldsymbol{p}| = |\boldsymbol{o}| = z$ has a type within T, and that all the types in T are attained by some such Parikh vector.

By [11, 3], we can effectively determine, for every T, whether $\Theta_T(z)$ is satisfiable and if it is, find a witness M_T such that $M_T \models \Theta_T(z)$. By doing so for every set $T \subseteq \mathbb{B}^{S' \times S'}$, we can set $K_0 = \max\{M_T : \Theta_T(z) \text{ is satisfiable}\}$. Then, for every $k > K_0$ if $\Upsilon(\mathcal{B}_1, k) = T$, then T has already been encountered at $M_T \leq K_0$, as required. ◀

The purpose of the bound K_0 obtained in Lemma 20 is to bound the minimal k for which $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, or equivalently $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$ (by Lemma 9 and Observation 19). This is captured in the following.

▶ **Lemma 21.** *Let $0 < k \neq k'$ such that $\Upsilon(\mathcal{B}_1, k') = \Upsilon(\mathcal{B}_1, k)$, then $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$ iff $L(\mathtt{Perm}_{k'}(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$.*

Proof. By the symmetry between k and k', it suffices to prove w.l.o.g. that if $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$, then $L(\mathtt{Perm}_{k'}(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$.

Assume the former, and let $w = (x', y') \in L(\mathtt{Perm}_{k'}(\mathcal{B}_1))$, where $(x', y') \in (\Sigma_I^{k'} \times \Sigma_O^{k'})^*$, and we denote $(x', y') = (\alpha'_1, \beta'_1) \cdots (\alpha'_n, \beta'_n)$ with $(\alpha'_j, \beta'_j) \in \Sigma_I^{k'} \times \Sigma_O^{k'}$ for every $1 \leq j \leq n$.

Since $\Upsilon(\mathcal{B}_1, k') = \Upsilon(\mathcal{B}_1, k)$, there is a mapping φ that takes every letter (α'_j, β'_j) in w (over $\Sigma_I^{k'} \times \Sigma_O^{k'}$) to a letter $(\alpha_j, \beta_j) \in \Sigma_I^k \times \Sigma_O^k$ that has same type in $\mathtt{Perm}_k(\mathcal{B}_1)$, so that we can find $(x, y) = (\alpha_1, \beta_1) \cdots (\alpha_n, \beta_n)$ such that for every $1 \leq j \leq n$ we have $\tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha_j), \mathfrak{P}(\beta_j)) = \tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha'_j), \mathfrak{P}(\beta'_j))$.

By the definition of the type of a Parikh vector, we have that

$$\tau_{\mathtt{Perm}_k(\mathcal{B}_1)}(\alpha_j, \beta_j) = \tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha_j), \mathfrak{P}(\beta_j)) = \tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha'_j), \mathfrak{P}(\beta'_j)) = \tau_{\mathtt{Perm}_{k'}(\mathcal{B}_1)}(\alpha'_j, \beta'_j).$$

In particular, since the type of a word is the concatenation (i.e., Boolean matrix product) of its underlying letters, we have that $\tau_{\mathtt{Perm}_k(\mathcal{B}_1)}(x, y) = \tau_{\mathtt{Perm}_{k'}(\mathcal{B}_1)}(x', y')$. Since $(x', y') \in L(\mathtt{Perm}_{k'}(\mathcal{B}_1))$, it follows that also $(x, y) \in L(\mathtt{Perm}_k(\mathcal{B}_1))$. Indeed, $(\tau_{\mathtt{Perm}_{k'}(\mathcal{B}_1)}(x', y'))_{s_0^1, s_f^1} = 1$ where s_0^1 and s_f^1 are an initial state and an accepting state of $\mathtt{Perm}_{k'}(\mathcal{B}_1)$, respectively. But the equality of the types implies that $(\tau_{\mathtt{Perm}_k(\mathcal{B}_1)}(x, y))_{s_0^1, s_f^1} = 1$ as well, so $\mathtt{Perm}_k(\mathcal{B}_1)$ has an accepting run on (x, y).

By our assumption, $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$, so $(x, y) = \varphi(w) \in L(\mathtt{Perm}_k(\mathcal{B}_2))$, or equivalently, $\varphi(w) \in L(\mathtt{Perm}_k(\mathcal{B}_2))$. We now essentially reverse the arguments above, but with $\mathcal{B}_2$ instead of $\mathcal{B}_1$. However, this needs to be done carefully, so that the mapping of letters lands us back at (x', y'), and not a different word. Thus, instead of finding a Parikh-equivalent word, we observe that for every $1 \leq j \leq n$, we also have

$$\tau_{\mathtt{Perm}_k(\mathcal{B}_2)}(\alpha_j, \beta_j) = \tau_{\mathcal{B}_2}(\mathfrak{P}(\alpha_j), \mathfrak{P}(\beta_j)) = \tau_{\mathcal{B}_2}(\mathfrak{P}(\alpha'_j), \mathfrak{P}(\beta'_j)) = \tau_{\mathtt{Perm}_{k'}(\mathcal{B}_2)}(\alpha'_j, \beta'_j),$$

This follows from Item 2 in Observation 18 and the fact that the permutation construction depends only on the transitions (and not on accepting states, which are the only difference between $\mathcal{B}_1$ and $\mathcal{B}_2$).

Thus, similarly to the arguments above, we have that $(x', y') \in L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$, and the mapping applied is in fact the the inverse map φ^{-1}, where $\varphi^{-1}(\varphi(w)) = w$. We conclude that $L(\mathtt{Perm}_{k'}(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$, as required.

The mapping is illustrated in Figure 5. ◀

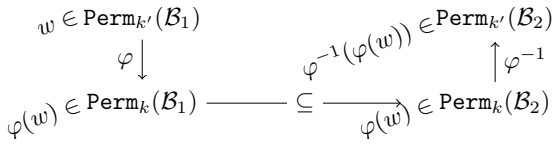


Figure 5 A diagram for the proof structure of Lemma 21.

Combining Lemmas 20 and 21, we can effectively compute K_0 such that if $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$ for some k, then this holds for some $k < K_0$. Finally, using Lemma 9, this concludes the proof of Theorem 14.

▶ Remark 22 (Complexity Results for Theorem 14 and Corollary 15). Let n be the number of states in $\mathcal{T}_1 \times \mathcal{T}_2$. Observe that the formula Ψ_τ constructed in Lemma 17 comprises a conjunction of $O(n^2)$ PA subformulas, where each subformula is either an existential PA formula of length $O(n)$, or the negation of one. Then, the formula Θ_T in Lemma 20 consists of a universal quantification, nesting a disjunction over $|T|$ formulas of the form Ψ_τ, conjuncted with $|T|$ existential quantifications, nesting a single Ψ_τ each. Overall, this amounts to a formula of length $|T| \leq 2^{n^2}$, with alternation depth 3. [5]

Using quantifier elimination [8, 20], we can obtain a witness for the satisfiability of Θ_T of size 4-exponential in n^2. Then, finding the overall bound K_0 amounts to $2^{2^{n^2}}$ calls to find such witnesses. Finally, we need K_0 oracle calls to Lemma 9 in order to decide existential simulation, and since K_0 may have a 4-exponential size description, this approach yields a whopping 5-EXP algorithm. This approach, however, does not exploit any of the structure of Θ_T. See Section 7 for additional comments.

5.3 Lower Bounds for Existential Round Simulation

The complexity bounds in Remark 22 are naively analyzed, and we leave it for future work to conduct a more in-depth analysis. In this section, we present lower bounds to delimit the complexity gap. Note that there are two relevant lower bounds: one on the complexity of deciding round simulation, and the other on the minimal value of K_0 in Theorem 14.

We start with the complexity lower bound, which applies already for round equivalence.

▶ **Theorem 23.** *The problem of deciding, given transducers $\mathcal{T}_1, \mathcal{T}_2$, whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$ for* any *k, is* PSPACE*-hard, even for a fixed Λ (given as a 5-state* DFA*).*

Proof sketch. We present a similar reduction to that of Theorem 12, from universality of NFAs (see the full version). In order to account for the unknown value of k, we allow padding words with a fresh symbol #, which is essentially ignored by the transducers. ◀

Next, we show that the minimal value for K_0 can be exponential in the size of the given transducers (in particular, of $\mathcal{T}_2$). See the full version for the complete details.

▶ **Example 24** (Exponential round length)**.** Let $p_1, p_2, \ldots, p_m$ be the first m prime numbers. We define two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over input and output alphabet $\mathcal{P} = \{1, \ldots, m\}$, as depicted in Figure 6 for $m = 3$. Intuitively, $\mathcal{T}_1$ reads input $w \in \Lambda = (1 \cdot 2 \cdots m)^*$ and simply outputs w, whereas $\mathcal{T}_2$ works by reading a letter $i \in \mathcal{P}$, and then outputting i for p_i steps (while reading p_i arbitrary letters) before getting ready to read a new letter i.

[5] Alternation depth is usually counted with the outermost quantifier being existential, which is not the case here, hence 3 instead of 2.

In order for $\mathcal{T}_2$ to k-round simulate $\mathcal{T}_1$, it must be able to output a permutation of $(1 \cdot 2 \cdots m)^*$. In particular, the number of 1's, 2's, etc. must be equal, so k must divide every prime up to p_m, hence it must be exponential in the size of $\mathcal{T}_2$.

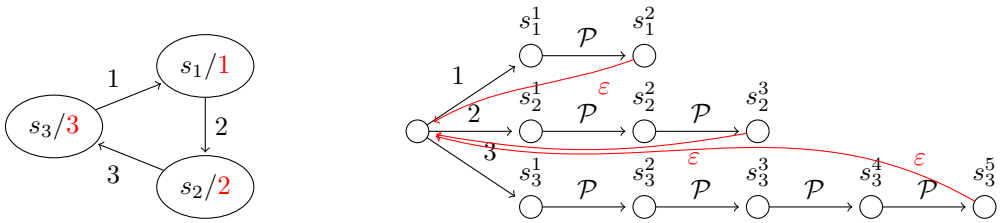


Figure 6 The transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) for $m = 3$ in Example 24. The ε-edge from a state $s_i^{p_i}$ to s_0 in $\mathcal{T}_2$ mean that the transition function from state $s_i^{p_i}$ behaves identically as from s_0.

6 From Process Symmetry to Round Equivalence

As mentioned in Section 1, our original motivation for studying round simulation comes from process symmetry. We present process symmetry with an example before introducing the formal model. Recall the Round Robin (RR) scheduler from Example 3. There, at each time step, the scheduler receives as input the IDs of processes in $\mathcal{P} = \{0, 1, 2\}$ that are making a request, and it responds with the IDs of those that are granted (either a singleton $\{i\}$ or $\emptyset$).

In *process symmetry*, we consider a setting where the identities of the processes may be permuted. This corresponds to the IDs representing e.g., ports, and the processes not knowing which port they are plugged into. Thus, the input received may be any permutation of the actual identities of the processes. Then, a transducer is *process symmetric*, if the outputs are permuted similarly to the inputs. For example, in the RR scheduler, the output corresponding to input $\{1, 2\}\{3\}\{3\}$ is $\{1\}\emptyset\{3\}$. However, if we permute the inputs by swapping 1 and 3, the output for $\{3, 2\}\{1\}\{1\}$ is $\emptyset\emptyset\emptyset$, so RR is not process symmetric.

In [1], several definitions of process symmetry are studied for probabilistic transducers. In the deterministic case, however, process symmetry is a very strict requirement. In order to overcome this, we allow some wriggle room, by letting the transducer do some local order changes in the word that correspond to the permutation. Thus, e.g., if we were allowed to rearrange the input $\{3, 2\}\{1\}\{1\}$ to $\{1\}\{1\}\{3, 2\}$, then the output becomes $\{1\}\emptyset\{3\}$, and once we apply the inverse permutation, this becomes $\{3\}\emptyset\{1\}$. This, in turn, can be again rearranged to obtain the original output (i.e., without any permutation) $\{1\}\emptyset\{3\}$. In this sense, the scheduler is "locally stable" against permutations of the identities of processes.

We now turn to give the formal model. Consider a set of processes $\mathcal{P} = \{1, \ldots, m\}$ and $k > 0$. For a permutation π of $\mathcal{P}$ (i.e. a bijection $\pi : \mathcal{P} \to \mathcal{P}$) and a letter $\sigma \in 2^{\mathcal{P}}$, we obtain $\pi(\sigma) \in 2^{\mathcal{P}}$ by applying π to each process in σ. We lift this to words $x \in (2^{\mathcal{P}})^*$ by applying the permutation letter-wise to obtain $\pi(x)$. A $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer $\mathcal{T} = \langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, Q, q_0, \delta, \boldsymbol{\ell} \rangle$ is *k-round symmetric* if for every permutation π of $\mathcal{P}$ for and every k-round word $x \in (2^{\mathcal{P}})^*$ there exists a k-round word $x' \in (2^{\mathcal{P}})^*$ such that $\pi(x) \asymp_k x'$ and $\pi(\mathcal{T}(x)) \asymp_k \mathcal{T}(x')$. We say that $\mathcal{T}$ is k-round symmetric w.r.t. π if the above holds for a certain permutation π.

Here, too, we consider two main decision problems: *fixed round symmetry* (where k is fixed) and *existential round symmetry* (where we decide whether there exists $k > 0$ for which this holds). Observe that $\Lambda = (2^{\mathcal{P}})^*$, and is hence ignored.

From Round Symmetry to Round Simulation. In order to solve the decision problems above, we reduce them to the respective problems about round symmetry. We start with the case where the permutation π is given.

Given the transducer $\mathcal{T}$ as above, we obtain from $\mathcal{T}$ a new transducer $\mathcal{T}^\pi$ which is identical to $\mathcal{T}$ except that it acts on a letter $\sigma \in 2^\mathcal{P}$ as $\mathcal{T}$ would act on $\pi^{-1}(\sigma)$, and it outputs σ where $\mathcal{T}$ would output $\pi^{-1}(\sigma)$. Formally, $\mathcal{T}^\pi = \langle 2^\mathcal{P}, 2^\mathcal{P}, Q, q_0, \delta^\pi, \boldsymbol{\ell}^\pi \rangle$ where $\delta^\pi(q,\sigma) = \delta(q, \pi^{-1}(\sigma))$ and $\boldsymbol{\ell}^\pi(q) = \pi(\boldsymbol{\ell}(q))$. It is easy to verify that for every $x \in (2^\mathcal{P})^*$ we have $\mathcal{T}^\pi(x) = \pi(\mathcal{T}(\pi^{-1}(x)))$. As we now show, once we have $\mathcal{T}^\pi$, round symmetry is equivalent to round simulation, so we can use the tools developed in Sections 4 and 5 to solve the problems at hand (see the full version for the proof).

▶ **Lemma 25.** *For a permutation π and $k > 0$, $\mathcal{T}$ is k-round symmetric w.r.t. π iff $\mathcal{T}^\pi \prec_k \mathcal{T}$.*

Closure Under Composition. In order to deal with the general problem of symmetry under all permutations, one could naively check for symmetry against each of the $m!$ permutations. We show, however, that the definition above is closed under composition of permutations (see the full version for the proof).

▶ **Lemma 26.** *Consider two permutations π, χ. If $\mathcal{T}^\pi \prec_k \mathcal{T}$ and $\mathcal{T}^\chi \prec_k \mathcal{T}$ then $\mathcal{T}^{\pi\circ\chi} \prec_k \mathcal{T}$.*

Recall that the group of all permutations of $\mathcal{P}$ is generated by two permutations: the transposition $(1\ 2)$ and the cycle $(1\ 2\ \ldots\ m)$ [5]. By Lemma 26 it is sufficient to check symmetry for these two generators in order to obtain symmetry for every permutation. Note that for the existential variant of the problem, even if every permutation requires a different k, by taking the product of the different values, we conclude that there is a uniform k for all permutations. We thus have the following.

▶ **Theorem 27.** *Both fixed and existential round symmetry are decidable. Moreover, fixed round symmetry is in* PSPACE.

Finally, the reader may notice that our definition of round symmetry w.r.t. π is not commutative, as was the case with round symmetry v.s. round equivalence. However, when we consider round symmetry w.r.t. to all permutations, the definition becomes inherently symmetric, as a consequence of Lemma 26 (see the full version for the proof).

▶ **Lemma 28.** *In the notations above, if $\mathcal{T}^\pi \prec_k \mathcal{T}$ then $\mathcal{T} \prec_k \mathcal{T}^\pi$.*

Thus, for symmetry, the notions of round simulation and round equivalence coincide.

7 Future Work

In this work, we introduced round simulation and provided decision procedures and lower bounds (some with remaining gaps) for the related algorithmic problems.

Round simulation, and in particular its application to Round Symmetry, is only an instantiation of a more general framework of symmetry, by which we measure the stability of transducers under local changes to the input. In particular, we plan to extend this study to other definitions, such as *window simulation*, where we use a sliding window of size k instead of disjoint k-rounds, and *Parikh round symmetry*, where the alphabet is of the form $2^\mathcal{P}$, and we are allowed not only to permute the letters in each round, but also to shuffle the individual signals between letters in the round. In addition, the setting of infinite words is of interest, where one can define *ultimate simulation*, requiring the simulation to only hold after a finite prefix. Finally, other types of transducers may require variants of simulation, such as probabilistic transducers, or streaming-string transducers [2].

References

1. S. Almagor. Process symmetry in probabilistic transducers. In *40th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2020*, 2020.
2. R. Alur. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, 2010.
3. I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976.
4. J. A. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973.
5. P. J. Cameron et al. *Permutation groups*, volume 45. Cambridge University Press, 1999.
6. E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal methods in system design*, 9(1-2):77–104, 1996.
7. E.M. Clarke, T.A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.
8. D. C Cooper. Theorem proving in arithmetic without multiplication. *Machine intelligence*, 7(91-99):300, 1972.
9. E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1-2):105–131, 1996.
10. H. Fernau, M. Paramasivan, and M. L. Schmid. Jumping finite automata: characterizations and complexity. In *International Conference on Implementation and Application of Automata*, pages 89–101. Springer, 2015.
11. M.J. Fischer and M.O. Rabin. *Super-exponential Complexity of Presburger Arithmetic*. Project MAC: MAC technical memorandum. Massachusetts Institute of Technology Project MAC, 1974. URL: `https://books.google.co.il/books?id=ijoNHAAACAAJ`.
12. C. Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. URL: `https://dl.acm.org/citation.cfm?id=3242964`.
13. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Proc. 8th Conferance on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, Warsaw, July 1997. Springer-Verlag.
14. M. P. Herlihy and J. M. Wing. Axioms for concurrent objects. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 13–26, 1987.
15. S. Hoffmann. State complexity bounds for the commutative closure of group languages. In *International Conference on Descriptional Complexity of Formal Systems*, pages 64–77. Springer, 2020.
16. C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal methods in system design*, 9(1-2):41–75, 1996.
17. A. W. Lin, T. K. Nguyen, P. Rümmer, and J. Sun. Regular symmetry patterns. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 455–475. Springer, 2016.
18. A. Meduna and P. Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(07):1555–1578, 2012.
19. R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 481–489. British Computer Society, 1971.
20. D. C. Oppen. A 222pn upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.
21. R. J. Parikh. On context-free languages. *J. of the ACM*, 13(4):570–581, 1966.
22. K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In *International Conference on Automated Deduction*, pages 337–352. Springer, 2005.

Useful Open Call-By-Need

Beniamino Accattoli
Inria & École Polytechnique, Palaiseau, France

Maico Leberle
Inria & École Polytechnique, Palaiseau, France

Abstract

This paper studies useful sharing, which is a sophisticated optimization for λ-calculi, in the context of call-by-need evaluation in presence of open terms. Useful sharing turns out to be harder in call-by-need than in call-by-name or call-by-value, because call-by-need evaluates inside environments, making it harder to specify when a substitution step is useful. We isolate the key involved concepts and prove the correctness and the completeness of useful sharing in this setting.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Theory of computation → Operational semantics

Keywords and phrases lambda calculus, call-by-need, operational semantics, sharing, cost models

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.4

Related Version There is an extended version with proofs.
Full Version: `https://arxiv.org/abs/2107.06591`

1 Introduction

Despite decades of research on how to best evaluate λ-terms, the topic is still actively studied and recent years have actually seen a surge in new results and sophisticated techniques. This paper is an attempt at harmonizing two of them, namely, *strong call-by-need* and *useful sharing*, under the influence of a third recently identified setting, *open call-by-value*. To describe our results, we have to first outline each of these approaches.

Call-by-Need. Call-by-need (shortened to CbNeed) is an evaluation scheme for the λ-calculus introduced in 1971 by Wadsworth [53] as an optimization of call-by-name (CbN), and nowadays lying at the core of the Haskell programming language. In the '90s, it was reformulated as operational semantics by Launchbury [45], Ariola and Felleisen [19], and Maraist et al. [48], and implemented by Sestoft [52] and further studied by Kutzner and Schmidt-Schauß [44]. Despite being decades old, CbNeed is still actively studied, perhaps more than ever before. The last decade indeed saw a number of studies by e.g. Ariola et al. [20], Chang and Felleisen [31], Danvy and Zerny [35], Downen et al. [36], Garcia et al. [37], Hackett and Hutton [39], Pédrot and Saurin [50], Mizuno and Sumii [49], Herbelin and Miquey [40], and Kesner et al. [42], plus those mentioned in the following paragraphs.

In the untyped, effect-free setting of the λ-calculus, CbNeed can be seen as borrowing the best aspects of call-by-value (CbV), of which it takes efficiency, and of CbN, of which it retains the better terminating behavior, as stressed in particular by Accattoli et al. [17]. In contrast to CbN and CbV, however, CbNeed cannot easily be managed at the small-step level of the usual operational semantics of the λ-calculus, based on β-reduction and meta-level substitution. Its fine dynamics, indeed, requires a decomposition of the substitution process acting on single variable occurrences at a time – what we refer to as *micro-step (operational) semantics* – and enriching λ-terms with some form of first-class *sharing*. While Wadsworth's original presentation is quite difficult to manage, along the years presentations of CbNeed have improved considerably ([45, 48, 19, 31]), up to obtaining neat definitions, as the one by

Accattoli et al. [6] (2014) in the *linear substitution calculus* (shortened to LSC), which led to elegant proofs of its correctness with respect to CbN, as done by Kesner [41] (2016), and of its relationship with neededness from a rewriting point of view, by Kesner et al. [43] (2018).

Strong Call-by-Need. Being motivated by functional languages, CbNeed is usually studied considering two restrictions with respect to the ordinary λ-calculus: 1) terms are closed, and 2) abstraction bodies are not evaluated. Let us call this setting *Closed CbNeed*. Extensions of CbNeed removing both these restrictions have been considered, obtaining what we shall refer to as *Strong CbNeed*. In his PhD thesis [27] (1999), Barras designs and implements an abstract machine for Strong CbNeed, which has then been used in the kernel of the Coq proof assistant to decide the convertibility of terms. Balabonski et al. [23] (2017) give instead the first formal operational semantics of Strong CbNeed, proving it correct with respect to Strong CbN– see also Barenbaum et al. [25], where the semantics of [23] is extended towards Barras's work; Biernacka and Charatonik [28], where it is studied via an abstract machine; Balabonski et al. [24] where it has recently been revisited and partially formalized.

CbNeed and the Strong Barrier. The definition of Strong CbNeed in [23] builds over the simple one in the LSC, and yet is very sophisticated and far from obvious. This is an instance of a more general fact concerning implementation techniques: dealing with the strong setting is orders of magnitude more difficult than with the closed setting, it is not just a matter of adapting a few definitions. New complex issues show up, requiring new techniques and concepts – let us refer to this fact as to *the strong barrier*. Another instance is the fact that Lévy's optimality [47] is far more complex in the strong case than in the weak one [29, 22].

For neededness, the tool to break the strong barrier is a complex notion of *needed evaluation context*, parametrized and defined by mutual induction with their sets of *needed variables*. Specifying the positions in a term where needed redexes take place is very subtle.

Reasonable Cost Models and the Strong Barrier. Another sophisticated form of sharing for λ-calculi arose recently in the study of whether the λ-calculus admits reasonable evaluation strategies, that is, strategies whose number of β steps is a reasonable time cost model (i.e. measure of time complexity) for λ-terms. The number of function calls (that is, β-steps) is the cost model often used in practice for functional programs – this is done for instance by Charguéraud and Pottier in [32]. A time cost model is *reasonable* when it is polynomially equivalent to the one of Turing machines, which is the requirement for good time cost models. For the λ-calculus, the theory justifying the practice of taking the number of function calls as a time cost model is far from trivial. It is an active research topic, see Accattoli [5].

The first result about λ-calculus reasonable strategies is due to Blelloch and Greiner [30] (1995), and concerns Closed CbV. The 2000s have seen similar results for Closed CbN and Closed CbNeed by Sands, Gustavson, and Moran [51] and Dal Lago and Martini [34, 33]. These cases are based on simulating the λ-calculus via simple forms of sharing such as those at work in abstract machines. The same kind of sharing can also be represented in the LSC, as shown by Accattoli et al. [6]. The strong case seemed elusive and was suspected not to be reasonable, because of Asperti and Mairson's result that Lévy's optimal (strong) strategy is not reasonable [21] – the elusiveness was just another instance of the strong barrier.

Useful Sharing. In 2014, Accattoli and Dal Lago managed to break the barrier, proving that Strong CbN (also known as leftmost-outermost evaluation, or *normal order*) is a reasonable strategy [14]. The proof rests on a simulation of Strong CbN in a refinement of the LSC with a new further level of sharing, deemed *useful sharing*. They also show useful sharing to be *mandatory* for breaking the strong barrier for reasonability.

Useful sharing amounts to doing minimal *unsharing* work, namely only when it contributes to creating β-steps, while avoiding to unfold the sharing (i.e. to substitute) when it only makes the term grow in size. Similarly to CbNeed, the specification of useful sharing can take place only at the micro-step level. Note that the replacement of a variable x in t with u can create a β redex only if u is (or shall reduce to) an abstraction *and* there is an applied occurrence of x in t (that is, $t = T\langle xs\rangle$ for some context T). Therefore, restricting to useful substitutions – that is, adopting useful sharing – amounts to two optimizations of the substitution/unfolding process:

1. *Never substitute normal applications*: one must avoid substitutions of terms which are not – and shall not reduce to – abstractions, such as, say, yz, because their substitution cannot create β-redexes. Indeed, $T\langle(yz)s\rangle$ has a β redex if and only if $T\langle xs\rangle$ does.
2. *Substituting abstractions on-demand*: when the term to substitute is an abstraction, one needs to be sure that the variable occurrence to replace is applied, because, for instance, replacing x with I in yx (obtaining $y\mathsf{I}$) is useless, as no β-redexes are created.

The first optimization is easy to specify, because it concerns the shape of the terms to substitute, that is, *what* to substitute – it has a small-step nature. The second one instead is very delicate, as it also concerns *where* to substitute. It depends on single variable occurrences and thus it is inherently *micro-step* – note that x has both a useful and a useless occurrence in xx. Similarly to Strong CbNeed, the difficulty is specifying *useful evaluation contexts*.

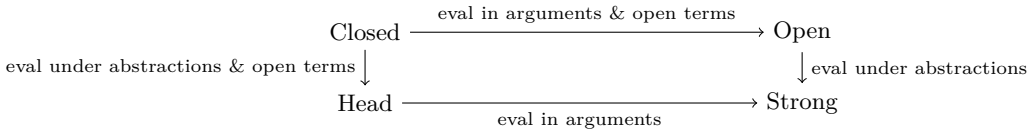
Strong CbNeed and Useful Sharing. Given the similar micro-step traits of CbNeed and useful sharing, and their similar difficulties, it is natural to wonder whether they can be combined. The operational semantics of Strong CbNeed in [23] has the easy useful optimization hardcoded, as it substitutes only abstractions. However, it ignores the delicate second optimization, and its number of β steps is therefore not a reasonable cost model. Concretely, this means that the practice of counting function calls does not reflect the cost of Balabonski et al.'s operational semantics for Strong CbNeed. Since Strong CbNeed is used in the implementation of Coq, this issue has both theoretical and practical relevance.

The aim of this paper is to start adapting useful sharing to call-by-need, developing reasonable operational semantics for CbNeed beyond the closed setting, and continuing a research line about CbNeed started by Accattoli and Barras [7, 8]. To explain our approach, we first need to overview a recent new perspective on the strong barrier.

Opening the Strong Barrier. The theory of the λ-calculus has mainly been developed in CbN. Historically, Barendregt stressed the importance of *head* evaluation (which does not evaluate arguments) for a meaningful representation of partial recursive functions – this is the leading theme of his famous book [26]. A decade later, Abramsky and Ong stressed the relevance of *weak* head evaluation (which does not evaluate abstraction bodies either) to model functional programming languages [1]. Therefore, the usual incremental way to understand strong evaluation is to start with the *closed* CbN case (i.e., weak head evaluation and closed terms), then turn to the *head* case (head evaluation and open terms), and finally add evaluation into arguments obtaining the *strong* case (and leftmost-outermost evaluation). This is for instance the progression that has been followed by Accattoli and Dal Lago to obtain a reasonable time cost model for Strong CbN [13, 14].

In a line of work by Accattoli and co-authors [9, 15, 16, 12] aimed at developing a theory of CbV beyond the usual closed case, it became clear that there is an alternative and better route to the strong setting. The idea is to consider the intermediate *open* setting (rather

than the head one) obtained by enabling evaluation in arguments and open terms (as in the strong case), while still forbidding evaluation in abstraction bodies (as in the closed case). One can summarize the situation with the following diagram:



They also show that useful sharing *factors* through the open setting, rather than through the head one: the two useful optimizations are irrelevant in the head case, while they make sense in the open one, where they can be studied without facing the whole of the strong barrier.

The strong setting can be seen as the iteration of the open one under abstraction, (but not as the iteration of the closed one, because diving into abstractions forces to deal with open terms). This view is adopted by Grégoire and Leroy in the design of the second strong abstract machine at work in Coq [38]. Useful sharing for the strong case then amounts to understanding how open useful sharing and the iteration interact, which is subtle and yet is an orthogonal problem. Studying the open case first is the progression followed recently by Accattoli and co-authors to prove that Strong CbV is reasonable for time [9, 16, 11].

This Paper. According to the decomposition of the strong barrier, here we study, as a first step, useful sharing for CbNeed in the open setting. Let us stress that, because of the barrier, it is not practicable to directly study the strong setting – this is also how the study for CbN and CbV, which are simpler than CbNeed, have been carried out in the literature.

An interesting aspect of useful sharing is that, while the underlying principle is the same, its CbN and CbV incarnations look very different, as the two strategies provide different invariants, leading to different realizations of the required optimizations. It is then interesting to explore useful sharing in CbNeed, which can be seen as a merge of CbN and CbV.

Difficulties. It turns out that useful sharing is quite more difficult to specify in CbNeed than in CbN or CbV. Useful sharing requires to know, for every variable replacement, both *what* is being substituted (is it an abstraction?) and *where* (is the variable to replace applied?). Evaluating only needed arguments, and only once, means that CbNeed evaluation moves deeply into a partially evaluated environment, making hard to keep track of both the *what* and the *where* of variable replacements. In particular, a variable might not be applied in the environment but at the same time be meant to replace an applied variable – thus being applied *up to sharing* – making the identification of applied variables a major difficulty.

The definition of useful rewriting steps is always involved. In CbN and CbV, they can nonetheless be specified compactly via the concept of *unfolding*, that is, iterated meta-level substitutions [14, 9]. These definitions can be called *semantical*, as they define useful *micro* steps via side conditions of a *small*-step nature. They are also somewhat *ineffective*, because they require further work to be made operational. Unfortunately, it is unclear how to give a semantic definition of usefulness in CbNeed. In particular, defining useful CbNeed evaluation contexts seems to require the unfolding of contexts, which is tricky, given that in CbNeed the context hole might be shared, thus risking being duplicated by the unfolding.

Outcome. Despite these difficulties, we succeed in designing an operational semantics for Open CbNeed with useful sharing, and proving that it validates the expected properties.

We proceed in three incremental steps. First, we provide a new *split* presentation of Closed CbNeed tuned for the study of useful sharing developed later on. Second, we extend it to the open setting, essentially mimicking Balabonski et al.'s approach [23], but limiting

it to the open fragment. The real novelty is the third step, providing the refinement into a *useful* open CbNeed calculus, of which we prove the good properties. The crucial and sophisticated concept is the one of *useful (CbNeed) evaluation contexts*, which isolate where useful needed substitutions can be triggered. They are parametrized and defined by mutual induction with the notions of both *applied* and *unapplied variables*, similarly to how needed evaluation contexts are parametrized and mutually dependent with needed variables. The isolation of these concepts and the proof of their properties are our main contribution.

Our definition of useful step is *operational* rather than semantical, as we give a direct – and unfortunately involved – definition of useful evaluation contexts, being unclear how to give a semantic definition based on unfoldings in CbNeed. On the positive side, ours is the first fully operational definition of usefulness in the literature. Previous work (in CbN and CbV) has either adopted semantical ones [14, 9], or has given abstract machines realizing the useful optimizations, but avoiding defining a useful calculus on purpose [3, 16, 11].

Among the properties that we prove, two can be seen as capturing the correctness and the completeness of useful sharing with respect to Open CbNeed:

- *Correctness*: useful substitution steps are eventually followed by a β step, the one that they contribute to create. That is, our useful steps correctly captures the intended semantics, as no steps irrelevant for β redexes are mistakenly considered as useful.
- *Completeness*: normal forms in Useful Open CbNeed unfold to normal forms in Open CbNeed (the unfolding of normal forms is easy to deal with). That is, useful steps do not stop *too soon*: no steps contributing to β redexes are mistakenly considered as useless.

Sketched Complexity Analysis. The third essential property for useful sharing, and its reason to be, is *reasonability*: the useful calculus can be implemented within a polynomial (or even linear) overhead in the number of β-steps. We sketch the complexity analysis at the end of the paper. A formal proof requires introducing an abstract machine implementing the calculus. We have developed the machine, but left it to a forthcoming paper for lack of space.

Intersection Types in the Background. Because of the inherent difficulties mentioned above, our calculus is involved, even very involved. To remove the suspicion that it is an ad-hoc calculus, we paired it with a characterization of its key properties via intersection types, used as a validation tool with a denotational flavor, refining the type-based studies in [41, 23, 17]. In such typing system, the delicate notions of useful evaluation contexts, and applied and unapplied variables have natural counterparts, and type derivations can be used to measure both evaluation lengths and the size of normal forms *exactly*. Such a companion study – omitted for lack of space – is in Leberle's PhD thesis [46].

Proofs. We adopt a meticulous approach, developing proofs in full details, almost at the level of a formalization in a proof assistant. The many technical details, mostly of a tedious nature, are in the technical report [18]. This paper explains the relevant concepts.

2 The Need For Useful Sharing

Here we show a paradigmatic case of size exploding family – which is a family of terms whose size grows exponentially with the number of β-steps – motivating the key optimization of useful sharing for open and strong evaluation. Actually, there are *two* paradigmatic cases of size explosion and, accordingly, *two* optimizations characterizing useful sharing. The first

optimization amounts to forbid the substitution of normal applications, and it is hardcoded into CbNeed evaluation, which by definition substitutes only values. Therefore, we omit discussing the first case of size explosion – more details can be found in [14, 11].

Size Explosion. The example of size-explosion we are concerned with is due to Accattoli [2] and based on the following families of terms, the t_i, and results, the u_i (where $\mathsf{I} := \lambda z.z$):

$$t_1 := \lambda x.\lambda y.yxx \qquad t_{n+1} := \lambda x.t_n(\lambda y.yxx) \qquad \Big| \qquad u_0 := \mathsf{I} \qquad u_{n+1} := \lambda y.yu_nu_n$$

▶ **Proposition 1** (Closed and strategy-independent size explosion, [2]). *Let $n > 0$. Then $t_n\mathsf{I} \to_\beta^n u_n$. Moreover, $|t_n\mathsf{I}| = \mathcal{O}(n)$, $|u_n| = \Omega(2^n)$, $t_n\mathsf{I}$ is closed, and u_n is normal.*

The Useful Optimization. It is easily seen that all the terms substituted along the evaluation of the family are abstractions, namely the identity I and instances of u_i, and that none of these abstractions ever becomes the abstraction (on the left) of a β-redex – that is, their substitution does not create, or it is not *useful* for, β-redexes. These abstractions are however duplicated and nested inside each other, being responsible for the exponential growth of the term size. Useful sharing is about avoiding such useless duplications.

If evaluation is weak, and substitution is *micro-step* (i.e. one variable occurrence at a time, when in evaluation position, in a formalism with sharing), then the family does not cause an explosion. The replaced variables indeed are all instances of x in some t_i which are under abstraction, and which are then never replaced in micro-step *weak* evaluation. With micro-step *strong* evaluation, however, these replacement do happen, and the size explodes. When evaluated with Balabonski et al. Strong CbNeed [23], this family takes a number of micro-steps exponential in the number of β steps, showing that – for as efficient as Strong CbNeed may be – the number of β steps does not reasonably measure its evaluation time.

To tame this problem, one needs to avoid useless substitutions, resting on an optimization sometimes called *substituting abstractions on-demand*, which is tricky. It requires abstractions to be substituted *only* on applied variable occurrences: note that the explosion is caused by replacements of variables (namely the instances of x) which are *not* applied, and that thus do not create β-redexes. For instance, the optimization should allow us substituting I on y in yx, because *it is useful*, that is, it creates a β redex, while it should forbid substituting it on x because it is *useless* for β-redexes. Note that this optimization makes sense only when one switches to micro-step evaluation, that is, at the level of machines, because in xx there are both a useful and a useless occurrence of x. The implementation of *substituting abstractions on-demand* is very subtle, also because by not performing useless substitutions, it breaks invariants of the usual open/strong evaluation process.

As shown by Accattoli and Guerrieri [16], in an open (but not strong) setting, substituting abstractions on-demand is not mandatory for reasonability. They also show, however, that it makes nonetheless sense to study it because it is mandatory for obtaining efficient implementations, as it reduces the complexity of the overhead from *quadratic* to *linear* with respect to the size of the initial term. On the other hand, the optimization is mandatory in strong settings, and it is easier to first study it in the open setting, because the iteration under abstraction (required to handle the strong case) introduces new complex subtleties.

3 The Split Presentation of Closed Call-by-Need

In this section we give an unusual *split* presentation of Closed CbNeed that shall be the starting point for our study of the open and the useful open cases of the next sections.

The Need to Split. The linear substitution calculus (LSC) provides a simple and elegant setting for studying CbNeed, as shown repeatedly by Accattoli, Kesner and co-authors [6, 41, 10, 23, 43, 17, 42]. The LSC extends the λ-calculus with *explicit substitutions* (shortened to ES), noted $t[x\leftarrow u]$, which are a compact notation for $\mathsf{let}\ x = u\ \mathsf{in}\ t$. Capture-avoiding meta-level substitution is noted $t\{x\leftarrow u\}$. To model the useful optimization explained above, we shall need to substitute abstractions only on applied variables. Now, in the LSC, ES can appear everywhere in the term, for instance there are terms such as $t := x[x\leftarrow \mathsf{I}]u$. Note that in t it is hard to say whether the replacement of x with I is useful by looking only at the scope of the ES (which is the left of the $[\cdot\leftarrow\cdot]$ construct): the subtlety being that the replacement is indeed useful, because the variable is applied and I is an abstraction, but the application it is involved in is outside the scope of the ES. To avoid this complication, we give a presentation of CbNeed where ES are separated from the term they act upon, and cannot be nested into each other, similarly to what happens in abstract machines. The split presentation is not mandatory to study useful sharing, but it is quite convenient.

Split Grammars. In the split syntax, a *term* is an ordinary λ-term (without ESs), and a *program* is a term together with – in a separate place – a list of ESs, called *environment*.

Given a countable set of variables Var, the syntax of Closed CbNeed is given by:

$$\begin{array}{llcl} \textsc{Values} & v, w & ::= & \lambda x.t \\ \textsc{Terms} & t, u, s & ::= & x \in \mathsf{Var} \mid v \mid tu \end{array} \qquad \begin{array}{llcl} \textsc{Environments} & e, e' & ::= & \epsilon \mid e[x\leftarrow t] \\ \textsc{Programs} & p, q & ::= & (t, e) \end{array}$$

Note that the body of a λ-abstraction is a term and not a program. Of course, extending the framework to strong evaluation – which is left for future work – requires to allow programs under λ-abstractions. Note also that variables are not values. This is standard in works dealing with implementations or efficiency, as excluding them brings a speed-up, as shown by Accattoli and Sacerdoti Coen [10]. In $e[x\leftarrow t]$ and $(u, e[x\leftarrow t])$ the variable x is bound in e and u. Terms and programs are identified modulo α-renaming. Environments are concatenated by simple juxtaposition. We also define the environment look-up operation as follows: set $e(x) := t$ if $e = e'[x\leftarrow t]e''$ and x is not bound in e', and $e(x) := \bot$ otherwise.

Split Contexts and Plugging. Micro-step CbN and CbV evaluation have easy *split* presentations, because their evaluation contexts may be seen as term contexts, using the environment only for look up, see for instance [8]. CbNeed evaluation contexts, instead, need to enter into the environment. Typically in a program such as $(xy, [x\leftarrow z][z\leftarrow t])$, whose head variable x has been found, CbNeed evaluation has to enter inside $[x\leftarrow\cdot]$, finding another (hereditary) head variable z, and in turn enter inside $[z\leftarrow\cdot]$ and evaluate t. The subtlety is that the evaluation of t can create new ESs, which should be added to the program without breaking its structure, that is, outside the ES which is being evaluated ($[z\leftarrow\cdot]$ in the example). The trick to make it work, is using an unusual notion of context plugging. Before defining evaluation contexts we simply discuss split contexts, which are used throughout the paper.

$$\begin{array}{llcl} \textsc{Term ctxs} & T, T' & ::= & \langle\cdot\rangle \mid Tt \mid tT \\ \textsc{Prog. ctxs} & P, Q & ::= & (T, e) \mid (t, E) \end{array} \qquad \begin{array}{llcl} \textsc{Env. ctxs} & E, E' & ::= & \epsilon \mid e\,[x\leftarrow T] \mid E\,[x\leftarrow u] \end{array}$$

Plugging of a term into a context is defined as expected. Plugging of a program into a context, the tricky bit, requires an auxiliary notation $p@[x\leftarrow t]$ for the appending of an ES $[x\leftarrow t]$ to the end of the environment of a program p:

Appending ES			Plugging of programs		
$(t,e)@[x\leftarrow u]$	$:=$	$(t, e[x\leftarrow u])$	$(T,e)\langle t,e'\rangle$	$:=$	$(T\langle t\rangle, e'e)$
$(T,e)@[x\leftarrow u]$	$:=$	$(T, e[x\leftarrow u])$	$(u,e[x\leftarrow T])\langle t,e'\rangle$	$:=$	$(u, e[x\leftarrow T\langle t\rangle]e')$
$(t,E)@[x\leftarrow u]$	$:=$	$(t, E[x\leftarrow u])$	$(u,E[x\leftarrow s])\langle t,e\rangle$	$:=$	$(u,E)\langle t,e\rangle@[x\leftarrow s]$

For instance, $(xy,[x\leftarrow t][y\leftarrow\langle\cdot\rangle][z\leftarrow u])\langle s,[x'\leftarrow t']\rangle = (xy,[x\leftarrow t][y\leftarrow s][x'\leftarrow t'][z\leftarrow u])$. The look-up operation is extended to environment and program contexts as expected.

Next, we define the CbNeed evaluation contexts in the split approach.

Head contexts	H, J	$::=$	$\langle\cdot\rangle \mid Ht$
Hereditary head contexts	H^*, J^*	$::=$	$(H,e) \mid H^*@[x\leftarrow t] \mid H^*\langle x\rangle@[x\leftarrow H]$

The third production for H^* is what allows evaluation to be iterated inside ES, seeing for instance $(xy,[x\leftarrow z][z\leftarrow\langle\cdot\rangle])$ as a hereditary head context of $(xy,[x\leftarrow z][z\leftarrow t])$ (by applying the production twice, the first time obtaining $(xy,[x\leftarrow\langle\cdot\rangle])$).

Split Evaluation Rules. In contrast to most λ-calculi, we do not define the root cases of the rules and then extend them by a closure by evaluation contexts. We rather define them directly at the global level. Adopting global rules is not mandatory, and yet it shall be convenient for dealing with the useful calculus – we use them here too for uniformity.

Closed CbNeed evaluation rules

Multiplicative	$H^*\langle(\lambda x.t)u\rangle$	$\to_{\mathsf{m}}$	$H^*\langle t,[x\leftarrow u]\rangle$	
Exponential	$H^*\langle x\rangle$	$\to_{\mathsf{e}}$	$H^*\langle v\rangle$	if $H^*(x)=v$

The names of the rules are due to the link between the LSC and linear logic, see Accattoli [4]. Note that we use both plugging of terms and programs, to ease up notations. An example of how rule $\to_{\mathsf{m}}$ exploits the unusual notion of plugging is $(xt,[x\leftarrow(\lambda y.u)st'][z\leftarrow u']) \to_{\mathsf{m}} (xt,[x\leftarrow ut'][y\leftarrow s][z\leftarrow u'])$. As it is standard in the study of CbNeed, garbage collection is simply ignored, because it is postponable at the micro-step level. Normal forms of Closed CbNeed are programs of the form (v,e), which are sometimes called *answers*.

4 Open Call-by-Need

We now shift to Open CbNeed, an evaluation strategy extending Closed CbNeed and allowing reduction to act on possibly *open* programs. Roughly, the strategy iterates CbNeed evaluation on the arguments of the head variable, when the normal form of ordinary CbNeed evaluation is not an abstraction, which can happen when terms are not necessarily closed. Various aspects of Closed CbNeed become subtler in Open CbNeed, namely the definition of evaluation contexts and the structure of normal forms, together with the new notion of needed variables. Essentially, we are giving an alternative presentation of the open fragment of Balabonski et al.'s Strong Call-by-Need, with which we compare at the end of the section.

Some Motivating Examples. We show a few examples of the rewriting relation that we aim at defining, as to guide the reader through the technical aspects. We want reduction to take place in arguments, after a (hereditary) head variable has been found, having e.g.:

$$(x((\lambda z.z)\mathsf{I}),[y\leftarrow t]) \to_{\mathsf{m}} (xz,[z\leftarrow\mathsf{I}][y\leftarrow t]) \quad \text{and} \quad (xz,[z\leftarrow\mathsf{I}][y\leftarrow t]) \to_{\mathsf{e}} (x\mathsf{I},[z\leftarrow\mathsf{I}][y\leftarrow t])$$

For appropriate generalizations of $\to_{\mathsf{m}}$ and $\to_{\mathsf{e}}$. Of course, we retain and extend to arguments the hereditary character of the reduction rules, therefore having also steps such as:

$$(yx, [x{\leftarrow}y((\lambda z.z)\mathsf{I})]) \to_{\mathsf{m}} (yx, [x{\leftarrow}yz][z{\leftarrow}\mathsf{I}]), \text{ and } (yx, [x{\leftarrow}yz][z{\leftarrow}\mathsf{I}]) \to_{\mathsf{e}} (yx, [x{\leftarrow}y\mathsf{I}][z{\leftarrow}\mathsf{I}])$$

While the intended behavior is – we hope – clear, specifying these steps via evaluation contexts requires some care and a few definitions. Essentially, we need to understand when evaluation can pass to the next argument, and thus characterize when terms are normal. This is easy for terms but becomes tricky for programs.

Evaluation Places and Needed Variables. The grammars of the language are the same as for split Closed CbNeed, but defining the open evaluation contexts is quite subtler. In Closed CbNeed there is only one place of the term where evaluation can take place, the hereditary head context H^*. In the open setting the situation is more general: there is one *active* evaluation place *plus* potentially many *passive* ones, which are those places where evaluation already passed and ended. On some of these passive places, evaluation ended on a free variable (occurrence). We refer to these free variables as *needed* (definition below[1]), as they shall end up in the normal form, given that at least one of their occurrences has already been evaluated and cannot be erased. For instance in $p := (x(y\mathsf{I}), [z{\leftarrow}x][y{\leftarrow}\mathsf{I}])$ the active place is y, the first occurrence of x is a needed occurrence, while the second one is not.

Needed vars for terms

$$\begin{array}{rcl} \mathsf{nv}(x) & := & \{x\} \\ \mathsf{nv}(\lambda x.t) & := & \emptyset \\ \mathsf{nv}(tu) & := & \mathsf{nv}(t) \cup \mathsf{nv}(u) \end{array}$$

Needed variables for programs

$$\begin{array}{rcl} \mathsf{nv}(t, \epsilon) & := & \mathsf{nv}(t) \\ \mathsf{nv}(t, e[x{\leftarrow}u]) & := & \begin{cases} \mathsf{nv}(t,e) & x \notin \mathsf{nv}(t,e) \\ (\mathsf{nv}(t,e) \setminus \{x\}) \cup \mathsf{nv}(u) & x \in \mathsf{nv}(t,e) \end{cases} \end{array}$$

The difficulty in defining Open CbNeed is in the inductive definition of both normal forms and evaluation contexts. The problem is that extending a term or a context with a new ES may re-activate a passive evaluation place, if the ES binds a needed variable occurrence. For instance, appending $[x{\leftarrow}\delta]$ to p above would reactivate the needed occurrence of x.

Normal Terms. In Open CbNeed normal forms are not simply answers (i.e. abstractions together with an environment), as free variables induce a richer structure. We shall later characterize the subtle inductive structure of normal programs. For now, we need predicates (that shall be later shown) characterizing normal *terms*, as they are used to define evaluation contexts. The definition and the terminology are borrowed from Open CbV [9, 15], where normal terms are called *fireballs* and are defined by mutual induction with *inert terms*:

$$\begin{array}{rrcl} \text{Values} & v, w & ::= & \lambda x.t \\ \text{Fireballs} & f, g & ::= & v \mid i \end{array} \qquad \begin{array}{rrcl} \text{Inert terms} & i, j & ::= & x \in \mathsf{Var} \mid if \\ \text{Non-var inert terms} & i^+ & ::= & if \end{array}$$

Later on, we shall often need to refer to inert terms that are not variables, which is why we introduce now a dedicated notation. We shall sometimes write $\mathsf{inert}(t)$ (resp., $\mathsf{abs}(t)$) to express that t is an inert term (resp., an abstraction).

[1] Needed variables are intended to be considered only for normal terms (or normal programs, or normal parts of a context), and yet the definition is given here for every term (in particular every applications, instead of only inert applications if). The reason for our lax definition is that the technical development requires at times to consider the needed variables of a term that is not yet known to be normal. The lax definition goes against the *needed* intuition, as one of the reviewers understandably complained about, suggesting to call these variables *frozen*, following Balabonski et al. [23]. We preferred to keep *needed* because they are similar but different from the frozen variables in [23], see the end of this section.

Needed vars for term ctxs

$$\begin{aligned}\mathsf{nv}(\langle\cdot\rangle) &:= \emptyset\\ \mathsf{nv}(Ht) &:= \mathsf{nv}(H)\\ \mathsf{nv}(iH) &:= \mathsf{nv}(i)\cup\mathsf{nv}(H)\end{aligned}$$

Open evaluation contexts and their needed vars

$$\frac{}{(H,\epsilon)\in\mathcal{E}_{\mathsf{nv}(H)}}\ \mathsf{O}_{\mathsf{AX}} \qquad \frac{P\in\mathcal{E}_{\mathcal{V}} \quad x\in\mathcal{V}}{P@[x{\leftarrow}i]\in\mathcal{E}_{(\mathcal{V}\setminus\{x\})\cup\mathsf{nv}(i)}}\ \mathsf{O}_{\mathsf{I}}$$

$$\frac{P\in\mathcal{E}_{\mathcal{V}} \quad x\notin\mathcal{V}}{P@[x{\leftarrow}t]\in\mathcal{E}_{\mathcal{V}}}\ \mathsf{O}_{\mathsf{GC}} \qquad \frac{P\in\mathcal{E}_{\mathcal{V}} \quad x\notin\mathcal{V}}{P\langle x\rangle@[x{\leftarrow}H]\in\mathcal{E}_{\mathcal{V}\cup\mathsf{nv}(H)}}\ \mathsf{O}_{\mathsf{HER}}$$

Figure 1 Needed variables for term contexts and the derivation rules for open evaluation contexts.

$$\frac{}{\mathsf{inert}(i,\epsilon)}\ \mathsf{I}_{\mathsf{AX}} \qquad \frac{\mathsf{inert}(p) \quad x\in\mathsf{nv}(p)}{\mathsf{inert}(p@[x{\leftarrow}i])}\ \mathsf{I}_{\mathsf{I}} \qquad \frac{\mathsf{inert}(p) \quad x\notin\mathsf{nv}(p)}{\mathsf{inert}(p@[x{\leftarrow}t])}\ \mathsf{I}_{\mathsf{GC}}$$

$$\frac{}{\mathsf{abs}(v,\epsilon)}\ \mathsf{A}_{\mathsf{AX}} \qquad \frac{\mathsf{abs}(p)}{\mathsf{abs}(p@[x{\leftarrow}t])}\ \mathsf{A}_{\mathsf{GC}}$$

Figure 2 Predicates for Open CbNeed normal programs.

Evaluation Contexts. Open evaluation contexts cannot be defined with a grammar, as for the closed case, because they are defined by mutual induction with their own set of needed variables, see the right part of Fig. 1. The notation $P\in\mathcal{E}_{\mathcal{V}}$ means that P is an open evaluation context of needed variables $\mathcal{V}$. We assume that $x\notin\mathsf{dom}(P)$ in rules O_{GC}, O_{I} and $\mathsf{O}_{\mathsf{HER}}$, in accordance with *Barendregt's variable convention*. The base case O_{AX} requires the notion of needed variables for term contexts, which is on the left side of Fig. 1.

Rule O_{AX} simply coerces term contexts to program contexts. The production $P@[x{\leftarrow}t]$ for the closed case here splits into the two rules O_{GC} and O_{I}. This is relative to needed variables: one can append the ES $[x{\leftarrow}t]$ only if x is not needed (O_{GC}) or, when x is needed, if the content t of the ES is inert (O_{I}), as to avoid re-activation of a passive evaluation place on x. Rule $\mathsf{O}_{\mathsf{HER}}$ is the open version of the production $P\langle x\rangle@[x{\leftarrow}H]$, with the needed variables constraint to prevent re-activations. Examples: $(xy,[y{\leftarrow}\langle\cdot\rangle])$ and $(xy,[y{\leftarrow}z][z{\leftarrow}\langle\cdot\rangle])$ for $\mathsf{O}_{\mathsf{HER}}$, $(xy,[y{\leftarrow}\langle\cdot\rangle][x{\leftarrow}zz])$ for O_{I}, $(xy,[y{\leftarrow}\langle\cdot\rangle][z{\leftarrow}zz])$ for O_{I}.

▶ **Lemma 2** (Unique parameterization of open evaluation contexts)**.** *Let $P\in\mathcal{E}_{\mathcal{V}}$ and $P\in\mathcal{E}_{\mathcal{W}}$. Then $\mathcal{V}=\mathcal{W}$.*

Open Evaluation Rules. The definition of the evaluation rules mimics exactly the one for the split closed case. Given an Open CbNeed evaluation context $P\in\mathcal{E}_{\mathcal{V}}$, we have:

Open CbNeed evaluation rules

$$\begin{array}{rlll} \text{Open multiplicative} & P\langle(\lambda x.t)u\rangle & \to_{\mathsf{om}} & P\langle t,[x{\leftarrow}u]\rangle \\ \text{Open exponential} & P\langle x\rangle & \to_{\mathsf{oe}} & P\langle v\rangle \qquad \text{if } P(x)=v \end{array}$$

We shall say that p reduces to q in the Open CbNeed evaluation strategy, and write $p\to_{\mathsf{ond}} q$, whenever $p\to_{\mathsf{om}} q$ or $p\to_{\mathsf{oe}} q$.

▶ **Proposition 3** (Determinism of Open CbNeed)**.** *Reduction $\to_{\mathsf{ond}}$ is deterministic.*

Normal Programs. Normal programs mimic normal terms and are of two kinds, inert or abstractions. The definition however now depends on needed variables and cannot be given as a simple grammar. The two predicates inert and abs are defined in Fig. 2. Finally, predicate onorm is defined as the union of inert and abs, that is, $\mathsf{onorm}(p)$ if $\mathsf{inert}(p)$ or $\mathsf{abs}(p)$. The intended meaning is that it characterizes programs in Open CbNeed-normal form.

▶ **Proposition 4** (Syntactic characterization of Open CbNeed-normal forms). *Let p be a program. Then p is in $\to_{\mathsf{ond}}$-normal form if and only if* $\mathsf{onorm}(p)$.

The proofs of Prop. 3 and 4 (in [18]) are subtler and longer than one might expect, because of the fact that evaluation contexts and needed variables are mutually defined.

Relationship with Balabonski et al. With respect to the definition of Strong CbNeed in [23], we follow essentially the same approach up to two differences, not counting the obvious fact that we are open and not strong. First, we use a split calculus, while they do not, because they do not study useful sharing.

Second, they have a similar but different parametrization of evaluation contexts. They are more liberal, as their sets of *frozen variables* used as parameters are supersets of our needed variables, but they also parametrize reduction steps, which we avoid. Our 'tighter' choice is related to the fine study of intersection types for Open CbNeed, which can be found in the Leberle's PhD thesis [46], and it is also essential for the refinement required by the useful extension of Sect. 5. In [24], a reformulation of [23] using a deductive system (parametrized by frozen variables) rather than evaluation contexts is used – it could also be used here.

5 Useful Open Call-by-Need

Roughly, useful sharing is an optimization of micro-step substitutions, that is, of exponential steps. The idea is that there are substitution steps that are useful to create β/multiplicative redexes and steps that are useless. For instance (the underline stresses the created β-redex):

Example of useful step	Example of useless step
$(xy, [x\leftarrow \mathsf{I}]) \to_{\mathsf{oe}} (\mathsf{I}\underline{y}, [x\leftarrow \mathsf{I}])$	$(xy, [y\leftarrow \mathsf{I}]) \to_{\mathsf{oe}} (x\mathsf{I}, [y\leftarrow \mathsf{I}])$

The main idea is that useful steps replace applied variable occurrences, while useless steps replace unapplied occurrences. The definition of the useful calculus then shall refine the open one by replacing the set of needed variables with *two* sets, one for applied and one for unapplied variable occurrences. Note a subtlety: variables can have both applied and unapplied needed occurrences, as x in xx. Therefore, usefulness is a concept that can be properly expressed only when considering replacements of single variable occurrences.

Usefulness unfortunately is not so simple. Consider the following step replacing z with I:

$$(xy, [x\leftarrow z][z\leftarrow \mathsf{I}]) \to_{\mathsf{oe}} (xy, [x\leftarrow \mathsf{I}][z\leftarrow \mathsf{I}]) \tag{1}$$

Is it useful or useless? It does not create a multiplicative redex – therefore it looks useless – but without it we cannot perform the next step $(xy, [x\leftarrow \mathsf{I}][z\leftarrow \mathsf{I}]) \to_{\mathsf{oe}} (\mathsf{I}y, [x\leftarrow \mathsf{I}][z\leftarrow \mathsf{I}])$ replacing x with I which is certainly useful – thus step (1) *has to be useful*.

We then have to refine the defining principle for usefulness: useful steps replace *hereditarily applied* variable occurrences, that is, occurrences that are applied, or that are by themselves (i.e. not in an application) and that are meant to replace a hereditarily applied occurrence.

Handling hereditarily applied variables is specific to CbNeed, and makes defining Useful Open CbNeed quite painful. The key point is the *global* character of the hereditary notion, that requires checking the evaluation context leading to the variable occurrence and it is then not of a local nature. We believe that hereditarily applied variables, nonetheless, are an unavoidable ingredient of usefulness in a CbNeed scenario, and not an ad-hoc point of our study. This opinion is backed by the fact that such a convoluted mechanism is modeled very naturally at the level of intersection types, as it is shown in Leberle's PhD Thesis [46].
Important: from now on, we ease the language saying *applied* to mean *hereditarily applied*.

Applied variables for terms and programs

$$\mathsf{a}(\lambda x.t) \coloneqq \emptyset \qquad \mathsf{a}(x) \coloneqq \emptyset \qquad \mathsf{a}(tu) \coloneqq \begin{cases} \{x\} \cup \mathsf{a}(u) & t = x \in \mathsf{Var} \\ \mathsf{a}(t) \cup \mathsf{a}(u) & t \notin \mathsf{Var} \end{cases} \qquad \mathsf{a}(t,\epsilon) \coloneqq \mathsf{a}(t)$$

$$\mathsf{a}(t, e[x{\leftarrow}u]) \coloneqq \begin{cases} \mathsf{a}(t,e) & x \notin \mathsf{nv}(t,e), \\ (\mathsf{a}(t,e){\setminus}\{x\}) \cup \mathsf{a}(u) & x \in \mathsf{nv}(t,e) \wedge (x \notin \mathsf{a}(t,e) \vee u \notin \mathsf{Var}), \\ (\mathsf{a}(t,e){\setminus}\{x\}) \cup \{y\} & x \in \mathsf{nv}(t,e) \wedge x \in \mathsf{a}(t,e) \wedge u = y \in \mathsf{Var} \end{cases}$$

Unapplied variables for terms and programs

$$\mathsf{u}(\lambda x.t) \coloneqq \emptyset \qquad \mathsf{u}(x) \coloneqq \{x\} \qquad \mathsf{u}(tu) \coloneqq \begin{cases} \mathsf{u}(u) & t \in \mathsf{Var} \\ \mathsf{u}(t) \cup \mathsf{u}(u) & t \notin \mathsf{Var} \end{cases} \qquad \mathsf{u}(t,\epsilon) \coloneqq \mathsf{u}(t)$$

$$\mathsf{u}(t, e[x{\leftarrow}u]) \coloneqq \begin{cases} \mathsf{u}(t,e) & x \notin \mathsf{u}(t,e) \wedge (x \notin \mathsf{nv}(t,e) \vee u = y \in \mathsf{Var}) \\ (\mathsf{u}(t,e){\setminus}\{x\}) \cup \mathsf{u}(u) & x \in \mathsf{u}(t,e) \vee (x \in \mathsf{nv}(t,e) \wedge u \notin \mathsf{Var}) \end{cases}$$

Applied vars of term contexts

$$\begin{array}{rcl} \mathsf{a}(\langle\cdot\rangle) & \coloneqq & \emptyset \\ \mathsf{a}(Ht) & \coloneqq & \mathsf{a}(H) \\ \mathsf{a}(iH) & \coloneqq & \mathsf{a}(i) \cup \mathsf{a}(H) \end{array}$$

Unapplied vars of term contexts

$$\begin{array}{rcl} \mathsf{u}(\langle\cdot\rangle) & \coloneqq & \emptyset \\ \mathsf{u}(Ht) & \coloneqq & \mathsf{u}(H) \\ \mathsf{u}(iH) & \coloneqq & \mathsf{u}(i) \cup \mathsf{u}(H) \end{array}$$

Figure 3 Applied and unapplied variables for terms, programs, and term contexts.

Applied and Unapplied Variables. We now define, for terms, programs, and term contexts, the sets of applied and unapplied variables $\mathsf{a}(\cdot)$ and $\mathsf{u}(\cdot)$, that are subsets of needed variables $\mathsf{nv}(\cdot)$. We shall prove that $\mathsf{nv}(t) = \mathsf{a}(t) \cup \mathsf{u}(t)$ (i.e., the two sets cover $\mathsf{nv}(t)$ exactly). As already pointed out, applied and unapplied variables, however, are *not* a partition of needed variables, that is, in general $\mathsf{a}(t) \cap \mathsf{u}(t) \neq \emptyset$ as a variable can have both applied and unapplied (needed) occurrences, as x in xx. The same holds also for programs and term contexts.

The set of applied variables of terms, programs, and term contexts are defined in Fig. 3 – explanations follow. Having in mind that we want to define $\mathsf{a}(p)$ in such a way that it satisfies $\mathsf{a}(p) \subseteq \mathsf{nv}(p)$, note that condition $x \in \mathsf{nv}(t,e) \wedge x \in \mathsf{a}(t,e) \wedge u = y \in \mathsf{Var}$ in the definition of $\mathsf{a}(t, e[x{\leftarrow}u])$ would more simply be $x \in \mathsf{a}(t,e) \wedge u = y \in \mathsf{Var}$. However, we have not proved yet that $\mathsf{a}(p) \subseteq \mathsf{nv}(p)$, which is why the definition is given in this more general form.

We give some examples. As expected, y is an applied variable of $y\,z$ and $z\,(y\,z)$. It is also applied in $p \coloneqq (z\,x, [x{\leftarrow}y\,z])$, even if x is not applied in $(z\,x, \epsilon)$. Thus, Useful Open CbNeed evaluation shall be defined as to include exponential steps such as $(z\,x, [x{\leftarrow}y\,z][z{\leftarrow}v]) \to_{\mathsf{oe}} (z\,x, [x{\leftarrow}v\,z][y{\leftarrow}v])$, which are useful. Note that y is not applied in $(x, [z{\leftarrow}yx])$, because applied variables have to be needed variables, and y is not needed. Another example: if $p \coloneqq (xt, [x{\leftarrow}y])$, then $y \in \mathsf{a}(p)$ (and also $z \in \mathsf{a}((xt, [x{\leftarrow}y][y{\leftarrow}z]))$). Useful Open CbNeed, then, shall retain the following two exponential steps of the open case, since the sequence is supposed to continue with a $\to_{\mathsf{m}}$ step, contracting the redex given by vt:

$$(xt, [x{\leftarrow}y][y{\leftarrow}v]) \to_{\mathsf{oe}} (xt, [x{\leftarrow}v][y{\leftarrow}v]) \to_{\mathsf{oe}} (vt, [x{\leftarrow}v][y{\leftarrow}v])$$

The set of unapplied variables of terms, programs, and term contexts are defined in Fig. 3. Once again, in the second clause defining $\mathsf{u}(t, e[x{\leftarrow}u])$ the side condition $x \in \mathsf{nv}(t,e)$ can be replaced by $x \in \mathsf{a}(t,e)$, after Lemma 5 below is proved.

$$\frac{}{(H,\epsilon) \in \mathcal{E}_{\mathsf{u}(H),\mathsf{a}(H)}}\ \mathsf{M}_{\mathsf{AX}} \qquad \frac{P \in \mathcal{E}_{\mathcal{U},\mathcal{A}} \quad x \in (\mathcal{U} \cup \mathcal{A})}{P@[x{\leftarrow}y] \in \mathcal{E}_{\mathsf{upd}(\mathcal{U},x,y),\mathsf{upd}(\mathcal{A},x,y)}}\ \mathsf{M}_{\mathsf{VAR}}$$

$$\frac{P \in \mathcal{E}_{\mathcal{U},\mathcal{A}} \quad x \notin (\mathcal{U} \cup \mathcal{A})}{P@[x{\leftarrow}t] \in \mathcal{E}_{\mathcal{U},\mathcal{A}}}\ \mathsf{M}_{\mathsf{GC}} \qquad \frac{P \in \mathcal{E}_{\mathcal{U},\mathcal{A}} \quad x \in (\mathcal{U} \cup \mathcal{A})}{P@[x{\leftarrow}i^+] \in \mathcal{E}_{(\mathcal{U}\setminus\{x\})\cup\mathsf{u}(i^+),(\mathcal{A}\setminus\{x\})\cup\mathsf{a}(i^+)}}\ \mathsf{M}_{\mathsf{I}}$$

$$\frac{P \in \mathcal{E}_{\mathcal{U},\mathcal{A}} \quad x \in (\mathcal{U} \setminus \mathcal{A})}{P@[x{\leftarrow}v] \in \mathcal{E}_{\mathcal{U}\setminus\{x\},\mathcal{A}}}\ \mathsf{M}_{\mathsf{U}} \qquad \frac{P \in \mathcal{E}_{\mathcal{U},\mathcal{A}} \quad x \notin (\mathcal{U} \cup \mathcal{A})}{P\langle x\rangle@[x{\leftarrow}H] \in \mathcal{E}_{\mathcal{U}\cup\mathsf{u}(H),\mathcal{A}\cup\mathsf{a}(H)}}\ \mathsf{M}_{\mathsf{HER}}$$

Figure 4 Derivation rules for multiplicative evaluation contexts.

We give some examples. A consequence of the definition is that, as for applied variables, y is not unapplied in $(xx, [z{\leftarrow}xy])$ because it is not needed. As it is probably expected, y is unapplied in $(zx, [z{\leftarrow}xy])$, even if xy is meant to replace z which is applied in zx. Perhaps counter-intuitively, instead, our definitions imply both $y \in \mathsf{a}(p)$ and $y \in \mathsf{u}(p)$ for $p := (xx, [x{\leftarrow}y])$, that is, the unique occurrence of y is both applied and unapplied in p[2].

▶ **Lemma 5** (Unapplied and applied cover needed variables)**.**

1. Terms*:* $\mathsf{nv}(t) = \mathsf{u}(t) \cup \mathsf{a}(t)$ *for every term* t*.*
2. Programs*:* $\mathsf{nv}(p) = \mathsf{u}(p) \cup \mathsf{a}(p)$ *for every program* p*.*
3. Term contexts*:* $\mathsf{nv}(H) = \mathsf{u}(H) \cup \mathsf{a}(H)$*, for every term context* H*.*

Finally, the derived concept of useless variable shall also be used.

▶ **Definition 6** (Useless variables)**.** *Given a term* t*, we define the set of* useless *variables as* $\mathsf{ul}(t) := \mathsf{u}(t) \setminus \mathsf{a}(t)$*. The set of* useless *variables of a program* p *is defined analogously.*

Useless variables are crucial in differentiating Useful Open CbNeed from Open CbNeed. We shall prove that if p is a useful open normal form and $x \in \mathsf{ul}(p)$, then $p@[x{\leftarrow}v]$ is also a useful open normal form (while it is not a open normal form). The notion of useless variables is intuitively simple but technically complex. Some examples. First, note that $\mathsf{ul}(x\,x, \epsilon) = \emptyset$. The example can be extended to a hereditary setting, noting that $\mathsf{ul}(y, [y{\leftarrow}x\,x]) = \emptyset$. However, the reasoning takes into account only needed occurrences, that is, note that $x \in \mathsf{ul}(z\,x, [y{\leftarrow}x\,x])$, as the occurrence of x that is applied to an argument is not needed.

Evaluation Contexts. The definition of evaluation contexts is particularly subtle in the useful case. First of all, their set $\mathcal{E}_{\mathcal{U},\mathcal{A}}$ is indexed by *two* sets of variables (rather than one as in the open case), the applied $\mathcal{A}$ and the unapplied $\mathcal{U}$ variables of the context, defined by mutual induction with the contexts themselves. The second key point is that there are two different kinds of evaluation contexts, a permissive one for multiplicative redexes, whose set is noted $\mathcal{E}_{\mathcal{U},\mathcal{A}}$, and a restrictive one for exponential redexes, noted $\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}$ and implementing the fact that the variable occurrence to be replaced has to be in an applied position. The asymmetry is unavoidable, because useful sharing concerns only exponential steps.

[2] This fact is in accordance with the companion intersection type study in Leberle's PhD thesis [46] mentioned in the introduction: in spite of y having only one syntactic occurrence in p, it is needed twice, and so intersection types derivations of p do type y twice.

$$\frac{}{(H^{@},\epsilon)\in\mathcal{E}^{@}_{\mathsf{u}(H^{@}),\mathsf{a}(H^{@})}}\ \mathsf{E}_{\mathsf{AX}_1} \qquad \frac{P\in\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}\quad x\notin(\mathcal{U}\cup\mathcal{A})}{P\langle x\rangle@[x{\leftarrow}H^{@}]\in\mathcal{E}^{@}_{(\mathcal{U}\setminus\{x\})\cup\mathsf{u}(H^{@}),\mathcal{A}\cup\mathsf{a}(H^{@})}}\ \mathsf{E}_{\mathsf{AX}_2}$$

$$\frac{P\in\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}\quad x\in(\mathcal{U}\cup\mathcal{A})}{P@[x{\leftarrow}y]\in\mathcal{E}^{@}_{\mathsf{upd}(\mathcal{U},x,y),\mathsf{upd}(\mathcal{A},x,y)}}\ \mathsf{E}_{\mathsf{VAR}} \qquad \frac{P\in\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}\quad x\in(\mathcal{U}\cup\mathcal{A})}{P@[x{\leftarrow}i^{+}]\in\mathcal{E}^{@}_{(\mathcal{U}\setminus\{x\})\cup\mathsf{u}(i^{+}),(\mathcal{A}\setminus\{x\})\cup\mathsf{a}(i^{+})}}\ \mathsf{E}_{\mathsf{I}}$$

$$\frac{P\in\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}\quad x\notin(\mathcal{U}\cup\mathcal{A})}{P@[x{\leftarrow}t]\in\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}}\ \mathsf{E}_{\mathsf{GC}} \qquad \frac{P\in\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}\quad x\in(\mathcal{U}\setminus\mathcal{A})}{P@[x{\leftarrow}v]\in\mathcal{E}^{@}_{\mathcal{U}\setminus\{x\},\mathcal{A}}}\ \mathsf{E}_{\mathsf{U}} \qquad \frac{P\in\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}\quad x\notin\mathcal{A}}{P\langle x\rangle@[x{\leftarrow}\langle\cdot\rangle]\in\mathcal{E}^{@}_{\mathcal{U}\setminus\{x\},\mathcal{A}}}\ \mathsf{E}_{\mathsf{NA}}$$

Figure 5 Derivation rules for exponential evaluation contexts.

Multiplicative Contexts. They are a refinement of the open contexts defined in Fig. 4. Their set is noted $\mathcal{E}_{\mathcal{U},\mathcal{A}}$. The refinement is needed even if useful sharing concerns only exponential steps: a multiplicative context such as $((yx)\langle\cdot\rangle,[x{\leftarrow}v])\in\mathcal{E}_{\emptyset,\{y\}}$ indeed is not an open context, because it contains a useless substitution step that in Open CbNeed would be fired before evaluating the hole. The definition of multiplicative contexts follows the one for Open CbNeed contexts (M_{AX}, M_{GC}, and $\mathsf{M}_{\mathsf{HER}}$ are essentially as before) *except for* rule:

$$\frac{P\in\mathcal{E}_{\mathcal{V}}\quad x\in\mathcal{V}}{P@[x{\leftarrow}i]\in\mathcal{E}_{(\mathcal{V}\setminus\{x\})\cup\mathsf{nv}(i)}}\ \mathsf{O}_{\mathsf{I}}$$

which is now generalized into 3 rules, depending on the kind of term contained in the ES. That is, given $P\in\mathcal{E}_{\mathcal{U},\mathcal{A}}$ and $x\in(\mathcal{U}\cup\mathcal{A})$, the constraints to extend P with an ES $[x{\leftarrow}t]$ are:

- *Rule* M_{I}: there are no constraints if t is a non-variable inert term i^{+}. Note that M_{I} and M_{GC} together imply that we can *always* append ESs containing inert terms to multiplicative contexts, without altering the Useful Open CbNeed order of reduction.
- *Rule* $\mathsf{M}_{\mathsf{VAR}}$: this rule covers the case where t is a variable y. It is used to handle the global applicative constraint, as in such a case, if the evaluation context is $P@[x{\leftarrow}y]$, then y has to be added to the applied and/or unapplied variables of the context, according to the role played by x in P, which is realized via the function upd defined as follows:

$$\mathsf{upd}(S,x,y):=\begin{cases}S & x\notin S\\ (S\setminus\{x\})\cup\{y\} & x\in S\end{cases}$$

- *Rule* M_{U}: it covers the case where t is a value v, requiring that x is not applied, that is, $\notin\mathcal{A}$. Such an extension would have re-activated x in the plain open case, and created a (useless) exponential redex, but here it shall not be the case. Note that it means that $P@[x{\leftarrow}t]$ is a multiplicative context *only* if $x\in(\mathcal{U}\setminus\mathcal{A})$, i.e. if x is a *useless* variable of P.

Exponential Contexts. Exponential contexts are even more involved, because they have to select only *applicative* variable occurrences and the applicative constraint is of a global nature. First, we need a notion of applicative term context, where the hole is applied.

▶ **Definition 7** (Applicative term contexts). *A term context H shall be called an* applicative term context *if it is derived using the grammar $H^{@},J^{@},I^{@}::=\langle\cdot\rangle t\mid H^{@}t\mid iH^{@}$.*

▶ **Definition 8** (Exponential evaluation contexts). *We shall say that an evaluation context P is a* exponential evaluation context *if it is derived with the rules in Fig. 5.*

Applicative term contexts serve as the base case of exponential evaluation contexts, now given by two refinements of the multiplicative case:

1. the base case $\mathsf{E}_{\mathsf{AX}_1}$ is akin to the base case M_{AX} for multiplicative contexts, except that it requires the term context to be applicative.
2. the plugging-based rule $\mathsf{M}_{\mathsf{HER}}$ splits in two. A first rule $\mathsf{E}_{\mathsf{AX}_2}$ which simply plugs an applicative context $H^{@}$ into a *multiplicative* evaluation context – note that this rule gives another base case for exponential evaluation contexts. A second rule E_{NA} that handles the special case of the global applicative constraint.

Let us see the differences between rules E_{NA} and $\mathsf{E}_{\mathsf{AX}_2}$ with two examples. Their side conditions ($x \notin (\mathcal{U} \cup \mathcal{A})$ and $x \notin \mathcal{A}$) are explained at page 17 of the technical report [18].

- E_{NA}: consider the program $p := (x\,t, [x{\leftarrow}z])$, where z is in applied position due to the global applicative constraint, as it substitutes x which is applied to t. We may derive an exponential evaluation context P that isolates z, that is, such that $P\langle z\rangle = p$, as follows:

$$\dfrac{\dfrac{}{(\langle\cdot\rangle\, t, \epsilon) \in \mathcal{E}^{@}_{\emptyset,\emptyset}}\ \mathsf{E}_{\mathsf{AX}_1} \quad x \notin \emptyset}{P := ((\langle\cdot\rangle\, t, \epsilon)\langle x\rangle)\,@[x{\leftarrow}\langle\cdot\rangle] \in \mathcal{E}^{@}_{\emptyset,\emptyset}}\ \mathsf{E}_{\mathsf{NA}}$$

  noting that $P = ((\langle\cdot\rangle\, t, \epsilon)\langle x\rangle)\,@[x{\leftarrow}\langle\cdot\rangle] = (x\,t, [x{\leftarrow}\langle\cdot\rangle])$, and so $p = P\langle z\rangle$ as expected. In this case, we are extending an exponential context, which is already applied.
- $\mathsf{E}_{\mathsf{AX}_2}$: consider $p := (x, [x{\leftarrow}z\,t])$, for which z is an applied variable because it is itself applied, while its ES binds the needed but unapplied variable x. Let us derive an exponential evaluation context P focusing on z in such a way that $P\langle z\rangle = p$ as follows:

$$\dfrac{\dfrac{}{(\langle\cdot\rangle, \epsilon) \in \mathcal{E}_{\emptyset,\emptyset}}\ \mathsf{O}_{\mathsf{AX}} \quad x \notin (\emptyset \cup \emptyset)}{P := ((\langle\cdot\rangle, \epsilon))\,\langle x\rangle @[x{\leftarrow}\langle\cdot\rangle\, t] \in \mathcal{E}^{@}_{\emptyset,\emptyset}}\ \mathsf{E}_{\mathsf{AX}_2}$$

  noting that $P = ((\langle\cdot\rangle, \epsilon))\,\langle x\rangle @[x{\leftarrow}\langle\cdot\rangle\, t] = (x, [x{\leftarrow}\langle\cdot\rangle\, t])$, and so $p = P\langle z\rangle$ as expected. Here the context $(\langle\cdot\rangle, \epsilon)$ in the hypothesis is *multiplicative* and it becomes *exponential* once extended with an ES containing an applicative term context.

The next proposition guarantees that exponential contexts are a restriction of multiplicative contexts, that is, that the introduced variations over the deduction rules do not add contexts that were not already available before.

▶ **Proposition 9** (Exponential contexts are multiplicative). *Let $P \in \mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}$. Then $P \in \mathcal{E}_{\mathcal{V},\mathcal{B}}$, for some $\mathcal{V} \subseteq \mathcal{U}$ and $\mathcal{B} \subseteq \mathcal{A}$.*

Let us repeat that, instead, multiplicative contexts are not in general exponential contexts, because they are not required to be applicative, for instance $P := (x\langle\cdot\rangle, [x{\leftarrow}yy]) \in \mathcal{E}_{\{y\},\{y\}}$ is a multiplicative context but not an exponential one.

Evaluation Rules. The reduction rules for the Useful Open CbNeed strategy are:

Useful Open CbNeed evaluation rules		
Useful multiplicative	$P\langle(\lambda x.t)u\rangle \to_{\mathsf{um}} P\langle t, [x{\leftarrow}u]\rangle$	if $P \in \mathcal{E}_{\mathcal{U},\mathcal{A}}$
Useful exponential	$P\langle x\rangle \to_{\mathsf{ue}} P\langle v\rangle$	if $P \in \mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}$ and $P(x) = v$

Moreover, we shall say that p reduces in the Useful Open CbNeed strategy to q, and write $p \to_{\mathsf{und}} q$, if $p \to_{\mathsf{um}} q$ or $p \to_{\mathsf{ue}} q$.

Determinism. The first property of useful evaluation that we consider is determinism, that is proved similarly for the open case, but for some further technicalities due to the existence of two sets of variables parametrizing evaluation contexts.

▶ **Proposition 10** (Determinism of Useful Open CbNeed). $\to_{\mathsf{und}}$ *is deterministic.*

Usefulness. We prove two properties ensuring that the defined reduction captures useful sharing. The first one is a *correctness* property, stating that useful exponential steps are eventually followed by a multiplicative step – *no useless exponential steps are possible.*

▶ **Proposition 11** (Usefulness of exponential steps). *Let* $p = P\langle x\rangle \to_{\mathsf{ue}} P\langle v\rangle = q$ *with* $P \in \mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}$ *and* $P(x) = v$*. Then there exists a program* r *and a reduction sequence* $d : q \to^{k}_{\mathsf{ue}} \to_{\mathsf{um}} r$ *s.t.:*
1. *the evaluation context of each* $\to_{\mathsf{ue}}$ *steps in* d *is in* $\mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}$*, and the one of* $\to_{\mathsf{um}}$ *is in* $\mathcal{E}_{\mathcal{U},\mathcal{A}}$*.*
2. $k \geq 0$ *is the number of* $\mathsf{E_{NA}}$ *rules in the derivation of* $P \in \mathcal{E}^{@}_{\mathcal{U},\mathcal{A}}$*.*

Completeness amounts to proving that useful normal forms, when unshared, give a Open CbNeed normal term. The point is that useful substitutions, if erroneously designed, might stop too soon, on programs that still contain – up to unsharing – some redexes. Completeness is developed in the following paragraph about useful normal forms.

Useful Normal Forms. We are now going to develop an inductive description of useful normal forms, that is, programs that are $\to_{\mathsf{und}}$-normal. The key property guiding the characterization of a useful normal program p is that if the sharing in p is unfolded (by turning ES into meta-level substitutions and obtaining a term) it produces a normal term of the open system, where the unfolding operation is defined as follows:

$$\text{Unfolding of programs} \qquad (t,\epsilon)\!\downarrow \coloneqq t \qquad\qquad (t, e[x\leftarrow u])\!\downarrow \coloneqq (t,e)\!\downarrow\{x\leftarrow u\}$$

The characterization rests on 3 predicates, defined in Fig. 6, for programs unfolding to variables ($\mathsf{genVar}_x(p)$), values ($\mathsf{uabs}(p)$), and non-variable inert terms ($\mathsf{uinert}(p)$). Programs satisfying $\mathsf{genVar}_x(p)$ are called *generalized variable* of *(hereditary) head* variable x – we also write $\mathsf{genVar}_{\#}(p)$ to state that there exists $x \in \mathsf{Var}$ such that $\mathsf{genVar}_x(p)$. Programs satisfying $\mathsf{uabs}(p)$ (resp. $\mathsf{uinert}(p)$), instead, are *useful abstractions* (resp. *useful inerts*). The predicate $\mathsf{unorm}(p)$ holds for programs satisfying either of the three described predicates, which we shall show being exactly programs that are normal in Useful Open CbNeed.

Generalized variables play a special role, because they can be extended to unfold to values or non-variable inert terms, by appending an appropriate ES to their environment with rule $\mathsf{A_{GV}}$ or $\mathsf{I_{GV}}$. For instance, a useful normal program such as $(x, [x\leftarrow y])$ unfolds to a variable but its useful normal extension $(x, [x\leftarrow y][y\leftarrow \mathsf{I}])$ unfolds to the value I, while $(x, [x\leftarrow y][y\leftarrow zz])$ unfolds to the non-variable inert term zz.

▶ **Proposition 12** (Disjointness and unfolding of useful predicates). *For every program* p*,* at most one *of the following holds:* $\mathsf{genVar}_{\#}(p)$*,* $\mathsf{uabs}(p)$*, or* $\mathsf{uinert}(p)$*. Moreover,*
1. *If* $\mathsf{genVar}_x(p)$ *then* $p\!\downarrow = x$*.*
2. *If* $\mathsf{uabs}(p)$ *then* $p\!\downarrow$ *is a value.*
3. *If* $\mathsf{uinert}(p)$ *then* $p\!\downarrow$ *is a non-variable inert term.*

While the concepts in the characterization of useful normal programs are relatively simple and natural, the proof of the next proposition is long and tedious, because of the complex shape of useful evaluation contexts and of their parametrization, see the technical report [18].

▶ **Proposition 13** (Syntactic characterization of Useful Open CbNeed-normal forms). *Let* p *be a program. Then* p *is in* $\to_{\mathsf{und}}$*-normal form if and only if* $\mathsf{unorm}(p)$*.*

$$\frac{}{\mathsf{genVar}_x(x,\epsilon)}\ \mathsf{GV}_{\mathsf{AX}} \qquad \frac{\mathsf{genVar}_x(p)}{\mathsf{genVar}_y(p@[x{\leftarrow}y])}\ \mathsf{GV}_{\mathsf{HER}} \qquad \frac{\mathsf{genVar}_x(p) \quad z \neq x}{\mathsf{genVar}_x(p@[z{\leftarrow}t])}\ \mathsf{GV}_{\mathsf{GC}}$$

$$\frac{}{\mathsf{uabs}(v,\epsilon)}\ \mathsf{A}_{\mathsf{Lift}} \qquad \frac{\mathsf{genVar}_x(p)}{\mathsf{uabs}(p@[x{\leftarrow}v])}\ \mathsf{A}_{\mathsf{GV}} \qquad \frac{\mathsf{uabs}(p)}{\mathsf{uabs}(p@[x{\leftarrow}t])}\ \mathsf{A}_{\mathsf{GC}}$$

$$\frac{}{\mathsf{uinert}(i^+,\epsilon)}\ \mathsf{I}_{\mathsf{Lift}} \qquad \frac{\mathsf{genVar}_x(p)}{\mathsf{uinert}(p@[x{\leftarrow}i^+])}\ \mathsf{I}_{\mathsf{GV}} \qquad \frac{\mathsf{uinert}(p) \quad x \in \mathsf{nv}(p)}{\mathsf{uinert}(p@[x{\leftarrow}i])}\ \mathsf{I}_{\mathsf{I}}$$

$$\frac{\mathsf{uinert}(p) \quad x \in \mathsf{u}(p) \quad x \notin \mathsf{a}(p)}{\mathsf{uinert}(p@[x{\leftarrow}v])}\ \mathsf{I}_{\mathsf{U}} \qquad \frac{\mathsf{uinert}(p) \quad x \notin \mathsf{nv}(p)}{\mathsf{uinert}(p@[x{\leftarrow}t])}\ \mathsf{I}_{\mathsf{GC}}$$

$$\frac{\mathsf{uinert}(p) \vee \mathsf{uabs}(p) \vee \mathsf{genVar}_{\#}(p)}{\mathsf{unorm}(p)}\ \mathsf{unorm}_P$$

Figure 6 Predicates characterizing Useful Open CbNeed normal forms.

The characterization of useful normal forms together with the fact that they unfold to Open CbNeed normal terms (Lemma 12) express the *completeness* of useful sharing: our useful evaluation does compute – up to unfolding – representations of normal terms.

Complexity. A precise complexity analysis requires an abstract machine implementing the search for redexes specified by evaluation contexts. The machine – which we have developed – is left to a forthcoming paper, for lack of space. Crucially, it avoids tracing sets of applied and unapplied variables by simply using a boolean that indicates – when evaluation moves into the environment – whether the current evaluation position is hereditarily applied.

We provide a sketch of the complexity analysis. The k in point 2 of Proposition 11 allows us to bound any sequence of consecutive $\to_{\mathsf{ue}}$ steps with the length of the environment, which – via the same reasoning used for the CbN case by Accattoli and Dal Lago [13] – gives a quadratic bound to the whole number of $\to_{\mathsf{ue}}$ steps in terms of $\to_{\mathsf{um}}$ steps. A finer amortized analysis, following Accattoli and Sacerdoti Coen [10], gives a linear bound. The cost of duplications in exponential steps $\to_{\mathsf{ue}}$ is bound by the size of the initial program, because the calculus evidently has the *subterm property* (i.e. only subterms of the initial programs are duplicated): it duplicates values but it does not substitute nor evaluate into them, therefore the initial ones are preserved. Then, the cost of implementing a reduction sequence $d : p \to^k_{\mathsf{und}} q$, omitting the cost of searching for redexes (itself usually realized linearly in the size $|p|$ of p by abstract machines [6, 9]), is linear in $|p|$ and in the number $|d|_{\mathsf{m}}$ of multiplicative/β steps in d.

Therefore, the number of multiplicative/β steps in our Useful Open CbNeed calculus is a reasonable time cost model, even realizable within an efficient, bilinear overhead.

References

1 Samson Abramsky and C.-H. Luke Ong. Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 105(2):159–267, 1993.

2 Beniamino Accattoli. The complexity of abstract machines. In *WPTE@FSCD 2016*, pages 1–15, 2016. doi:10.4204/EPTCS.235.1.

3 Beniamino Accattoli. The Useful MAM, a Reasonable Implementation of the Strong λ-Calculus. In *WoLLIC 2016*, pages 1–21, 2016. doi:10.1007/978-3-662-52921-8_1.

4 Beniamino Accattoli. Proof nets and the linear substitution calculus. In Bernd Fischer and Tarmo Uustalu, editors, *Theoretical Aspects of Computing – ICTAC 2018 – 15th International Colloquium, Stellenbosch, South Africa, October 16-19, 2018, Proceedings*, volume 11187 of *Lecture Notes in Computer Science*, pages 37–61. Springer, 2018. doi:10.1007/978-3-030-02508-3_3.

5 Beniamino Accattoli. A fresh look at the lambda-calculus (invited talk). In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 1:1–1:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.1.

6 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.

7 Beniamino Accattoli and Bruno Barras. Environments and the complexity of abstract machines. In *PPDP 2017*, pages 4–16, 2017. doi:10.1145/3131851.3131855.

8 Beniamino Accattoli and Bruno Barras. The negligible and yet subtle cost of pattern matching. In Bor-Yuh Evan Chang, editor, *Programming Languages and Systems – 15th Asian Symposium, APLAS 2017, Suzhou, China, November 27-29, 2017, Proceedings*, volume 10695 of *Lecture Notes in Computer Science*, pages 426–447. Springer, 2017. doi:10.1007/978-3-319-71237-6_21.

9 Beniamino Accattoli and Claudio Sacerdoti Coen. On the relative usefulness of fireballs. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.23.

10 Beniamino Accattoli and Claudio Sacerdoti Coen. On the value of variables. *Inf. Comput.*, 255:224–242, 2017. doi:10.1016/j.ic.2017.01.003.

11 Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively. In *LICS*, pages 1–14. IEEE, 2021.

12 Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling abstract machines. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15. ACM, 2019. doi:10.1145/3354166.3354169.

13 Beniamino Accattoli and Ugo Dal Lago. On the invariance of the unitary cost model for head reduction. In *RTA*, volume 15 of *LIPIcs*, pages 22–37. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

14 Beniamino Accattoli and Ugo Dal Lago. (Leftmost-Outermost) Beta-Reduction is Invariant, Indeed. *Logical Methods in Computer Science*, 12(1), 2016. doi:10.2168/LMCS-12(1:4)2016.

15 Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. In Atsushi Igarashi, editor, *Programming Languages and Systems – 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016. doi:10.1007/978-3-319-47958-3_12.

16 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019. doi:10.1016/j.scico.2019.03.002.

17 Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In Luís Caires, editor, *Programming Languages and Systems – 28th European Symposium on Programming, ESOP 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 11423 of *Lecture Notes in Computer Science*, pages 410–439. Springer, 2019. doi:10.1007/978-3-030-17184-1_15.

18 Beniamino Accattoli and Maico Leberle. Useful open call-by-need. *CoRR*, abs/2107.06591, 2021. URL: https://arxiv.org/abs/2107.06591, arXiv:2107.06591.

19 Zena M. Ariola and Matthias Felleisen. The call-by-need lambda calculus. *J. Funct. Program.*, 7(3):265–301, 1997. URL: `http://journals.cambridge.org/action/displayAbstract?aid=44089`.

20 Zena M. Ariola, Hugo Herbelin, and Alexis Saurin. Classical call-by-need and duality. In C.-H. Luke Ong, editor, *Typed Lambda Calculi and Applications – 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 27–44. Springer, 2011. `doi:10.1007/978-3-642-21691-6_6`.

21 Andrea Asperti and Harry G. Mairson. Parallel beta reduction is not elementary recursive. *Inf. Comput.*, 170(1):49–80, 2001. `doi:10.1006/inco.2001.2869`.

22 Thibaut Balabonski. Weak optimality, and the meaning of sharing. In *ICFP*, pages 263–274. ACM, 2013.

23 Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. Foundations of strong call by need. *PACMPL*, 1(ICFP):20:1–20:29, 2017. `doi:10.1145/3110264`.

24 Thibaut Balabonski, Antoine Lanco, and Guillaume Melquiond. A strong call-by-need calculus. In *FSCD*, volume 195 of *LIPIcs*, pages 9:1–9:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

25 Pablo Barenbaum, Eduardo Bonelli, and Kareem Mohamed. Pattern matching and fixed points: Resource types and strong call-by-need: Extended abstract. In David Sabel and Peter Thiemann, editors, *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*, pages 6:1–6:12. ACM, 2018. `doi:10.1145/3236950.3236972`.

26 Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1984.

27 Bruno Barras. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, Université Paris 7, 1999.

28 Malgorzata Biernacka and Witold Charatonik. Deriving an abstract machine for strong call by need. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.FSCD.2019.8`.

29 Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the weak lambda-calculus. In *Processes, Terms and Cycles*, volume 3838 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2005.

30 Guy E. Blelloch and John Greiner. Parallelism in sequential functional languages. In John Williams, editor, *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995*, pages 226–237. ACM, 1995. `doi:10.1145/224164.224210`.

31 Stephen Chang and Matthias Felleisen. The call-by-need lambda calculus, revisited. In Helmut Seidl, editor, *Programming Languages and Systems – 21st European Symposium on Programming, ESOP 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 7211 of *Lecture Notes in Computer Science*, pages 128–147. Springer, 2012. `doi:10.1007/978-3-642-28869-2_7`.

32 Arthur Charguéraud and François Pottier. Machine-checked verification of the correctness and amortized complexity of an efficient union-find implementation. In *ITP 2015*, pages 137–153, 2015. `doi:10.1007/978-3-319-22102-1_9`.

33 Ugo Dal Lago and Simone Martini. Derivational complexity is an invariant cost model. In Marko C. J. D. van Eekelen and Olha Shkaravska, editors, *Foundational and Practical Aspects of Resource Analysis – First International Workshop, FOPARA 2009, Eindhoven, The Netherlands, November 6, 2009, Revised Selected Papers*, volume 6324 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2009. `doi:10.1007/978-3-642-15331-0_7`.

34 Ugo Dal Lago and Simone Martini. On constructor rewrite systems and the lambda calculus. *Logical Methods in Computer Science*, 8(3), 2012. doi:10.2168/LMCS-8(3:12)2012.

35 Olivier Danvy and Ian Zerny. A synthetic operational account of call-by-need evaluation. In Ricardo Peña and Tom Schrijvers, editors, *15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013*, pages 97–108. ACM, 2013. doi:10.1145/2505879.2505898.

36 Paul Downen, Luke Maurer, Zena M. Ariola, and Daniele Varacca. Continuations, processes, and sharing. In Olaf Chitil, Andy King, and Olivier Danvy, editors, *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, pages 69–80. ACM, 2014. doi:10.1145/2643135.2643155.

37 Ronald Garcia, Andrew Lumsdaine, and Amr Sabry. Lazy evaluation and delimited control. *Log. Methods Comput. Sci.*, 6(3), 2010. URL: http://arxiv.org/abs/1003.5197.

38 Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In Mitchell Wand and Simon L. Peyton Jones, editors, *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002*, pages 235–246. ACM, 2002. doi:10.1145/581478.581501.

39 Jennifer Hackett and Graham Hutton. Call-by-need is clairvoyant call-by-value. *Proc. ACM Program. Lang.*, 3(ICFP):114:1–114:23, 2019. doi:10.1145/3341718.

40 Hugo Herbelin and Étienne Miquey. A calculus of expandable stores: Continuation-and-environment-passing style translations. In *LICS*, pages 564–577. ACM, 2020.

41 Delia Kesner. Reasoning about call-by-need by means of types. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures – 19th International Conference, FOSSACS 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 9634 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 2016. doi:10.1007/978-3-662-49630-5_25.

42 Delia Kesner, Loïc Peyrot, and Daniel Ventura. The spirit of node replication. In *FoSSaCS*, volume 12650 of *Lecture Notes in Computer Science*, pages 344–364. Springer, 2021.

43 Delia Kesner, Alejandro Ríos, and Andrés Viso. Call-by-need, neededness and all that. In *FoSSaCS*, volume 10803 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2018.

44 Arne Kutzner and Manfred Schmidt-Schauß. A non-deterministic call-by-need lambda calculus. In Matthias Felleisen, Paul Hudak, and Christian Queinnec, editors, *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP '98), Baltimore, Maryland, USA, September 27-29, 1998*, pages 324–335. ACM, 1998. doi:10.1145/289423.289462.

45 John Launchbury. A natural semantics for lazy evaluation. In Mary S. Van Deusen and Bernard Lang, editors, *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993*, pages 144–154. ACM Press, 1993. doi:10.1145/158511.158618.

46 Maico Leberle. *Dissecting call-by-need by customizing multi type systems*. Theses, Institut Polytechnique de Paris, May 2021. URL: https://tel.archives-ouvertes.fr/tel-03284370.

47 Jean-Jacques Lévy. Réductions correctes et optimales dans le lambda-calcul. Thése d'Etat, Univ. Paris VII, France, 1978.

48 John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *J. Funct. Program.*, 8(3):275–317, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=44169.

49 Masayuki Mizuno and Eijiro Sumii. Formal verifications of call-by-need and call-by-name evaluations with mutual recursion. In *APLAS*, volume 11893 of *Lecture Notes in Computer Science*, pages 181–201. Springer, 2019.

50 Pierre-Marie Pédrot and Alexis Saurin. Classical by-need. In Peter Thiemann, editor, *Programming Languages and Systems – 25th European Symposium on Programming, ESOP 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 9632 of *Lecture Notes in Computer Science*, pages 616–643. Springer, 2016. doi:10.1007/978-3-662-49498-1_24.

51 David Sands, Jörgen Gustavsson, and Andrew Moran. Lambda calculi and linear speedups. In Torben Æ. Mogensen, David A. Schmidt, and Ivan Hal Sudborough, editors, *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday]*, volume 2566 of *Lecture Notes in Computer Science*, pages 60–84. Springer, 2002. doi:10.1007/3-540-36377-7_4.

52 Peter Sestoft. Deriving a lazy abstract machine. *J. Funct. Program.*, 7(3):231–264, 1997. URL: http://journals.cambridge.org/action/displayAbstract?aid=44087.

53 Christopher P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD Thesis, Oxford, 1971.

Gardening with the Pythia
A Model of Continuity in a Dependent Setting

Martin Baillon ✉
INRIA and LS2N, Nantes, France

Assia Mahboubi ✉
INRIA and LS2N, Nantes, France

Pierre-Marie Pédrot ✉
INRIA and LS2N, Nantes, France

Abstract

We generalize to a rich dependent type theory a proof originally developed by Escardó that all System T functionals are continuous. It relies on the definition of a syntactic model of Baclofen Type Theory, a type theory where dependent elimination must be strict, into the Calculus of Inductive Constructions. The model is given by three translations: the *axiom translation*, that adds an oracle to the context; the *branching translation*, based on the dialogue monad, turning every type into a tree; and finally, a layer of *algebraic binary parametricity*, binding together the two translations. In the resulting type theory, every function $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is externally continuous.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Type theory, continuity, syntactic model

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.5

Supplementary Material *Software (Source Code)*: `https://gitlab.inria.fr/mbaillon/gardening-with-the-pythia`; archived at `swh:1:dir:0cb62da6eae2909b912bdf29e37e9b0e6875ff52`

Introduction

A folklore result from computatibility theory is that any computable function must be continuous [4]. A more operational way to phrase this property is that a function can only inspect a finite amount of its argument to produce a finite amount of output. There are many ways to prove, or even merely state, this theorem, since it depends in particular on how computable functions are represented [18, 37, 21]. Assuming we pick the λ-calculus as our favourite computational system, a modern straightforward proof would boil down to building a semantic model, typically some flavour of complete partial orders (cpos). By construction, cpos are a specific kind of topological spaces, and all functions are interpreted as continuous functions in the model. For some types simple enough, cpo-continuity implies continuity in the traditional sense, thus proving the claim.

Instead of going down the semantic route, Escardó developed an alternative syntactic technique called *effectful forcing* [11] to prove the continuity of all functionals $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ definable in System T. While semantic models such as cpos are defined inside a non-computational metatheory, Escardó's technique amounts to building a model of System T inside the dependent type theory MLTT, which is intrinsically a programming language with a built-in notion of computation. The *effectful* epithet is justified by the fact that the model construction extends System T with two different kinds of side-effects, and constrains those two extensions by a logical relation.

A clear advantage of this approach is that there is a simple computational explanation for why continuity holds in terms of elementary side-effects, which is not immediately apparent in cpos. This computational aspect is reminiscent of a similar realizability model of NuPRL internalizing continuity with a system of fresh exceptions [30]. But contrarily to the latter,

the purely syntactic nature of Escardó's argument can actually be leveraged to interpret much richer languages than System T while preserving desirable properties that would be lost with a semantic realizability model, such as decidability of type-checking.

Indeed, it happens that this technique can be formulated pretty much straightforwardly as a syntactic model [13, 33]. From this initial observation, we show in this paper how it can be generalized to a rich dependent type theory similar to MLTT, notably featuring universes and a form of large dependent elimination. Unfortunately, since Escardó's model introduces observable side-effects in the sense of [25], the type theory resulting from our generalization needs to be slightly weakened down or would otherwise be inconsistent. This effectively means we provide a model of Baclofen Type Theory (BTT) rather than MLTT. The main difference between those two theories lies in the typing rule for dependent elimination [28]. In MLTT, the predicate of a dependent elimination is arbitrary, while it must be computationally strict in BTT. This is discussed in detail in Section 1.2.

In the end we recover the continuity result of Escardó applied to BTT rather than System T. That is, from any $\vdash_{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ we get a proof that it is continuous.

Plan of the paper

Section 1 exposes preliminaries that are needed to understand this paper. In Section 2, we describe a particular structure known as dialogue trees that will be critical for the rest of the paper. Section 3 is dedicated to the model contruction *per se*. Section 4 provides the proof that all functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ of this model are indeed continuous. Section 5 frames our result in a larger context and discusses potential extensions.

1 Preliminaries

1.1 Syntactic Conventions

In this paper, we will work with various flavours of type theory. Our base system will always be CC_ω, a predicative version of the Calculus of Constructions featuring an infinite hierarchy of universes $\square_i$ and dependent functions, which is summarized in Figure 1. We add inductive types to this negative fragment, leading to the Calculus of Inductive Constructions (CIC) or Baclofen Type Theory (BTT) depending on the formulation of dependent elimination. We will summarize the defining features of BTT in the next section. Since we will manipulate several type theories, we will write $\mathsf{T} := \mathsf{CIC}$ as a notational device to make explicit that we are referring to the ambient type theory.

For brevity, we will define inductive types in a Coq-like syntax, but we will use a pattern-matching syntax à la Agda for definitions by induction. As an example, we give below the definition of natural numbers and the resulting formal typing and conversion rules.

$$\texttt{Inductive}\ \mathbb{N} := \mathsf{O} : \mathbb{N} \mid \mathsf{S} : \mathbb{N} \to \mathbb{N}$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathbb{N} : \square_i} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{O} : \mathbb{N}} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{S} : \mathbb{N} \to \mathbb{N}}$$

$$\frac{\Gamma \vdash P : \mathbb{N} \to \square_i \qquad \Gamma \vdash t_{\mathsf{O}} : P\ \mathsf{O} \qquad \Gamma \vdash t_{\mathsf{S}} : \Pi n : \mathbb{N}.\, P\ n \to P\ (\mathsf{S}\ n)}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}} : \Pi n : \mathbb{N}.\, P\ n}$$

$$\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ \mathsf{O} \equiv t_{\mathsf{O}} \qquad \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ (\mathsf{S}\ n) \equiv t_{\mathsf{S}}\ n\ (\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n)$$

$$A, B, M, N ::= \square_i \mid x \mid M\ N \mid \lambda x : A.\, M \mid \Pi x : A.\, M \mid \Sigma x : A.\, B \mid M.\pi_1 \mid M.\pi_2 \mid (M, N)$$

$$\Gamma, \Delta ::= \cdot \mid \Gamma, x : A$$

$$\frac{}{\vdash \cdot} \qquad \frac{\Gamma \vdash A : \square_i}{\vdash \Gamma, x : A} \qquad \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma \qquad i < j}{\Gamma \vdash \square_i : \square_j}$$

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma \vdash M : B}{\Gamma, x : A \vdash M : B} \qquad \frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Pi x : A.\, B : \square_{\max(i,j)}}$$

$$\frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B\{x := N\}} \qquad \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : \square_i}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : \square_i \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash M : B}$$

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Sigma x : A.\, B : \square_{\max(i,j)}} \qquad \frac{\Gamma \vdash M : \Sigma x : A.\, B}{\Gamma \vdash M.\pi_1 : A} \qquad \frac{\Gamma \vdash M : \Sigma x : A.\, B}{\Gamma \vdash M.\pi_2 : B\{x := M.\pi_1\}}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B\{x := M\} \qquad \Sigma x : A.\, B : \square_i}{\Gamma \vdash (M, N) : \Sigma x : A.\, B} \qquad \text{(conversion omitted)}$$

Figure 1 Syntax of CC_ω extended with Σ-types.

We will mostly ignore universe constraints and silently rely on typical ambiguity for the sake of readability. Definitions indexed by universe variables i, j are meant to be universe-polymorphic in those variables [34]. All the translations we will give can be annotated with universe variables to handle an arbitrary hierarchy of universes, but we will refrain from doing so. We sometimes use implicit function arguments, which we bind with braces in definitions.

Writing explicit terms in type theory can quickly become cumbersome for proofs, hence we will omit them when the computational content is not important and write instead an underscore as in $\vdash \_ : A$.

1.2 Dependence in an Effectful Setting

In this paper, we will build type theories that feature computational effects. Regrettably, adding effects to dependent type theory is not without consequences. They make indeed observable the difference between call-by-value and call-by-name [20], a phenomenon that puts us in front of a dilemma [25]. If we stick to by-name, we preserve the behaviour of the negative fragment, i.e. Π-types, but we break dependent elimination. If we stick to by-value, we now preserve dependent elimination, but functions become quite different to what one is used to, as substitution is now restricted to syntactic values. For historical reasons, there is a clear bias in type theory towards by-name, and we will follow the same doctrine.

As explained above, an effectful call-by-name type theory does not support full-blown dependent elimination in general. As dependent elimination is quite a critical feature [23], this might look concerning. Thankfully, most effectful theories we know of support a restricted

form of it, which essentially amounts to forcing the predicate used in the eliminator to be *strict* in its inductive argument[1]. The resulting theory is known as Baclofen Type Theory [28], or BTT for short.

Contrarily to MLTT, which has a single dependent eliminator $\mathcal{I}_{\mathbf{ind}}$ for any given inductive type $\mathcal{I}$, BTT has two eliminators: a non-dependent one $\mathcal{I}_{\mathbf{cse}}$, and a strict dependent one $\mathcal{I}_{\mathbf{rec}}$. These three eliminators enjoy the same computational ι-rules, i.e. they reduce on constructors. The difference lies in their typing rules. The predicate of $\mathcal{I}_{\mathbf{cse}}$ does not depend on its inductive argument, i.e. it is basically simply-typed. Meanwhile, the predicate of $\mathcal{I}_{\mathbf{rec}}$ is wrapped in a *storage operator* [19] $\mathcal{I}_{\mathbf{str}}$ that locally evaluates its argument in a by-value fashion. This guarantees that it will only ever be applied to values, and never to effectful, or non-standard, inductive terms. The important observation is that $\mathcal{I}_{\mathbf{str}}$ can be defined in a systematic way out of $\mathcal{I}_{\mathbf{cse}}$, namely it is simply an η-expansion in CPS style. To make things self-contained, we recall below the BTT eliminators for $\mathbb{N}$.

$$\frac{\Gamma \vdash P : \square \qquad \Gamma \vdash t_{\mathsf{O}} : P \qquad \Gamma \vdash t_{\mathsf{S}} : \mathbb{N} \to P \to P}{\Gamma \vdash \mathbb{N}_{\mathbf{cse}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}} : \mathbb{N} \to P}$$

$$\frac{\Gamma \vdash P : \mathbb{N} \to \square \qquad \Gamma \vdash t_{\mathsf{O}} : \mathbb{N}_{\mathbf{str}}\ \mathsf{O}\ P \qquad \Gamma \vdash t_{\mathsf{S}} : \Pi(n : \mathbb{N}).\, \mathbb{N}_{\mathbf{str}}\ n\ P \to \mathbb{N}_{\mathbf{str}}\ (\mathsf{S}\ n)\ P}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}} : \Pi(n : \mathbb{N}).\, \mathbb{N}_{\mathbf{str}}\ n\ P}$$

where
$$\begin{aligned}&\mathbb{N}_{\mathbf{str}}\ (n : \mathbb{N})\ (P : \mathbb{N} \to \square) : \square :=\\ &\quad \mathbb{N}_{\mathbf{cse}}\ ((\mathbb{N} \to \square) \to \square)\ (\lambda(Q : \mathbb{N} \to \square).\, Q\ \mathsf{O})\\ &\qquad (\lambda(m : \mathbb{N})\ (\_ : (\mathbb{N} \to \square) \to \square)\ (Q : \mathbb{N} \to \square).\, Q\ (\mathsf{S}\ m))\ n\ P.\end{aligned}$$

From within CIC, one can prove that $\Pi(n : \mathbb{N})\ (P : \mathbb{N} \to \square).\, \mathbb{N}_{\mathbf{str}}\ n\ P = P\ n$. Hence, this strictification is akin to double-negation translation, in so far as BTT is finer-grained than CIC, just as LJ is finer-grained than LK where $\neg\neg A \leftrightarrow A$. Note that in particular BTT is a subset of CIC, a fact on which we will rely on silently in this paper.

1.3 Continuity

In the remainder of this article, we suppose given two types $\vdash_{\mathsf{T}} \mathbf{I} : \square_0$ and $\vdash_{\mathsf{T}} \mathbf{O} : \mathbf{I} \to \square_0$. For simplicity, we set them in the lowest universe level, but all of the constructions to come can handle an arbitrary base level by bumping them by an appropriate amount.

The type $\mathbf{I}$ is to be understood as a type of input or questions to a black-box, called an oracle. Dually, $\mathbf{O}$ is the type of output or answers from the oracle. Since $\mathbf{O}$ depends on $\mathbf{I}$, we can encode a pretty much arbitrary interaction. Finally, we define the type of oracles as $\mathbf{Q} := \Pi(i : \mathbf{I}).\, \mathbf{O}\ i$. A reader more inclined towards computer science could also consider that $\mathbf{O}$ and $\mathbf{I}$ describe an interface for system calls, and $\mathbf{Q}$ is the type of operating systems implementing these calls.

Let us formally define the notion of continuity over $\mathbf{Q}$.

▶ **Definition 1.** *Given $\alpha_1, \alpha_2 : \mathbf{Q}$ and $\ell : \mathsf{list}\ \mathbf{I}$, we say that α_1 and α_2 are finitely equal on ℓ, written $\alpha_1 \approx_\ell \alpha_2$ when the following inductively defined predicate holds.*

$$\frac{}{\alpha_1 \approx_{\mathsf{nil}} \alpha_2} \qquad \frac{\alpha_1\ i = \alpha_2\ i \qquad \alpha_1 \approx_l \alpha_2}{\alpha_1 \approx_{(\mathsf{cons}\ i\ \ell)} \alpha_2}$$

[1] As in programming language theory, not as in higher category theory.

▶ **Definition 2.** *We say that a function is continuous when it satisfies the continuity predicate*

$$\begin{array}{lcl} \mathcal{C} & : & \Pi\{A : \square\}.\,(\mathbf{Q} \to A) \to \square \\ \mathcal{C}\ f & := & \Pi(\alpha : \mathbf{Q}).\,\Sigma(\ell : \mathsf{list}\ \mathbf{I}).\,\Pi(\beta : \mathbf{Q}).\,\alpha \approx_\ell \beta \to f\ \alpha = f\ \beta. \end{array}$$

This definition captures in a generic way the intuitive notion that a computable functional only needs a finite amount of information from its argument to produce an output. Note that in particular the list of points ℓ where the function is evaluated depends on the argument α, so this notion of continuity is weaker than uniform continuity, where the two quantifiers for ℓ and α are swapped. Depending on the expressivity of T, one can also consider weaker variants where the existential is squashed with various proof-irrelevant modalities [12, 30, 31].

2 Dialogue Trees and Intensionality

2.1 Talking with Trees

It is now time to justify the title of this article by giving some explanations on the links between trees, oracles and functions. We consider an operator $\mathfrak{D} : \square \to \square$, which given a type $A : \square$, associates the type of well-founded trees, with leaves labelled in A. Each inner node is labelled with a certain $i : \mathbf{I}$ and has $\mathbf{O}\ i$ children. In T, this amounts to the following inductive definition:

$$\texttt{Inductive}\ \mathfrak{D}\ (A : \square) : \square\ :=\ \eta : A \to \mathfrak{D}\ A\ \mid\ \beta : \Pi(i : \mathbf{I}).\,(\mathbf{O}\ i \to \mathfrak{D}\ A) \to \mathfrak{D}\ A.$$

This type of *dialogue trees* is known under several other names and has a lot of close relatives [36, 26, 22, 16, 39]. They can be easily interpreted as functionals of type $\mathbf{Q} \to A$. Intuitively, every inner node is an inert call to an oracle $\alpha : \mathbf{Q}$, and the answer is the label of the leaf. This interpretation is implemented by a recursively defined dialogue function.

$$\begin{array}{lcl} \partial & : & \Pi\{A : \square\}\,(\alpha : \mathbf{Q})\,(d : \mathfrak{D}\ A).\,A \\ \partial\ \alpha\ (\eta\ x) & := & x \\ \partial\ \alpha\ (\beta\ i\ k) & := & \partial\ \alpha\ (k\ (\alpha\ i)). \end{array}$$

▶ **Definition 3** (Eloquent functions). *A function $f : \mathbf{Q} \to A$ is said to be* eloquent *if there is a dialogue tree $d : \mathfrak{D}\ A$ and a proof that $\Pi\alpha : \mathbf{Q}.\ f\ \alpha = \partial\ \alpha\ d$.*

Representing functions as trees is a well-known way to extract intensional content from them [17, 14]. Moreover, elements of $\mathfrak{D}\ A$ being well-founded, we get the following.

▶ **Theorem 4** (Continuity). *Eloquent functions are continuous.*

Proof. The proof of the theorem is straightforward by induction on the dialogue tree d. ◀

This theorem is the fundamental insight of the proof. The rest of the paper is devoted to the construction of a model where every function is eloquent and therefore continuous.

2.2 Liberating the Dialogue Monad

In an extensional enough setting, the $\mathfrak{D}$ type former turns out to be a monad. The η natural transformation is already part of the definition, and we can recursively define a `bind` function:

$$\begin{array}{lcl} \texttt{bind} & : & \Pi\{A\ B : \square\}\,(f : A \to \mathfrak{D}\ B)\,(d : \mathfrak{D}\ A).\,\mathfrak{D}\ B \\ \texttt{bind}\ f\ (\eta\ x) & := & f\ x \\ \texttt{bind}\ f\ (\beta\ i\ k) & := & \beta\ i\ (\lambda(o : \mathbf{O}\ i).\,\texttt{bind}\ f\ (k\ o)) \end{array}$$

▶ **Lemma 5.** *Assuming function extensionality,* $(\mathfrak{D}, \eta, \texttt{bind})$ *is a monad.*

Since we want to build a model of dependent type theory, we need to preserve a call-by-name equational theory, i.e. generated by the unrestricted β-rule. Following [28], this means that we need to interpret types as some kind of $\mathfrak{D}$-algebras. Unfortunately, the standard categorical definition of monads and their algebras is not usable in our context because it fundamentally relies on funext, which is not available in CIC. Thankfully, even by categorical standards, $\mathfrak{D}$ is a very particular monad.

▶ **Definition 6.** *A free monad in* CIC *is a parameterized inductive type* $\mathcal{M} : \square \to \square$ *with a dedicated constructor* $\eta : \Pi(A : \square).\, A \to \mathcal{M}\ A$ *and a finite set of constructors*

$$\mathsf{c}_i : \Pi(A : \square).\, \Phi_i\ (\mathcal{M}\ A) \to \mathcal{M}\ A$$

where $\Phi_i : \square \to \square$ *is a type former syntactically strictly positive in its argument.*

Note that the formal definition of free monad from category theory requires a forgetful functor to specify against what the monad would be free. The closest thing to our definition would be a free monad relatively to pointed functors, but even there our definition is stricter. A free monad can be thought of as a way to extend a type with unspecified, inert side-effects, a trivial form of algebraic effects [27, 1]. Since we have neither QITs [2] nor HITs [38] in CIC, we cannot enforce equations on these effects but we can still go a long way.

Free monads in CIC enjoy a lot of interesting properties. As the name implies, they are indeed monads. Again, the η function is given by definition, and `bind` can be defined functorially by induction similarly to the $\mathfrak{D}$ case. Furthermore, the algebras of a free monad can be described in an intensionally-friendly way.

▶ **Definition 7.** *Given* $\mathcal{M}$ *as above, the type of intensional* $\mathcal{M}$*-algebras is the record type*

$$\boxbb^{\mathcal{M}} := \{A : \square; \ldots; \mathsf{p}_i : \Phi_i\ A \to A; \ldots\}.$$

where the Φ_i *are the same as in Definition 6.*

▶ **Theorem 8.** *Assuming* funext, $\boxbb^{\mathcal{M}}$ *is isomorphic to the usual definition of* $\mathcal{M}$*-algebras.*

Said otherwise, the p_i functions are equivalent to the usual morphism $h_A : \mathcal{M}\ A \to A$ preserving the monadic structure, except that this presentation does not require any equation. This results in the main advantage of intensional algebras, namely that they are closed under product type in a purely intensional setting. That is, if $A : \square$ and $B : A \to \boxbb^{\mathcal{M}}$ then $\Pi x : A.\, (B\ x).\pi_1$ can be equipped with an intensional algebra structure defined pointwise. This solves a similar issue encountered in [35].

It is clear that $\mathfrak{D}$ is a free monad, so we can define similarly intensional $\mathfrak{D}$-algebras.

▶ **Definition 9** (Pythias)**.** *A pythia for* $A : \square$ *is a term* $p_A : \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to A) \to A$.

Per the above theorem, pythias for A are extensionally in one-to-one correspondence with $\mathfrak{D}$-algebra structures over A, but are much better behaved intensionally. This will be the crux of the branching translation from Section 3.3.

3 The Syntactic Model

3.1 Overview

We prove that all BTT functions are continuous using a generalization of Escardó's model. While the latter only provides a model of System T, a simply-typed language, our model accomodates not only dependent types, but also universes and inductive types equipped with a strict form of dependent elimination. It is given as a program translation, and thus belongs to the class of syntactic models [13, 7]. The final model is built in three stages, namely

1. An axiom model (Section 3.2),
2. A branching model (Section 3.3),
3. An algebraic parametricity model (Section 3.4).

The first two models are standalone, and the third one glues them together. Each model can be explained computationally. The axiom model adds an blackbox oracle as a global variable. Asking the oracle is just function application, so there is no internal way to observe calls to the oracle. The branching model does the exact converse, as it provides an oracle in a purely inert way. Every single call to the branching oracle is tracked as a node of a dialogue tree, a representation that is reminiscent of game semantics. Finally, the algebraic parametricity model internalizes the fact that these two interpretations are computing essentially the same thing, behaving like a proof-relevant logical relation.

3.2 Axiom Translation

Let us fix a reserved variable $\alpha : \mathbf{Q}$. The axiom translation simply consists in adding α as the first variable of the context. Everywhere else, this translation is transparent. Reserving a variable has no technical consequence, if we were to use De Bruijn indices it just amounts to shifting them all by one. We will also annotate both free and bound variables with an a subscript for readability of the future parts of the paper, where we mix together different translations. We formally give the translation of the negative fragment in Figure 2.

$$
\begin{array}{lcl@{\qquad}lcl}
[x]_a & := & x_a & [\Box_i]_a & := & \Box_i \\
[\lambda x : A.\, M]_a & := & \lambda x_a : [\![A]\!]_a.\, [M]_a & [\![A]\!]_a & := & [A]_a \\
[M\ N]_a & := & [M]_a\ [N]_a & [\![\cdot]\!]_a & := & \alpha : \mathbf{Q} \\
[\Pi x : A.\, B]_a & := & \Pi x_a : [\![A]\!]_a.\, [\![B]\!]_a & [\![\Gamma,\ x : A]\!]_a & := & [\![\Gamma]\!]_a, x_a : [\![A]\!]_a
\end{array}
$$

Figure 2 Axiom Translation (negative fragment).

▶ **Theorem 10.** *The axiom translation is a trivial syntactic model of* CIC *and hence of* BTT.

3.3 Branching Translation

Using the results from Section 2.2, we can use a simplified form of the weaning construction [28] to define the *branching translation*. It all boils down to interpreting types as intensional $\mathfrak{D}$-algebras, whose type will be defined as

$$\Box^b := \Sigma(A : \Box).\, \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to A) \to A.$$

Figure 3 defines the negative branching translation, which translates a type A as $[A]_b : \Box^b$, i.e. a pair $([\![A]\!]_b, \beta_A)$ where $[\![A]\!]_b : \Box$ and β_A is a pythia for $[\![A]\!]_b$. For readability, we give the translation of types as these two components through a slight abuse of notation.

$$
\begin{array}{lcllcl}
[x]_b & := & x_b & [\![A]\!]_b & := & [A]_b.\pi_1 \\
[\lambda x : A.\, M]_b & := & \lambda x_b : [\![A]\!]_b.\, [M]_b & [\![\cdot]\!]_b & := & \cdot \\
[M\ N]_b & := & [M]_b\ [N]_b & [\![\Gamma, x : A]\!]_b & := & [\![\Gamma]\!]_b, x_b : [\![A]\!]_b \\
[\![\square]\!]_b & := & \square^b & & & \\
\beta_\square & := & \lambda(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \square^b).\, \mho_b & & & \\
[\![\Pi x : A.\, B]\!]_b & := & \Pi x_b : [\![A]\!]_b.\, [\![B]\!]_b & & & \\
\beta_{\Pi x:A.\, B} & := & \lambda(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \Pi x : [\![A]\!]_b.\, [\![B]\!]_b)\,(x : [\![A]\!]_b).\, \beta_B\ i\ (\lambda o : \mathbf{O}\ i.\, k\ o\ x) & & &
\end{array}
$$

Figure 3 Branching Translation (negative fragment).

The main difficulty is to endow $\square^b$ with a $\mathfrak{D}$-algebra structure. Since there is no constraint on this structure, we simply assume as a parameter of the translation a dummy $\mathfrak{D}$-algebra $\mho_b : \square^b$. We will similarly need an inhabitant $\omega_b : \mho_b.\pi_1$ to define dependent elimination. There are many possible choices for $\mho_b$, the simplest one being the unit type which is trivially inhabited and algebraic. As a simple instance of weaning, we get the following.

▶ **Proposition 11** (CC_ω Soundness). *We have the following.*

- *If* $M \equiv_{\mathsf{CC}_\omega} N$ *then* $[M]_b \equiv_{\mathsf{T}} [N]_b$.
- *If* $\Gamma \vdash_{\mathsf{CC}_\omega} M : A$ *then* $[\![\Gamma]\!]_b \vdash_{\mathsf{T}} [M]_b : [\![A]\!]_b$.

The interpretation of inductive types is fairly straightforward. Given an inductive type $\mathcal{I}$, we create an inductive type $\mathcal{I}_b$ whose constructors are the pointwise translation of the constructors of $\mathcal{I}$, together with an additional $\beta_{\mathcal{I}}$ constructor turning it into a free $\mathfrak{D}$-algebra. We give as an example below the translation of $\mathbb{N}$, which will be the running example for the remainder of this paper. Parameters and indices present no additional difficulty and we refer to [28] for more details.

$$\texttt{Inductive}\ \mathbb{N}_b : \square := \mathsf{O}_b : \mathbb{N}_b \ \mid\ \mathsf{S}_b : \mathbb{N}_b \to \mathbb{N}_b \ \mid\ \beta_{\mathbb{N}} : \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to \mathbb{N}_b) \to \mathbb{N}_b.$$

An astute reader would have remarked that $[\![\mathbb{N}]\!]_b$ is not defined in the same way as in Escardó's proof. This particular fact and its consequences are further discussed in Section 5.1.

▶ **Theorem 12.** *For any inductive type* $\mathcal{I}$*, its branching translation* $\mathcal{I}_b$ *is well-typed and satisfies the strict positivity criterion.*

We must now implement the eliminators. We first define the non-dependent ones.

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{cse}}]_b & : & \Pi P : \square^b.\, [\![P]\!]_b \to (\mathbb{N}_b \to [\![P]\!]_b \to [\![P]\!]_b) \to \mathbb{N}_b \to [\![P]\!]_b \\
[\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ \mathsf{O}_b & := & p_{\mathsf{O}} \\
[\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{S}_b\ n) & := & p_{\mathsf{S}}\ n\ ([\mathbb{N}_{\mathbf{cse}}]_b\ \ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n) \\
[\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\beta_{\mathbb{N}}\ i\ k) & := & \beta_P\ i\ (\lambda(o : \mathbf{O}\ i).\, [\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (k\ o))
\end{array}
$$

As $P : [\![\square]\!]_b$, it has a pythia $\beta_P : \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to [\![P]\!]_b) \to [\![P]\!]_b$. Every time we encounter a branching occurence of $\beta_{\mathbb{N}}$, we can thus use β_P and propagate the call recursively in the branches. This is the usual by-name semantics of recursors.

However, problems arise with dependent elimination. Given $P : \mathbb{N}_b \to \square^b$ and subproofs for O_b and S_b, there is no clear way to produce a term of type $(P\ (\beta_{\mathbb{N}}\ i\ k)).\pi_1$. There is actually a good reason for that: if it were possible, this would make T inconsistent [25]. Following [28], we therefore restrict ourselves to a strict dependent elimination, relying on the storage operator $\mathbb{N}_{\mathbf{str}}$ from Section 1.2. Since it is given in direct style, its translation is systematic.

▶ **Lemma 13.** *We have the following conversions.*

1. $[\mathbb{N}_{\mathbf{str}}]_b\ \mathsf{O}_b\ P \equiv P\ \mathsf{O}_b$
2. $[\mathbb{N}_{\mathbf{str}}]_b\ (\mathsf{S}_b\ n)\ P \equiv P\ (\mathsf{S}_b\ n)$
3. $[\mathbb{N}_{\mathbf{str}}]_b\ (\beta_{\mathbb{N}}\ i\ k)\ P \equiv \mho_b$

Note that the two first equations above are a consequence of the conversion rules of $\mathbb{N}_{\mathbf{cse}}$ and thus hold in any model of BTT. Only the last one is specific to the current model at hand. Using this, we define the dependent eliminator below. Thanks to the fact that the predicate is wrapped in a storage operator, it is able to return a dummy term when applied to an effectful argument.

$$\begin{array}{lcl}
\mathbb{N}_{\mathbf{rec}} & : & \Pi P : \mathbb{N} \to \square.\ P\ \mathsf{O} \to \\
 & & \quad (\Pi n : \mathbb{N}.\ \mathbb{N}_{\mathbf{str}}\ n\ P \to \mathbb{N}_{\mathbf{str}}\ (\mathsf{S}\ n)\ P) \to\ \Pi n : \mathbb{N}.\ \mathbb{N}_{\mathbf{str}}\ n\ P \\
{[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ \mathsf{O}_b} & := & p_{\mathsf{O}} \\
{[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{S}_b\ n)} & := & p_{\mathsf{S}}\ n\ ([\mathbb{N}_{\mathbf{rec}}]_b\ \ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n) \\
{[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\beta_{\mathbb{N}}\ i\ k)} & := & \omega_b
\end{array}$$

▶ **Theorem 14.** *The branching translation provides a syntactic model of* BTT.

3.4 Algebraic Parametricity Translation

Following Escardó, we now have to relate the two translations. We achieve this through a third layer of *algebraic parametricity*. There are two major differences compared to Escardó's model [11]. The first one is that the logical relation does not live in the metatheory anymore and is defined as a syntactic model similar to parametricity [5]. This is not unexpected, but it is needed to interpret dependent types in a satisfactory way. The second difference is that the parametricity predicate *itself* must be endowed with an algebraic structure. This was a much more surprising structure that happens to be required to interpret large dependent elimination.

Intuitively, every type $A : \square$ is translated as a predicate $[\![A]\!]_\varepsilon : [\![A]\!]_a \to [\![A]\!]_b \to \square$. Note that $\alpha : \mathbf{Q}$ is implicitly part of the context as in the axiom model. As explained above, we also ask for the predicate to be $\mathfrak{D}$-algebraic in the sense that it must be equipped with a proof

$$\beta_A^\varepsilon : \Pi(x_a : [\![A]\!]_a)\ (i : \mathbf{I})\ (k : \mathbf{O}\ i \to [\![A]\!]_b).\ [\![A]\!]_\varepsilon\ x_a\ (k\ (\alpha\ i)) \to [\![A]\!]_\varepsilon\ x_a\ (\beta_A\ i\ k).$$

We will write the type of such algebraic parametricity predicates as

$$\begin{array}{lcl}
\square^\varepsilon\ (A_a : [\![\square]\!]_a)\ (A_b : [\![\square]\!]_b) & := & \Sigma(A_\varepsilon : [\![A]\!]_a \to [\![A]\!]_b \to \square). \\
 & & \quad \Pi(x_a : [\![A]\!]_a)\ (i : \mathbf{I})\ (k : \mathbf{O}\ i \to [\![A]\!]_b)\ (x_\varepsilon : A_\varepsilon\ x_a\ (k\ (\alpha\ i))). \\
 & & \quad\ A_\varepsilon\ x_a\ (\beta_A\ i\ k)
\end{array}$$

Just as we did for the branching translation, given $A : \square_i$ we define separately the predicate $[\![A]\!]_\varepsilon$ and the proof of parametric algebraicity β_A^ε. We define the translation in Figure 4. As before we also ask for a dummy algebraic predicate $\mho_\varepsilon : \Pi(A : \square).\ \square^\varepsilon\ A\ \mho_b$ which can be taken to be always a trivially inhabited predicate, together with an arbitrary proof $\omega_\varepsilon : \Pi(A : \square)\ (x : A).\ (\mho_\varepsilon\ A).\pi_1\ x\ \omega_b$.

▶ **Theorem 15** (CC_ω Soundness)**.** *We have the following.*

- *If* $M \equiv_{\mathsf{CC}_\omega} N$ *then* $[M]_\varepsilon \equiv_{\mathsf{T}} [N]_\varepsilon$.
- *If* $\Gamma \vdash_{\mathsf{CC}_\omega} M : A$ *then* $[\![\Gamma]\!]_\varepsilon \vdash_{\mathsf{T}} [M]_\varepsilon : [\![A]\!]_\varepsilon\ [M]_a\ [M]_b$.

$$
\begin{array}{lcl}
[\![\square]\!]_\varepsilon & := & \lambda(A_a : [\![\square]\!]_a)\,(A_b : [\![\square]\!]_b).\,\square^\varepsilon\ A_a\ A_b \\
\beta^\varepsilon_\square & := & \lambda(A_a : [\![\square]\!]_a)\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to [\![\square]\!]_b)\,(A_\varepsilon : [\![\square]\!]_\varepsilon\ A_a\ (k\ (\alpha\ i))).\,\mho_\varepsilon\ A_a \\
[x]_\varepsilon & := & x_\varepsilon \\
[\lambda x : A.\,M]_\varepsilon & := & \lambda(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b).\,[M]_\varepsilon \\
[M\ N]_\varepsilon & := & [M]_\varepsilon\ [N]_a\ [N]_b\ [N]_\varepsilon \\
[\![\Pi x : A.\,B]\!]_\varepsilon & := & \lambda(f_a : [\![\Pi x : A.\,B]\!]_a)\,(f_b : [\![\Pi x : A.\,B]\!]_b). \\
 & & \quad \Pi(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b).\,[\![B]\!]_\varepsilon\ (f_a\ x_a)\ (f_b\ x_b) \\
\beta^\varepsilon_{\Pi x:A.\,B} & := & \lambda(f_a : [\![\Pi x : A.\,B]\!]_a)\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to [\![\Pi x : A.\,B]\!]_b). \\
 & & \quad \lambda(f_\varepsilon : [\![\Pi x : A.\,B]\!]_\varepsilon\ f_a\ (k\ (\alpha\ i))). \\
 & & \quad\quad \lambda(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b). \\
 & & \quad\quad\quad \beta^\varepsilon_B\ (f_a\ x_a)\ i\ (\lambda(o : \mathbf{O}\ i).\,k\ o\ x_b)\ (f_\varepsilon\ x_a\ x_b\ x_\varepsilon) \\
[\![A]\!]_\varepsilon & := & [A]_\varepsilon.\pi_1 \\
[\![\cdot]\!]_\varepsilon & := & \alpha : \mathbf{Q} \\
[\![\Gamma, x : A]\!]_\varepsilon & := & [\![\Gamma]\!]_\varepsilon,\ x_a : [\![A]\!]_a,\ x_b : [\![A]\!]_b,\ x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b
\end{array}
$$

Figure 4 Algebraic Parametricity Translation (negative fragment).

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{N}_\varepsilon\ (\alpha : \mathbf{Q}) : \mathbb{N} \to \mathbb{N}_b \to \square := \\
\mid \mathsf{O}_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ \mathsf{O}\ \mathsf{O}_b \\
\mid \mathsf{S}_\varepsilon : \Pi(n_a : \mathbb{N})\,(n_b : \mathbb{N}_b)\,(n_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ n_a\ n_b).\,\mathbb{N}_\varepsilon\ \alpha\ (\mathsf{S}\ n_a)\ (\mathsf{S}_b\ n_b) \\
\mid \beta^\varepsilon_{\mathbb{N}} : \Pi(n_a : \mathbb{N})\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \mathbb{N}_b)\,(n_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ n_a\ (k\ (\alpha\ i))).\,\mathbb{N}_\varepsilon\ \alpha\ n_a\ (\beta_{\mathbb{N}}\ i\ k)
\end{array}
$$

$$
[\mathbb{N}]_\varepsilon := (\mathbb{N}_\varepsilon\ \alpha, \beta^\varepsilon_{\mathbb{N}}\ \alpha) \qquad [\mathsf{O}]_\varepsilon := \mathsf{O}_\varepsilon\ \alpha \qquad [\mathsf{S}]_\varepsilon := \mathsf{S}_\varepsilon\ \alpha
$$

Figure 5 Algebraic Parametricity for $\mathbb{N}$.

The algebraic parametric translation of inductive types sticks closely to the branching one. Given an inductive type $\mathcal{I}$, we create an inductive type $\mathcal{I}_\varepsilon$ whose constructors are the pointwise $[\![\cdot]\!]_\varepsilon$ translation of those of $\mathcal{I}$. An additional constructor $\beta^\varepsilon_{\mathcal{I}}$ freely implements the algebraicity requirement. Since $\alpha : \mathbf{Q}$ is implicitly part of the translated context, we have to take it as a parameter of the translated inductive type and explicitly pass it as an argument when interpreting those types and their proof of algebraicity. We give the translation on our running example in Figure 5. Once again, parameters and indices present no particular problem and are handled similarly to [28].

▶ **Theorem 16.** *For any inductive type $\mathcal{I}$, its algebraic parametricity translation $\mathcal{I}_\varepsilon$ is well typed and satisfies the positivity criterion.*

As for the branching translation, we retrieve a restricted form of dependent elimination based on storage operators. The argument is virtually the same, but now at the level of parametricity, which makes the syntactic burden even heavier since we now have everything repeated three times. To enhance readability, we will use the following shorthand for binders:

$$
\langle x : A \rangle := x_a : [\![A]\!]_a, x_b : [\![A]\!]_b, x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b
$$

and similarly for application to variables. We give the eliminators for our running example in this lighter syntax, which is already the limit of what can be done on paper.

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{cse}}]_\varepsilon & : & \Pi\langle P : \square\rangle\,\langle p_{\mathsf{O}} : P\rangle\,\langle p_{\mathsf{S}} : \mathbb{N} \to P \to P\rangle\,\langle n : \mathbb{N}\rangle. \\
 & & \quad [\![P]\!]_\varepsilon\ [\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_a\ [\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_b
\end{array}
$$

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ \mathsf{O}_\varepsilon & := & p_{\mathsf{O}\varepsilon} \\
[\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\mathsf{S}_\varepsilon\ \langle n\rangle) & := & p_{\mathsf{S}\varepsilon}\ \langle n\rangle\ ([\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \langle n\rangle) \\
[\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\beta^\varepsilon_{\mathbb{N}}\ n_a\ i\ k\ n_\varepsilon) & := & \beta^\varepsilon_P \\
 & & \quad ([\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}]_a\ n_a)\ i \\
 & & \quad (\lambda(o : \mathbf{O}\ i).\ [\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}]_b\ (k\ o)) \\
 & & \quad ([\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ n_a\ (k\ (\alpha\ i))\ n_\varepsilon)
\end{array}
$$

Note that the $\beta^\varepsilon_{\mathbb{N}}$ case explicitly calls the global axiom α to relate the oracular term with the branching one. This is one of the few places that introduce an actual use of the oracle in the translation, by opposition to merely passing it around.

We define $[\mathbb{N}_{\mathbf{str}}]_\varepsilon$ as before, using the fact it is given directly in the source in terms of $\mathbb{N}_{\mathbf{cse}}$. In particular we do not have to write its translation explicitly. Finally, we can define the dependent eliminators, following the same structure as before.

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon & : & \Pi\langle P : \mathbb{N} \to \square\rangle\ \langle p_{\mathsf{O}} : P\ \mathsf{O}\rangle\ \langle p_{\mathsf{S}} : \Pi(n : \mathbb{N}).\ \mathbb{N}_{\mathbf{str}}\ n\ P \to \mathbb{N}_{\mathbf{str}}\ (\mathsf{S}\ n)\ P\rangle. \\
 & & \quad \Pi\langle n : \mathbb{N}\rangle.\ [\![\mathbb{N}_{\mathbf{str}}\ n\ P]\!]_\varepsilon\ [\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_a\ [\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_b
\end{array}
$$

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ \mathsf{O}_\varepsilon & := & p_{\mathsf{O}\varepsilon} \\
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\mathsf{S}_\varepsilon\ \langle n\rangle) & := & p_{\mathsf{S}\varepsilon}\ \langle n\rangle\ ([\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \langle n\rangle) \\
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\beta^\varepsilon_{\mathbb{N}}\ n_a\ i\ k\ n_\varepsilon) & := & \omega_\varepsilon\ (P_a\ n_a)\ ([\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}]_a\ n_a)
\end{array}
$$

Following the results from [28], this translation can be generalized to any inductive type, potentially with parameters and indices. Indeed, it basically amounts to the composition of weaning with binary parametricity.

▶ **Theorem 17.** *Algebraic parametricity is a syntactic model of* BTT*.*

4 Continuity of $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$

This section is dedicated to the proof of the main theorem which we formally state below.

▶ **Theorem 18.** *If* $\vdash_{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ *then* $\vdash_{\mathsf{CIC}} \_ : \mathcal{C}\ f$*.*

Proof. The proof follows the same structure as Escardó's proof for System T, and requires a clever instance of the model described above.
In short, we will define an element $\gamma_b : \mathbb{N}_b \to \mathbb{N}_b$ and lift it as a constant $\gamma : \mathbb{N} \to \mathbb{N}$ in the source theory. Computationally, it behaves as an impure function that tracks the arguments it is called on. We will then use it to prove that f is eloquent, with tree witness $[f\ \gamma]_b \equiv [f]_b\ \gamma_b$. Before getting to the nitty-gritty, we will fix henceforth the oracular type parameters for the remainder of this section as

$$\mathbf{I} := \mathbb{N} \qquad \text{and} \qquad \mathbf{O} := \lambda(i : \mathbf{I}).\ \mathbb{N}.$$

Some results exposed in this section are still independent from this precise choice of oracle. When this is the case, we will stick to the $\mathbf{Q}$ notation to highlight this fact.

Since $\mathbb{N}_b$ is essentially a free algebra, we can define a dialogue function $\partial^{\mathbb{N}}$ similar to the one defined in Section 2.

$$
\begin{array}{lcl}
\partial^{\mathbb{N}} & : & \mathbf{Q} \to \mathbb{N}_b \to \mathbb{N} \\
\partial^{\mathbb{N}}\ \alpha\ \mathsf{O}_b & := & \mathsf{O} \\
\partial^{\mathbb{N}}\ \alpha\ (\mathsf{S}_b\ n_b) & := & \mathsf{S}\ (\partial^{\mathbb{N}}\ \alpha\ n_b) \\
\partial^{\mathbb{N}}\ \alpha\ (\beta_{\mathbb{N}}\ i\ k) & := & \partial^{\mathbb{N}}\ \alpha\ (k\ (\alpha\ i)).
\end{array}
$$

▶ **Proposition 19** (Unicity of specification). *There is a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbf{Q})\,\langle n : \mathbb{N}\rangle.\, n_a = \partial^{\mathbb{N}}\ \alpha\ n_b.$$

Proof. By induction on n_ε. ◀

▶ **Proposition 20** (Generic parametricity). *There is a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbf{Q})\,(n_b : \mathbb{N}_b).\, \mathbb{N}_\varepsilon\ \alpha\ (\partial^{\mathbb{N}}\ \alpha\ n_b)\ n_b.$$

Proof. By induction on n_b. ◀

Let us now define our generic element $\gamma_b : \mathbb{N}_b \to \mathbb{N}_b$.

▶ **Definition 21** (Generic tree). *We define in* T *the* generic tree $\mathfrak{t}$ *as*

$$\begin{array}{lcll} \mathfrak{t} & : & \mathbb{N} \to \mathbb{N}_b & \textit{where} \\ \mathfrak{t} & := & \lambda(n : \mathbb{N}).\, \beta_{\mathbb{N}}\ n\ \eta_{\mathbb{N}} & \end{array} \qquad \begin{array}{lcl} \eta_{\mathbb{N}} & : & \mathbb{N} \to \mathbb{N}_b \\ \eta_{\mathbb{N}}\ \mathsf{O} & := & \mathsf{O}_b \\ \eta_{\mathbb{N}}\ (\mathsf{S}\ n) & := & \mathsf{S}_b\ (\eta_{\mathbb{N}}\ n). \end{array}$$

▶ **Lemma 22** (Fundamental property of the generic tree). *We have a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\,(n : \mathbb{N}).\, \partial^{\mathbb{N}}\ \alpha\ (\mathfrak{t}\ n) = \alpha\ n.$$

Proof. Immediate by the definition of the ∂ function. ◀

▶ **Definition 23** (Generic element). *We define the generic element* $\gamma_b : \mathbb{N}_b \to \mathbb{N}_b$ *as follows.*

$$\begin{array}{lcll} \gamma_b\ n_b & := & \gamma_0\ \mathsf{O}\ n_b & \textit{where} \\ & & & \end{array} \qquad \begin{array}{lcl} \gamma_0 & : & \mathbb{N} \to \mathbb{N}_b \to \mathbb{N}_b \\ \gamma_0\ a\ \mathsf{O}_b & := & \mathfrak{t}\ a \\ \gamma_0\ a\ (\mathsf{S}_b\ n_b) & := & \gamma_0\ (\mathsf{S}\ a)\ n_b \\ \gamma_0\ a\ (\beta_{\mathbb{N}}\ i\ k) & := & \beta_{\mathbb{N}}\ i\ (\lambda o : \mathbb{N}.\, \gamma_0\ a\ (k\ o)). \end{array}$$

Intuitively, γ_b adds a layer to its argument, replacing each leaf by a $\mathfrak{t}\ n$, where n is the number of S_b encountered in the branch. It has the following property.

▶ **Lemma 24** (Fundamental property of the generic element). *We have a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\,(n_b : \mathbb{N}_b).\, \partial^{\mathbb{N}}\ \alpha\ (\gamma_b\ n_b) = \alpha\ (\partial^{\mathbb{N}}\ \alpha\ n_b).$$

Proof. Straightforward by induction on n_b, using Lemma 22 for the O_b case. ◀

▶ **Proposition 25.** *The* γ_b *term can be lifted to a function* $\gamma : \mathbb{N} \to \mathbb{N}$ *in the source theory.*

Proof. It is sufficient to derive the following sequents, the first two being trivial.

$$\alpha : \mathbb{N} \to \mathbb{N} \vdash_{\mathsf{T}} \alpha : [\![\mathbb{N} \to \mathbb{N}]\!]_a \qquad \vdash_{\mathsf{T}} \gamma_b : [\![\mathbb{N} \to \mathbb{N}]\!]_b \qquad \alpha : \mathbb{N} \to \mathbb{N} \vdash_{\mathsf{T}} \gamma_\varepsilon : [\![\mathbb{N} \to \mathbb{N}]\!]_\varepsilon\ \alpha\ \gamma_b$$

For γ_ε, assuming $\langle n : \mathbb{N}\rangle$ we have to prove $[\![\mathbb{N}]\!]_\varepsilon\ (\alpha\ n_a)\ (\gamma_b\ n_b)$. By Proposition 19, this is the same as $[\![\mathbb{N}]\!]_\varepsilon\ (\alpha\ (\partial^{\mathbb{N}}\ \alpha\ n_b))\ (\gamma_b\ n_b)$. By Proposition 24, this is the same as $[\![\mathbb{N}]\!]_\varepsilon\ (\partial^{\mathbb{N}}\ \alpha\ (\gamma_b\ n_b))\ (\gamma_b\ n_b)$. We conclude by Proposition 20. ◀

We can now get to the proof of the main result. Let $\vdash_{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$. Since $\gamma : \mathbb{N} \to \mathbb{N}$ can be reflected from the model into BTT, we can consider the term $\vdash_{\mathsf{BTT}} f\ \gamma : \mathbb{N}$. By soundness, it results in the three terms below.

$$\begin{array}{lll} \alpha : \mathbb{N} \to \mathbb{N} & \vdash_{\mathsf{T}} & [f]_a\ \alpha : \mathbb{N} \\ & \vdash_{\mathsf{T}} & [f]_b\ \gamma_b : \mathbb{N}_b \\ \alpha : \mathbb{N} \to \mathbb{N} & \vdash_{\mathsf{T}} & [f]_\varepsilon\ \alpha\ \gamma_b\ \gamma_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ ([f]_a\ \alpha)\ ([f]_b\ \gamma_b) \end{array}$$

Applying Proposition 19 to $[f]_a$, $[f]_b$ and $[f]_\varepsilon$, we get:

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N}).\, [f]_a\ \alpha = \partial^{\mathbb{N}}\ \alpha\ ([f]_b\ \gamma_b)$$

Since f is a term in BTT that does not use any impure extension of the model, it is easy to check that $[f]_a \equiv f$. Therefore, f is eloquent. By Theorem 4, this implies that f is continuous, which concludes our proof. ◀

5 Discussion and Related Work

5.1 Comparison with Similar Models

As already stated, our proof follows the argument given by Escardó [11] for System T, which can also be found as a close variant by Sterling that uses streams instead of trees [35]. Yet, in order to scale to BTT there are a few non-trivial technical differences in our version that ought to be highlighted.

The first obvious one is that Escardó's model does not really qualify as a syntactic model of System T. Rather, it is a model in a type-theoretic metatheory. The difference is subtle, and lies in the fact that the source language is an AST of the ambient type theory in Escardó's model, while there is no such thing in sight in our variant. Actually, this would not even have been possible because in order to internalize type theory inside itself, one needs some form of induction-recursion to handle universes. Morally, we got rid of the middle man of an overaching standard syntactic model of BTT [3].

Another major difference is that the parametricity predicates must be compatible with the $\mathfrak{D}$-algebra structure of the underlying types. This is needed to interpret large elimination, which is absent from System T. This requirement is thus void in Escardó's model. It was a surprising part of the model design, but in hindsight it is obvious that it would pop up eventually. Furthermore, both to preserve conversion and to scale to richer inductive types, the parametricity predicate needs to be given in an inductive way following the underlying source type, rather than as an ad-hoc equality between two terms.

We emphasized that our interpretation of $\mathbb{N}$ is not the same as Escardó's, which uses instead $[\![\mathbb{N}']\!]_b := \mathfrak{D}\ \mathbb{N}$. The reason for that has been already briefly observed in [35] but it is worth elaborating here. Said bluntly, Escardó's interpretation is actually *not* a model of System T. While it is indeed possible to write a simply-typed eliminator

$$\mathbb{N}'_{\mathbf{cse}} : \Pi(P : \square).\, P \to (\mathbb{N}' \to P \to P) \to P$$

it does not enjoy the correct computational behaviour. Namely, in general

$$\mathbb{N}'_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{S}'\ n) \quad \not\equiv \quad p_{\mathsf{S}}\ n\ (\mathbb{N}'_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n).$$

A typical situation where this equation would break happens when n is an effectful term, i.e. its translation is of the form $\beta\ i\ k$. This can be explained by the fact that recursive constructors in effectful call-by-name need to thunk their arguments, i.e. pattern-matching on the head of an inductive term must not evaluate the subterms of the constructor. This is not the case for Escardó's intepretation, which is closer to a call-by-value embedding of $\mathbb{N}$ in call-by-name. Since dependent type theory makes the requirement that this equation holds in the typing rules themselves, we need to pick the right interpretation of $\mathbb{N}$.

Escardó and Xu also gave related models to internalize uniform continuity [40, 10]. Contrarily to the above one, they build these models out of sheaves, which have also been used similarly by Coquand and Jaber [8, 9]. Sheaves form a locally closed cartesian category,

hence they only implement a small fragment of MLTT. It is well-known that the universe of sheaves is not a sheaf in general, and in particular the existence of universes in the first model is an open problem. We have several remarks to make. First, assuming univalence and HITs in the target theory, it turns out to be straightforward to build a syntactic sheaf model of MLTT [32]. Univalence is typically needed to relax the strict uniqueness requirement of sheaves into its fibrant version.
More interestingly, a closer look at [32] shows that univalent sheafification is basically the HIT

$$\begin{aligned}
&\texttt{Inductive } \mathfrak{S}\ (A : \square) : \square := \\
&\mid \eta : A \to \mathfrak{S}\ A \\
&\mid \beta : \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to \mathfrak{S}\ A) \to \mathfrak{S}\ A \\
&\mid \sigma_1 : \Pi(i : \mathbf{I})\, (x : \mathfrak{S}\ A).\, \beta\ i\ (\lambda(o : \mathbf{O}\ i).\, x) = x \\
&\mid \sigma_2 : \dots
\end{aligned}$$

where $\mathbf{O} : \mathbf{I} \to \mathsf{hProp}$ and σ_2 is such that $(\beta, \sigma_1, \sigma_2)$ prove that $\lambda(x : \mathfrak{S}\ A)\, (\_ : \mathbf{O}\ i).\, x$ defines an equivalence $\mathfrak{S}\ A \cong (\mathbf{O}\ i \to \mathfrak{S}\ A)$. The relationship to $\mathfrak{D}$ is obvious, and leads us to challenge Escardó's claim that the dialogue model is not a sheaf model. The higher equalities are precisely what is missing to implement full dependent elimination, i.e. to ensure that sheafification preserves observational purity. Otherwise said, the dialogue monad is an impure variant of the sheafification monad, giving a curious and unexpected double entendre to the phrase *effectful forcing*.

Rahli et al. [30] give another proof of uniform continuity for NuPRL using a form of delimited exceptions. Computationally, their model tracks the accesses to the argument of functions by passing them exception-raising placeholders. The control flow is inverted w.r.t. Escardó's model, which requires non-terminating realizers, but we believe that the fundamental mechanism is similar. In the same context Rahli et al. [31] defines a sheaf model with bar induction in mind, but this principle is inextricably tied to uniform continuity [6].

5.2 Internalization

In this paper we have constructed a model of BTT that associates to every closed term $\vdash f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ a proof in CIC that it is continuous. Can we do better? First, we know that there is a major limitation. Indeed, MLTT extended with the internal statement

$$\Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\, \mathcal{C}\ f$$

results in an inconsistent theory [12]. We will call this property *internal continuity* below. The proof crucially relies on two ingredients, namely congruence of conversion and large dependent elimination. Thus, there might be hope for BTT where the latter is restricted.

▶ **Theorem 26.** *Internal continuity holds in our model iff it holds in* T.

This is obviously disappointing, since it implies that T is inconsistent. One can then wonder if it is possible to aim for a middle ground, where we internalize the modulus of continuity itself, but keep the computation of this modulus in the target. That is, construct a term of type $\Pi(\alpha : \mathbb{N} \to \mathbb{N})\, \langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \rangle.\, [\![\mathcal{C}\ f]\!]$, where $[\![A]\!]$ stands for the triple $\Sigma(x_a : [\![A]\!]_a)\, (x_b : [\![A]\!]_b).\, [\![A]\!]_\varepsilon\ x_a\ x_b$. The implication regarding the target theory is a bit more subtle.

▶ **Lemma 27.** *If we have* $\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\, \langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \rangle.\, [\![\mathcal{C}\ f]\!]$ *then we can also get a proof that* $\vdash_{\mathsf{T}} \_ : \Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\, f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} f \to \mathcal{C}\ f$*, where* $\sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ *is the canonical setoid equality on the functional type.*

Proof. Let $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ such that $f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} f$, and $\alpha : \mathbb{N} \to \mathbb{N}$ in T. We define a term $\tilde{f} : [\![(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}]\!]$ as follows.

$$[\tilde{f}]_a := f \qquad [\tilde{f}]_b := \lambda(u_b : [\![\mathbb{N}]\!]_b \to [\![\mathbb{N}]\!]_b).\, \eta_{\mathbb{N}}\ (f\ (\lambda n : \mathbb{N}.\, \partial^{\mathbb{N}}\ \alpha\ (u_b\ (\eta_{\mathbb{N}}\ n))))$$

These two terms are proved to be in relation by the parametricity predicate by applying the preservation of pointwise equality followed by an induction on the parametricity proof of the argument. Finally, if we have a term of type $\Pi(\alpha : \mathbb{N} \to \mathbb{N})\, \langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle.\, [\![\mathcal{C}\ f]\!]$, then we have $[\![\mathcal{C}\ \tilde{f}]\!]$ and thus $\mathcal{C}\ f$ by projection. ◀

This lemma implies in particular that if our target theory features funext, internalization of the modulus of continuity implies continuity of all functions $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ in T. Thus, by the aforementioned theorem, our theory is inconsistent. Conversely, if our theory is consistent, internalization of the modulus of continuity is out of reach.

As funext is independent from CIC, internalization of the modulus of continuity is unattainable if our target theory is plain CIC. If our target theory does not feature funext, the diagonalization argument of Escardó and Xu does not work anymore.

However, in BTT it is unclear whether it is possible to construct a similar paradox, or if there exists a model of it which validates the internalization of the modulus of continuity. This is still an open question. We nonetheless conjecture that adding an additional layer of presheaves to allow a varying number of oracles in the context could be the key to realize such a model. Indeed, adding a modal type of exceptions to MLTT is precisely what permits to go from the external Markov's rule [29] to the internal Markov's principle [24]. If we were able to locally create a fresh generic element independent from all the previously allocated ones, it seems that we could turn the external continuity rule into an internal one, mimicking what happens for the implementation of Markov's principle. Fresh exceptions are precisely used by Rahli et al. [30] to get what amounts to an independent generic element at every call, so this argument does not seem far-fetched. We leave this to future work.

5.3 Coq Formalization

The results from this paper have been formalized in Coq using a presentation similar to category with families. It is a shallow embedding in the style of [15], hence in particular all conversions are interpreted as definitional equalities. The development relies on universe polymorphism to implement universes in the model, but it could have been avoided at the cost of duplicating the code for every level existing in the hierarchy. As usual, we use negative pairs to handle context extensions in a definitional way. Apart from this, the development does not make use of any fancier feature from the Coq kernel. The code can be found at `https://gitlab.inria.fr/mbaillon/gardening-with-the-pythia`.

Conclusion

This paper gives a purely syntactic proof that functionals of a rich dependent type theory are continuous. Not only is the argument syntactic, but it is also expressed as a program translation into another dependent type theory. Thus, everything computes by construction and conversion in the source is interpreted as conversion in the target. Despite being a generalization of a simpler proof by Escardó, the dependently-typed presentation gives more insight about the constraints one has to respect for it to work properly, and highlights a few hidden flaws of the original version. Finally, the model gives empirical foothold to the claim

that BTT is a natural setting for dependently-typed effects. We believe it is not merely an ad-hoc set of rules, but a system that keeps appearing in various contexts, and thus a generic effectful type theory.

References

1. Danel Ahman. Handling fibred algebraic effects. *Proc. ACM Program. Lang.*, 2(POPL):7:1–7:29, 2018. doi:10.1145/3158095.
2. Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types. 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 293–310. Springer, 2018. doi:10.1007/978-3-319-89366-2_16.
3. Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20–22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.
4. Michael Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*. Number 6 in Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer, Berlin Heidelberg New York Tokyo, 1985.
5. Jean-Philippe Bernardy and Marc Lasson. Realizability and parametricity in pure type systems. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures – 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2011. doi:10.1007/978-3-642-19805-2_8.
6. Mark Bickford, Liron Cohen, Robert L. Constable, and Vincent Rahli. Computability beyond Church-Turing via choice sequences. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 245–254. ACM, 2018. doi:10.1145/3209108.3209200.
7. Simon Boulier. Extending type theory with syntactic models. *Logic in Computer Science [cs.LO]. École nationale supérieure Mines-Télécom Atlantique*, 2018. URL: https://tel.archives-ouvertes.fr/tel-02007839.
8. Thierry Coquand and Guilhem Jaber. A note on forcing and type theory. *Fundam. Informaticae*, 100(1-4):43–52, 2010. doi:10.3233/FI-2010-262.
9. Thierry Coquand and Guilhem Jaber. A computational interpretation of forcing in type theory. In Peter Dybjer, Sten Lindström, Erik Palmgren, and Göran Sundholm, editors, *Epistemology versus Ontology – Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf*, volume 27 of *Logic, Epistemology, and the Unity of Science*, pages 203–213. Springer, 2012. doi:10.1007/978-94-007-4435-6_10.
10. Martín Escardó and Chuangjie Xu. A constructive manifestation of the Kleene-Kreisel continuous functionals. *Ann. Pure Appl. Log.*, 167(9):770–793, 2016. doi:10.1016/j.apal.2016.04.011.
11. Martin Hötzel Escardó. Continuity of Gödel's System T definable functionals via effectful forcing. *Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2013, New Orleans, LA, USA*, June 23–25, 2013. doi:10.1016/j.entcs.2013.09.010.
12. Martin Hötzel Escardó and Chuangjie Xu. The inconsistency of a brouwerian continuity principle with the Curry-Howard interpretation. *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, Warsaw, Poland*, July 1-3, 2015. doi:10.4230/LIPIcs.TLCA.2015.153.

13 Martin Hofmann. *Extensional constructs in intensional type theory*. CPHC/BCS distinguished dissertations. Springer, 1997.

14 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000. doi:10.1006/inco.2000.2917.

15 Ambrus Kaposi, András Kovács, and Nicolai Kraus. Shallow embedding of type theory is morally correct. In Graham Hutton, editor, *Mathematics of Program Construction – 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, volume 11825 of *Lecture Notes in Computer Science*, pages 329–365. Springer, 2019. doi:10.1007/978-3-030-33636-3_12.

16 Oleg Kiselyov and Hiromi Ishii. Freer monads, more extensible effects. In Ben Lippmeier, editor, *Proceedings of the 8th ACM SIGPLAN Symposium on Haskell, Haskell 2015, Vancouver, BC, Canada, September 3-4, 2015*, pages 94–105. ACM, 2015. doi:10.1145/2804302.2804319.

17 S.C. Kleene. Recursive functionals and quantifiers of finite types revisited I. In J.E. Fenstad, R.O. Gandy, and G.E. Sacks, editors, *Generalized Recursion Theory II*, volume 94 of *Studies in Logic and the Foundations of Mathematics*, pages 185–222. Elsevier, 1978. doi:10.1016/S0049-237X(08)70933-9.

18 G. Kreisel, D. Lacombe, and J.R. Shoenfield. Partial recursive functionals and effective operations. In A. Heyting, editor, *Constructivity in Mathematics*, Studies in Logic and the Foundations of Mathematics, pages 290–297, Amsterdam, 1959. North-Holland. Proc. Colloq., Amsterdam, Aug. 26–31, 1957.

19 Jean-Louis Krivine. Opérateurs de mise en mémoire et traduction de Gödel. *Arch. Math. Log.*, 30(4):241–267, 1990. doi:10.1007/BF01792986.

20 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.

21 John Longley and Dag Normann. *Higher-Order Computability*. Theory and Applications of Computability. Springer, 2015. doi:10.1007/978-3-662-47992-6.

22 Conor McBride. Turing-completeness totally free. In Ralf Hinze and Janis Voigtländer, editors, *Mathematics of Program Construction – 12th International Conference, MPC 2015, Königswinter, Germany, June 29 – July 1, 2015. Proceedings*, volume 9129 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2015. doi:10.1007/978-3-319-19797-5_13.

23 Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types*. Habilitation à diriger des recherches, Université Claude Bernard – Lyon I, December 1996. URL: https://tel.archives-ouvertes.fr/tel-00431817.

24 Pierre-Marie Pédrot. Russian constructivism in a prefascist theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 782–794. ACM, 2020. doi:10.1145/3373718.3394740.

25 Pierre-Marie Pédrot and Nicolas Tabareau. The fire triangle: how to mix substitution, dependent elimination, and effects. *Proc. ACM Program. Lang.*, 4(POPL):58:1–58:28, 2020. doi:10.1145/3371126.

26 Maciej Piróg and Jeremy Gibbons. The coinductive resumption monad. In Bart Jacobs, Alexandra Silva, and Sam Staton, editors, *Proceedings of the 30th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2014, Ithaca, NY, USA, June 12-15, 2014*, volume 308 of *Electronic Notes in Theoretical Computer Science*, pages 273–288. Elsevier, 2014. doi:10.1016/j.entcs.2014.10.015.

27 Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Log. Methods Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:23)2013.

28 Pierre-Marie Pédrot and Nicolas Tabareau. An effectful way to eliminate addiction to dependence. *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland*, June 20-23, 2017. doi:10.1109/LICS.2017.8005113.

29 Pierre-Marie Pédrot and Nicolas Tabareau. Failure is not an option an exceptional type theory. *ESOP 2018 – 27th European Symposium on Programming*, 2018. doi:10.1007/978-3-319-89884-1_9.

30 Vincent Rahli and Mark Bickford. Validating Brouwer's continuity principle for numbers using named exceptions. *Math. Struct. Comput. Sci.*, 28(6):942–990, 2018. doi:10.1017/S0960129517000172.

31 Vincent Rahli, Mark Bickford, Liron Cohen, and Robert L. Constable. Bar induction is compatible with constructive type theory. *J. ACM*, 66(2):13:1–13:35, 2019. doi:10.1145/3305261.

32 Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020. doi:10.23638/LMCS-16(1:2)2020.

33 Pierre-Marie Pédrot Simon Boulier and Nicolas Tabareau. The next 700 syntactical models of type theory. *Certified Programs and Proofs (CPP 2017), Jan 2017, Paris, France. pp.182–194*, 2017. doi:10.1145/3018610.3018620.

34 Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in Coq. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving – 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2014. doi:10.1007/978-3-319-08970-6_32.

35 Jonathan Sterling. Higher order functions and Brouwer's thesis. *Journal of Functional Programming*, 31:e11, 2021. *Bob Harper Festschrift Collection*. doi:10.1017/S0956796821000095.

36 Wouter Swierstra. Data types à la carte. *J. Funct. Program.*, 18(4):423–436, 2008. doi:10.1017/S0956796808006758.

37 G. S. Tseitin. Algorithmic operators in constructive metric spaces. In *Problems of the constructive direction in mathematics. Part 2. Constructive mathematical analysis*, volume 67, pages 295–361, Moscow–Leningrad, 1962. USSR Academy of Sciences.

38 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

39 Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. Interaction trees: representing recursive and impure programs in coq. *Proc. ACM Program. Lang.*, 4(POPL):51:1–51:32, 2020. doi:10.1145/3371119.

40 Chuangjie Xu and Martín Hötzel Escardó. A constructive model of uniform continuity. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, volume 7941 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2013. doi:10.1007/978-3-642-38946-7_18.

Weighted Automata and Expressions over Pre-Rational Monoids

Nicolas Baudru
Aix Marseille Univ, CNRS, LIS, Marseille, France

Louis-Marie Dando
Aix Marseille Univ, CNRS, LIS, Marseille, France

Nathan Lhote
Aix Marseille Univ, CNRS, LIS, Marseille, France

Benjamin Monmege
Aix Marseille Univ, CNRS, LIS, Marseille, France

Pierre-Alain Reynier
Aix Marseille Univ, CNRS, LIS, Marseille, France

Jean-Marc Talbot
Aix Marseille Univ, CNRS, LIS, Marseille, France

Abstract

The Kleene theorem establishes a fundamental link between automata and expressions over the free monoid. Numerous generalisations of this result exist in the literature; on one hand, lifting this result to a weighted setting has been widely studied. On the other hand, beyond the free monoid, different monoids can be considered: for instance, two-way automata, and even tree-walking automata, can be described by expressions using the free inverse monoid. In the present work, we aim at combining both research directions and consider weighted extensions of automata and expressions over a class of monoids that we call pre-rational, generalising both the free inverse monoid and graded monoids. The presence of idempotent elements in these pre-rational monoids leads in the weighted setting to consider infinite sums. To handle such sums, we will have to restrict ourselves to rationally additive semirings. Our main result is thus a generalisation of the Kleene theorem for pre-rational monoids and rationally additive semirings. As a corollary, we obtain a class of expressions equivalent to weighted two-way automata, as well as one for tree-walking automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Weighted Automata and Expressions, Inverse Monoids, Two-Way Automata

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.6

Related Version *Full Version*: `https://arxiv.org/abs/2110.12395` [2]

Funding Supported by the DeLTA ANR project (ANR-16-CE40-0007).

1 Introduction

Automata are a convenient tool for algorithmically processing regular languages. However, when a short and human-readable description is required, regular expressions offer a much more proper formalism. When it comes to weighted automata (and transducers as a special case), the Kleene-Schützenberger theorem [20] relates weighted languages defined by means of such automata on one side, and rational series on the other side. Unfortunately, such expressions seem to fit mainly for one-way machines. Indeed, when it comes to two-way machines, finding adequate formalisms for expressions is not easy [13, 14].

Two-way automata have been studied in the setting of the Boolean semiring in [9]. In this work, Janin and Dicky consider a fragment of the free inverse monoid called overlapping tiles. They show that runs of a two-way automaton can be described as a recognizable language of

overlapping tiles, which are words enriched with a starting and an ending position. Hence, thanks to the Kleene theorem, such two-way runs can be described as regular expressions (over tiles).

A particular class of weighted automata is that of transducers, where weights are words on an output alphabet. For this setting, Alur *et al* proposed in [1] a formalism to describe word transformations given as a deterministic streaming string transducer, a model equivalent with deterministic (or unambiguous) two-way transducers [12]. This formalism is based on some operators defining basic transformations that are composed to define the target transformation. An alternative construction of these expressions starting directly from two-way unambiguous transducers has been proposed in [3]. These expressions have also been extended to run on infinite words [8]. The general case of non-deterministic two-way transducers is much more challenging [13], as these machines may admit infinitely many accepting runs on an input word. While this general case is still open (meaning that no equivalent models of expressions are known), a solution has been proposed for the particular case where both input and output alphabets are unary [6].

For a further weighted generalisation, the ability to sum values computed by different runs on the same input structure (no matter if it is a word, a tree or even a graph) is also crucial in terms of expressiveness. However, not all weighted two-way automata (or weighted one-way automata with ε-transitions) are valid: indeed, as these machines may have infinitely many runs over a single input, it may be the case that the automaton does not provide any semantics for such inputs, infinite sums being not guaranteed to converge. To overcome this issue, additional properties are required over the considered semiring: for instance, *rationally additive semirings* [11] allow one to define valid non-deterministic two-way automata [15].

Our initial motivation was to elaborate on the approach proposed by Janin and Dicky in the setting of weighted languages. As already said, the main ingredient of their approach is to consider the free inverse monoid as an input structure. However, going one step further, we consider a generalisation, namely *pre-rational monoids*. These are monoids M such that for all finite alphabets A and for all morphisms from the free monoid A^* to M, the pre-image of $m \in M$ is a rational language of A^*. This class of monoids contains, in particular, the free inverse monoid. After introducing the monoids and semirings of interest in Section 2, we present our main contributions, which hold for pre-rational monoids and rationally additive semirings:

1. We prove in Section 3 that all weighted automata are valid.
2. We introduce in Section 4 a syntax for weighted expressions and show that the semantics of these expressions is always well-defined.
3. We prove in Section 5 a Kleene-like theorem stating that weighted automata and weighted expressions define the same series.
4. We deal with the particular case of unambiguous automata and expressions in Section 6. More precisely, our conversions are shown to preserve the ambiguity, meaning that an element of the monoid "accepted" k times by a weighted automaton can be "decomposed" in k different ways by the weighted expression we obtain, and vice versa.
5. In Section 7, we apply our results on two-way word automata and tree-walking automata which can be viewed as part of the free inverse monoids (which are pre-rational) and show how expressions are quite natural to write via a variety of examples. As a corollary, we obtain a formalism of expressions equivalent to non-deterministic two-way transducers (relying on the unambiguity result presented in the previous section).

Our results can be understood as a trade-off between the generality of the monoid and that of the semiring. Indeed, instead of rationally additive semirings, one could have considered *continuous semirings* in which all infinite sums are well-defined. On such semirings, weighted

automata are valid on all input monoids [19]. However, our framework allows one to consider semirings that are not continuous, and as a consequence we have to restrict in this case the input monoid. On the other end of the spectrum, restricting oneself to graded monoids (as also done in [19]) allows one to consider any semiring, since only finite sums are then involved. However, the free inverse monoid is a typical example of non-graded monoid.

2 Monoids and semirings

We recall that a *monoid* $(M, \cdot, \varepsilon_M)$ is given by a set M and an associative product $\cdot$ with ε_M as neutral element. For our purpose, we consider special classes of monoids:

▶ **Definition 1.** *A monoid $(M, \cdot, \varepsilon_M)$ is* pre-rational *if for every finite alphabet A, for every morphism $\mu\colon A^* \to M$, and for every $m \in M$, the language $\mu^{-1}(m) \subseteq A^*$ is rational.*

Many natural examples of monoids are pre-rational: the free monoid $(A^*, \cdot, \varepsilon)$ over a finite alphabet A, the natural monoid $(\mathbb{N}, +, 0)$, and even the one completed with an infinite element $(\mathbb{N} \cup \{+\infty\}, +, 0)$. Other examples, of particular interest in this article, are *free inverse monoids* that we study in Section 7. Another non-trivial example of pre-rational monoid is $(\{L \subseteq A^* \mid \varepsilon \in L\}, \cdot, \{\varepsilon\})$, with A a finite alphabet. In contrast, a typical example of monoid that is not pre-rational is the free group generated by one element, or $(\mathbb{Z}, +, 0)$ equivalently. For instance, given the morphism $\mu\colon \{a, \bar{a}\}^* \to \mathbb{Z}$ mapping a to 1 and $\bar{a}$ to -1, then $\mu^{-1}(0) = \left\{w \in \{a, \bar{a}\}^* \mid \quad |w|_a = |w|_{\bar{a}}\right\}$ which is not rational.

Showing pre-rationality might sometimes be challenging, since considering arbitrary alphabets and arbitrary morphisms is not really convenient. An easier definition is however possible for monoids M that are generated by a finite family $G = \{g_1, \dots, g_n\}$ of generators. In this case, consider the canonical morphism φ from the free monoid G^* (considering generators as letters) to M, that consists in evaluating the sequence of generators in M. Then, M is pre-rational if and only if for all $m \in M$, the language $\varphi^{-1}(m) \subseteq G^*$ is rational. Pre-rationality is then easier to check, and this, without much of a restriction: the automata and expressions we will consider thereafter only use a finite set of elements of the monoid as atoms, and we can thus restrict ourselves to the finitely generated submonoid. An even simpler sufficient condition for pre-rationality is:

▶ **Lemma 2.** *If every element m of a monoid M has a finite number of* prefixes*, i.e. elements $p \in M$ such that there exists $p' \in M$ with $m = p \cdot p'$, then M is pre-rational.*

Proof. For a finite alphabet A and a morphism $\mu\colon A^* \to M$, and an element $m \in M$, with $\{m_1, \dots, m_n\}$ as finite set of prefixes, we can build a finite automaton reading letters of A and, after having read a word $w \in A^*$, storing the current element $\mu(w)$ when it is a prefix of m (going to a non-accepting sink state otherwise). This automaton can then be used to recognise $\mu^{-1}(m)$, by starting in the prefix ε_M and accepting in the prefix m. ◀

This allows us to easily show that all finitely generated graded monoids [19] (i.e. monoids M equipped with a gradation $\varphi\colon M \to \mathbb{N}$ such that $\varphi(m) = 0$ only if $m = \varepsilon_M$, and $\varphi(mn) = \varphi(m) + \varphi(n)$ for all $m, n \in M$) are pre-rational. Indeed, the gradation ensures that each element $m \in M$ can have only a finite number of prefixes [19, Chap. III, Cor. 1.2,p.384], allowing us to apply the previous lemma. However, notice that the condition in Lemma 2 is not a necessary one: $(\mathbb{N} \cup \{+\infty\}, +, 0)$ does not fulfil the condition, since $+\infty$ has infinitely many factors, while it is indeed pre-rational.

A *semiring* $(\mathbb{K}, +, \times, 0, 1)$ is an algebraic structure such that $(\mathbb{K}, \times, 1)$ is a monoid, $(\mathbb{K}, +, 0)$ is a commutative monoid, the product $\times$ distributes over the sum $+$, and 0 is absorbing for $\times$. Once again, we consider special classes of semirings, introduced in [11]:

▶ **Definition 3.** *A semiring* $(\mathbb{K}, +, \times, 0, 1)$ *is* rationally additive *if it is equipped with a partial operator defining sums of countable families, associating with some infinite families* $(\alpha_i)_{i\in I}$*, with* I *at most countable, an element* $\sum_{i\in I} \alpha_i$ *of* $\mathbb{K}$ *such that for all families* $(\alpha_i)_{i\in I}$*:*

Ax.1 *If* I *is finite, the value* $\sum_{i\in I} \alpha_i$ *exists and coincides with the usual sum in the semiring.*

Ax.2 *For each* $\alpha \in \mathbb{K}$*,* $\sum_{n=0}^{\infty} \alpha^n$ *exists.*

Ax.3 *If* $\sum_{i\in I} \alpha_i$ *exists and* $\beta \in \mathbb{K}$*, then* $\sum_{i\in I} \beta\alpha_i$ *and* $\sum_{i\in I} \alpha_i\beta$ *exist, and are respectively equal to* $\beta(\sum_{i\in I} \alpha_i)$ *and* $(\sum_{i\in I} \alpha_i)\beta$*.*

Ax.4 *Let* I *be the disjoint union of* $(I_j)_{j\in J}$ *with* J *at most countable. If for all* $j \in J$*,* $r_j = \sum_{i\in I_j} \alpha_i$ *exists, and if* $r = \sum_{j\in J} r_j$ *exists, then* $\sum_{i\in I} \alpha_i$ *exists and is equal to* r*.*

Ax.5 *Let* I *be the disjoint union of* $(I_j)_{j\in J}$ *with* J *at most countable. If* $s = \sum_{i\in I} \alpha_i$ *exists, and for all* $j \in J$*,* $r_j = \sum_{i\in I_j} \alpha_i$ *exists, then* $\sum_{j\in J} r_j$ *exists and is equal to* s*.*

Examples of rationally additive semirings are the Boolean semiring, natural semirings over positive rationals or reals $(\mathbb{Q}_+ \cup \{\infty\}, +, \times, 0, 1)$[1], the tropical (or arctic) semiring $(\mathbb{Q}\cup\{-\infty, +\infty\}, \sup, +, -\infty, 0)$, the language semiring over a finite alphabet $(2^{A^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$, the sub-semiring of rational languages, or distributive lattices. Throughout this article, $\mathbb{K}$ will denote a rationally additive semiring.

Let us state a few useful properties of rationally additive semirings. The *support* of a family $(\alpha_i)_{i\in I}$ is the set $\{i \in I \mid \alpha_i \neq 0\}$ of indices of non-zero elements.

▶ **Lemma 4** ([11])**.** *Let* $(\alpha_i)_{i\in I}$ *be a countable family in* $\mathbb{K}$*, of support* J*. Then,* $\sum_{i\in I} \alpha_i$ *exists if and only if* $\sum_{i\in J} \alpha_i$ *exists, and when these sums exist, they are equal.*

▶ **Lemma 5.** *Let* $(\alpha_i)_{i\in I}$ *and* $(\beta_i)_{i\in I}$ *be two countable families of* $\mathbb{K}$ *of disjoint supports, i.e. for all* $i \in I$*,* $\alpha_i = 0$ *or* $\beta_i = 0$ *(or both). If* $\sum_{i\in I} \alpha_i$ *and* $\sum_{i\in I} \beta_i$ *exist, then* $\sum_{i\in I}(\alpha_i+\beta_i)$ *exists and is equal to* $(\sum_{i\in I} \alpha_i) + (\sum_{i\in I} \beta_i)$*.*

Proof. Let J_α and J_β be the support of the families $(\alpha_i)_{i\in I}$ and $(\beta_i)_{i\in I}$, and $J_0 = J\setminus(J_\alpha\cup J_\beta)$. If $\sum_{i\in I} \alpha_i$ and $\sum_{i\in I} \beta_i$ exist, $\sum_{i\in I} \alpha_i+\sum_{i\in I} \beta_i$ exists, and by Lemma 4, is equal to $\sum_{i\in J_\alpha} \alpha_i+\sum_{i\in J_b} \beta_i$. Since the supports are disjoint, this is equal to $\sum_{i\in J_\alpha}(\alpha_i + \beta_i) + \sum_{i\in J_b}(\alpha_i + \beta_i)$. By definition of J_0, $\sum_{i\in J_0}(\alpha_i + \beta_i)$ exists and is equal to 0. Therefore, $\sum_{i\in I} \alpha_i + \sum_{i\in I} \beta_i$ is equal to $\sum_{i\in J_\alpha}(\alpha_i + \beta_i) + \sum_{i\in J_b}(\alpha_i + \beta_i) + \sum_{i\in J_0}(\alpha_i + \beta_i)$. **Ax.**4 allows us to conclude. ◀

▶ **Lemma 6.** *Let* $(\alpha_{i,j})_{(i,j)\in I\times J}$ *be a countable family of elements of* $\mathbb{K}$*, such that* $\alpha_{i,J} = \sum_{j\in J} \alpha_{i,j}$ *exists for all* $i \in I$*, and* $\alpha_{I,j} = \sum_{i\in I} \alpha_{i,j}$ *exists for all* $j \in J$*. Then,* $\sum_{i\in I} \alpha_{i,J}$ *exists if and only if* $\sum_{j\in J} \alpha_{I,j}$ *exists, and when these sums exists, they are equal.*

Proof. Immediate by **Ax.**4 and **Ax.**5. ◀

3 Series and Weighted Automata

A $\mathbb{K}$-*series over* M is a mapping $s\colon M \to \mathbb{K}$ associating a weight $s(m)$ with each element m of the monoid. The set of all such series is denoted by $\mathbb{K}\langle\langle M\rangle\rangle$. Notice that the pointwise sum of two series s_1 and s_2, defined for all $m \in M$ by $(s_1 + s_2)(m) = s_1(m) + s_2(m)$, is a series. However, the Cauchy product $s_1 \cdot s_2$ mapping m to the possibly infinite sum $\sum_{m_1m_2=m} s_1(m_1) \times s_2(m_2)$ might not exist[2]. We define two canonical injections: $M \to \mathbb{K}\langle\langle M\rangle\rangle$ which maps m to the characteristic function of m (mapping m to 1 and the other

[1] All infinite sums of elements in $\mathbb{Q}_+$ do not converge towards a rational number or $+\infty$, but all *geometric sums* do. In particular, this semiring is not *continuous* (see [19, Chap. III, Sec. 5]).

[2] Here and in the following, $\sum_{m_1m_2=m}$ is the sum over all pairs $(m_1, m_2) \in M^2$ such that $m_1m_2 = m$.

elements from M to 0), and $\mathbb{K} \to \mathbb{K}\langle\langle M \rangle\rangle$ which maps k to the function mapping the neutral element ε_M of M to k and all other values to 0. For this reason, we often abuse notations and consider $\mathbb{K}$ and M as subsets of $\mathbb{K}\langle\langle M \rangle\rangle$.

We now introduce the notion of weighted automata we consider in this article: weights are taken from a rationally additive semiring $\mathbb{K}$ and *labels* from a pre-rational monoid M.

▶ **Definition 7.** *A* $\mathbb{K}$-automaton over M*, or simply a* weighted automaton*, is a tuple* $\mathcal{A} = (Q, I, \Delta, F)$*, with* Q *a finite set of states,* $I \subseteq Q$ *the set of initial states,* $\Delta \subseteq Q \times M \times \mathbb{K} \times Q$ *the finite set of transitions each equipped with a label in* M *and a weight in* $\mathbb{K}$*, and* $F \subseteq Q$ *the set of final states.*

We introduce two mappings $\lambda_{\mathcal{A}}$ and $\pi_{\mathcal{A}}$ that extract the label and the weight of a transition, that we can extend to morphisms from Δ^* to M and the multiplicative monoid of $\mathbb{K}$, respectively. A *run* of $\mathcal{A}$ is then a sequence w of transitions $(p_i, m_i, k_i, q_i)_{1 \leq i \leq n}$ such that for all i, $q_i = p_{i+1}$. The *label* of a run is given by $\lambda_{\mathcal{A}}(w)$; its weight is $\pi_{\mathcal{A}}(w)$. The run is said to be *accepting* if $p_1 \in I$ and $q_n \in F$. We let $R_{\mathcal{A}} \subseteq \Delta^*$ denote the rational language of all accepting runs. The *semantics* of $\mathcal{A}$ is the series $[\![\mathcal{A}]\!]$ such that for all $m \in M$, the weight $[\![\mathcal{A}]\!](m)$ is the sum of the weights of accepting runs that are labelled by m, if the (potentially infinite) sum exists: $[\![\mathcal{A}]\!](m) = \sum_{w \in R_{\mathcal{A}} \cap \lambda_{\mathcal{A}}^{-1}(m)} \pi_{\mathcal{A}}(w)$.

The automaton $\mathcal{A}$ is called *valid* if the sum in $[\![\mathcal{A}]\!](m)$ exists for all $m \in M$. Instead of enforcing properties on the automata for them to be valid, we ensure their validity by combining the rational additivity of $\mathbb{K}$ and the pre-rationality of M. The crucial technical property considers the special case of the monoid of strings A^* over a finite alphabet A. We then lift the result using pre-rationality. For a language $L \subseteq A^*$ and a semiring $\mathbb{K}$, we denote by $\chi_L \in \mathbb{K}\langle\langle A^* \rangle\rangle$ its *characteristic series* in $\mathbb{K}$, defined for all $w \in A^*$ as $\chi_L(w) = 1$ if $w \in L$, and 0 otherwise. By Lemma 4, we have that for all series s over A^*,

$$\sum_{w \in L} s(w) \text{ is defined iff } \sum_{w \in A^*} s(w)\chi_L(w) \text{ is defined, and then these sums are equal.} \tag{1}$$

▶ **Lemma 8.** *For every finite alphabet* A*, morphism* $\pi \colon A^* \to \mathbb{K}$*, and rational language* $L \subseteq A^*$*, the sum* $\sum_{w \in L} \pi(w)$ *exists.*

Proof. The proof is by induction on rational languages, denoted by unambiguous regular expressions [5]. Indeed, all rational languages can be obtained by closing the set of finite languages by the operations of disjoint unions, unambiguous concatenations (the concatenation $L_1 \cdot L_2$ is unambiguous when each word w of $L_1 \cdot L_2$ can be uniquely decomposed as $w = w_1 \cdot w_2$ with $w_1 \in L_1$ and $w_2 \in L_2$), and unambiguous Kleene stars (the Kleene star L^* is unambiguous when each word $w \in L^*$ can be uniquely decomposed as $w = w_1 \cdots w_n$ with $n \in \mathbb{N}$ and $w_i \in L$ for all i). Please note that for convenience, the sentences "$A = B$" should be read as "B exists and is equal to A".

First, for finite languages L, the sum $\sum_{w \in L} \pi(w)$ exists, by **Ax.**1. In the case where L is the disjoint union of two languages L_1 and L_2, such that $\sum_{w \in L_1} \pi(w)$ and $\sum_{w \in L_2} \pi(w)$ exist,

$$\begin{aligned}
\sum_{w \in L_1} \pi(w) + \sum_{w \in L_2} \pi(w) &= \sum_{w \in A^*} \pi(w)\chi_{L_1}(w) + \sum_{w \in A^*} \pi(w)\chi_{L_2}(w) && \text{(by 1)} \\
&= \sum_{w \in A^*} (\pi(w)\chi_{L_1}(w) + \pi(w)\chi_{L_2}(w)) && \text{(by Lemma 5)} \\
&= \sum_{w \in A^*} \pi(w)\chi_{L_1 \cup L_2}(w) && \text{(disjoint union)} \\
&= \sum_{w \in L_1 \cup L_2 = L} \pi(w).
\end{aligned}$$

If L is the unambiguous concatenation of two languages L_1 and L_2 such that $\sum_{u\in L_1}\pi(u)$ and $\sum_{v\in L_2}\pi(v)$ exist, then

$$\begin{aligned}\Big(\sum_{u\in L_1}\pi(u)\Big)\times\Big(\sum_{v\in L_2}\pi(v)\Big) &= \sum_{u\in L_1}\Big(\pi(u)\times\sum_{v\in L_2}\pi(v)\Big) && \text{(by \textbf{Ax.}3)}\\ &= \sum_{u\in L_1}\sum_{v\in L_2}\pi(u)\pi(v) && \text{(by \textbf{Ax.}3)}\\ &= \sum_{(u,v)\in L_1\times L_2}\pi(u)\pi(v) && \text{(by \textbf{Ax.}4)}\\ &= \sum_{(u,v)\in L_1\times L_2}\pi(uv) && (\pi \text{ is a morphism}).\end{aligned}$$

Moreover, by unambiguity, there exists a bijection from the pairs of $L_1\times L_2$ to the words of the concatenation $L_1\cdot L_2$ sending (u,v) to uv. Bijections on the support of families conserve the summability property by [11, Proposition 3], therefore $\sum_{w\in L}\pi(w)$ exists (and is equal to $\sum_{(u,v)\in L_1\times L_2}\pi(uv)$).

Finally, suppose that L is the unambiguous Kleene star L_1^*, and $\sum_{w\in L_1}\pi(w)$ exists. In particular, for all $n\in\mathbb{N}$, the iterated concatenation L_1^n is unambiguous, and thus, with a straightforward induction using the previous case, $\sum_{w\in L_1^n}\pi(w)$ exist and we have

$$\Big(\sum_{w\in L_1}\pi(w)\Big)^n = \sum_{w\in L_1^n}\pi(w).$$

By **Ax.**2, $\sum_{n=0}^{\infty}\Big(\sum_{w\in L_1}\pi(w)\Big)^n$ exists, and by (1), we have:

$$\sum_{n=0}^{\infty}\Big(\sum_{w\in L_1}\pi(w)\Big)^n = \sum_{n=0}^{\infty}\sum_{w\in L_1^n}\pi(w) = \sum_{n=0}^{\infty}\sum_{w\in A^*}\pi(w)\chi_{L_1^n}(w).$$

By unambiguity, for all $w\in A^*$, the infinite sum $\sum_{n=0}^{\infty}\pi(w)\chi_{L_1^n}(w)$ has finite support (at most 1 non-zero element) and therefore exists (by Lemma 4). By Lemma 6, we deduce that

$$\begin{aligned}\sum_{n=0}^{\infty}\sum_{w\in A^*}\pi(w)\chi_{L_1^n}(w) &= \sum_{w\in A^*}\sum_{n=0}^{\infty}\pi(w)\chi_{L_1^n}(w) = \sum_{w\in A^*}\pi(w)\sum_{n=0}^{\infty}\chi_{L_1^n}(w) && \text{(by \textbf{Ax.}3)}\\ &= \sum_{w\in A^*}\pi(w)\chi_{L_1^*}(w) && \text{(by unambiguity)}\\ &= \sum_{w\in L}\pi(w). && \blacktriangleleft\end{aligned}$$

From this result, to have a sufficient condition for validity we only need to have sums over rational languages, hence our requirement that M is pre-rational.

▶ **Theorem 9.** *If M is a pre-rational monoid, then every $\mathbb{K}$-automaton $\mathcal{A}$ over M is valid, i.e. $[\![\mathcal{A}]\!](m)$ exists for all $m\in M$.*

Proof. Since M is pre-rational, the morphism $\lambda_{\mathcal{A}}$ is such that for all $m\in M$, $\lambda_{\mathcal{A}}^{-1}(m)$ is a rational language. Therefore, so is the language $R_{\mathcal{A}}\cap\lambda_{\mathcal{A}}^{-1}(m)$ of accepting runs that are labelled by the element m. Lemma 8 gives that $[\![\mathcal{A}]\!](m)=\sum_{w\in R_{\mathcal{A}}\cap\lambda_{\mathcal{A}}^{-1}(m)}\pi_{\mathcal{A}}(w)$ exists. ◀

Together with reasonable assumptions on computability for $\mathbb{K}$ and M, this also gives a procedure to evaluate the weight $[\![\mathcal{A}]\!](m)$. Notice that this is a priori non-trivial, since it involves an infinite sum. We say that M is *effectively* pre-rational if for all morphisms $\mu\colon A^*\to$

M and elements $m \in M$, one can compute a representation of the rational language $\mu^{-1}(m)$. We say that $\mathbb{K}$ is *computable* if internal operations (finite sums and products) of $\mathbb{K}$ are computable, as well as Kleene star (geometric sum). Observe that we do not require computability of arbitrary infinite sums, but only geometric ones.

▶ **Proposition 10.** *If M is effectively pre-rational and $\mathbb{K}$ is computable, then for all $\mathbb{K}$-automata $\mathcal{A}$ over M and all elements $m \in M$, one can compute $[\![\mathcal{A}]\!](m)$. This computation is achieved using a number of internal operations of $\mathbb{K}$ (i.e. sum, product and Kleene iteration) that is polynomial in the size of $\mathcal{A}$ and in the size of a deterministic automaton recognising $\lambda_{\mathcal{A}}^{-1}(m)$.*

Proof. By assumption of pre-rationality, the language $\lambda_{\mathcal{A}}^{-1}(m)$ is rational. Moreover, by effectiveness, we can let $\mathcal{D}_m$ be a deterministic automaton that recognises $\lambda_{\mathcal{A}}^{-1}(m)$. We denote by n_m its number of states. The $\mathbb{K}$-automaton $\mathcal{A}_m$ obtained by considering the product of $\mathcal{A}$ and $\mathcal{D}_m$ (with respect to the alphabet Δ of transitions of $\mathcal{A}$) thus restricts the runs of $\mathcal{A}$ to the ones labelled by m. In addition, as $\mathcal{D}_m$ is deterministic, the accepting runs of $\mathcal{A}$ over m are in bijection with those of $\mathcal{A}_m$. If we denote by n the number of states of $\mathcal{A}$, then $\mathcal{A}_m$ has $n \times n_m$ states. By removing all labels (replacing them by ε_M), we obtain a $\mathbb{K}$-automaton that associates with the element ε_M the weight $[\![\mathcal{A}_m]\!](\varepsilon_M) = [\![\mathcal{A}]\!](m)$. Applying classical translations from automata to regular expressions such as state-elimination algorithms yields an expression equivalent to $[\![\mathcal{A}]\!](m)$. This expression involves sum and product in $\mathbb{K}$, as well as Kleene star, which can be computed in $\mathbb{K}$. As this expression only involves element ε_M, it can be evaluated during its computation, allowing to obtain the value of $[\![\mathcal{A}]\!](m)$ using a number of internal operations of $\mathbb{K}$ that is polynomial in n and n_m. ◀

4 Weighted Expressions

We now introduce the formalism of weighted expressions to generate $\mathbb{K}$-series over a monoid M.

▶ **Definition 11.** *The set of* $\mathbb{K}$-expressions over M, *or simply* weighted expressions, *is generated by the grammar (with $k \in \mathbb{K}$ and $m \in M$):*

$$W ::= k \mid m \mid W + W \mid W \cdot W \mid W^*.$$

Expressions k and m are said to be *atomic*. We call *subexpressions* of W all the weighted expressions appearing inside W: for instance, the subexpressions of $W = (2 \cdot a + b)^*$ are 2, a, b, $2 \cdot a$, $2 \cdot a + b$, and W. To define the semantics of weighted expressions, we will use a sum operator over infinite families. As the semiring $\mathbb{K}$ is supposed to be rationally additive, some of these infinite sums exist, some others do not[3]. Then, the semantics of a weighted expression W is the series $[\![W]\!] \in \mathbb{K}\langle\langle M \rangle\rangle$ defined inductively as follows:

- $[\![k]\!]$ is the series mapping ε_M to k and other elements to 0;
- $[\![m]\!]$ is the characteristic series of m;
- $[\![U + V]\!] = [\![U]\!] + [\![V]\!]$;
- for all $m \in M$, $[\![U \cdot V]\!](m) = \sum_{m_1 m_2 = m} [\![U]\!](m_1) \times [\![V]\!](m_2)$ if the sum exists;
- for all $m \in M$, $[\![W^*]\!](m) = \sum_{n=0}^{\infty} [\![W^n]\!](m)$ if the sum exists (with W^n the expression inductively defined by 1 if $n = 0$ and $W \cdot W^{n-1}$ otherwise).

[3] In the rationally additive semiring $(\mathbb{Q}_+ \cup \{\infty\}, +, \times, 0, 1)$, the infinite sum $\sum_{i \in \mathbb{N}} 1/i!$ does not exist, since it converges to the non-rational real number e.

The last two cases, defining the semantics of the concatenation (or Cauchy product) of two weighted expressions, and the Kleene star of a weighted expression, are subject to the existence of the infinite sums: we say that a weighted expression is *valid* when its semantics exists for all $m \in M$ (as well as the semantics of all its subexpressions, in particular).

More usual regular expressions are recovered by considering the Boolean semiring and the monoid A^* over a finite alphabet A: in the following, such expressions are called *Kleene expressions*, and denoted by letters E, F, G, while weighted expressions are denoted by letters U, V, W. Notice that Kleene expressions are valid, as expected, since the infinite *sum* (i.e. disjunction in the Boolean semiring) is always defined in this case. Their semantics $[\![E]\!]$ is the characteristic series of the language $\mathcal{L}(E)$ classically associated with such a regular expression: alternatively, we can see $\mathcal{L}(E)$ as the support of $[\![E]\!]$ (all words $w \in A^*$ such that $[\![E]\!](w)$ is true). For any other semiring $\mathbb{K}$, we let χ_E be the characteristic function of the language of E to the semiring $\mathbb{K}$, i.e. a shortcut notation for the series $\chi_{\mathcal{L}(E)} \in \mathbb{K}\langle\langle A^* \rangle\rangle$ defined in Section 3.

We shall see that thanks to our hypothesis of $\mathbb{K}$ being rationally additive, and restricting ourselves to pre-rational monoids, all weighted expressions are valid:

▶ **Theorem 12.** *Let $\mathbb{K}$ be a rationally additive semiring, and M be a pre-rational monoid. Every $\mathbb{K}$-expression W over M is valid,* i.e. *the semantics $[\![W]\!](m)$ exists for all $m \in M$.*

Notice that this theorem relies on both its assumptions on M and $\mathbb{K}$:

- If M is not pre-rational, then the expressions may not be valid. For instance, consider M to be the free group generated by a single element a (with a^{-1} its inverse in the free group), and $\mathbb{K}$ be the semiring of rational languages over the alphabet $\{A, B\}$. Then, the expression $(a \cdot \{A\} + a^{-1} \cdot \{B\})^*$ would associate with the element ε_M of M the language of words over $\{A, B\}$ having as many A's than B's, which is not rational, and thus not a member of $\mathbb{K}$.
- If $\mathbb{K}$ is not rationally additive, then the expressions may not be valid. For instance, considering the semiring $(\mathbb{Q}, +, \times, 0, 1)$, and the (pre-rational) trivial monoid $\{\varepsilon_M\}$, the expression $W = (-1)^*$ gives as a semantics $[\![W]\!](\varepsilon_M) = \sum_{n \in \mathbb{N}} (-1)^n$ that is the archetypal diverging series in $\mathbb{Q}$.

The rest of this section is devoted to the proof of this theorem. This proof is split into two parts. We first show that the validity of a weighted expression obtained by the rewriting of "letters" in an *unambiguous* Kleene expression is equivalent to the existence of sums resembling the ones of Lemma 8. We then explain how to generate such an unambiguous Kleene expression from a weighted expression W, and apply the previous result to show the validity of W.

More formally, a Kleene expression E (over a monoid A^*) is called unambiguous if for all its subexpressions E':

- if $E' = F + G$, then $\mathcal{L}(F) \cap \mathcal{L}(G) = \emptyset$;
- if $E' = F \cdot G$, then for all $w \in A^*$, there exists at most one pair $(w_1, w_2) \in \mathcal{L}(F) \times \mathcal{L}(G)$ such that $w_1 w_2 = w$;
- if $E' = F^*$, then for all $w \in A^*$, there exists at most one natural number n, and one sequence $(w_1, w_2, \ldots, w_n) \in (\mathcal{L}(F))^n$ such that $w_1 w_2 \cdots w_n = w$.

As a direct corollary, for every semiring $\mathbb{K}$,

- if $E + F$ is unambiguous, then $\chi_{E+F} = \chi_E + \chi_F$;
- if $E \cdot F$ is unambiguous, then $\chi_{E \cdot F}(w) = \sum_{uv=w} \chi_E(u)\chi_F(v)$;

- if E^* is unambiguous, then $\chi_{E^*} = \sum_{n=0}^{\infty} \chi_{E^n}$, this infinite sum having indeed a finite support and being thus meaningful in any semiring (and formally existing in a rationally additive semiring).

Given two morphisms $\lambda\colon A^* \to M$ and $\pi\colon A^* \to \mathbb{K}$, we let $E_{\lambda,\pi}$ be the weighted expression obtained from a Kleene expression E by substituting every letter a appearing in E by the expression $\lambda(a)\cdot\pi(a)$, and by replacing Booleans *true* and *false* by elements $1 \in \mathbb{K}$ and $0 \in \mathbb{K}$.

The next lemma aims at linking the validity of $E_{\lambda,\pi}$ with the existence of specific infinite sums. The same result is also fundamental in our later proofs of translations between automata and expressions in the next section.

▶ **Lemma 13.** *Let E be an unambiguous Kleene expression over a free monoid A^*, M be a monoid (not necessarily pre-rational), $\mathbb{K}$ be a rationally additive semiring, $\lambda\colon A^* \to M$ and $\pi\colon A^* \to \mathbb{K}$ be two morphisms. Then, $E_{\lambda,\pi}$ is valid if and only if for all $m \in M$ and all subexpressions F of E, the sum $\sum_{\lambda(w)=m} \pi(w)\chi_F(w)$ exists (where the sum is over all words $w \in A^*$ such that $\lambda(w) = m$). In this case, for all $m \in M$, $[\![E_{\lambda,\pi}]\!](m) = \sum_{\lambda(w)=m} \pi(w)\chi_E(w)$.*

Starting from a weighted expression W, and in order to use Lemma 13 which only applies to unambiguous Kleene expressions, we will modify W to interpret it as an unambiguous Kleene expression. We define its *indexed expression* $I(W)$ as the Kleene expression over an alphabet being a finite subset of $(\mathbb{K} \cup M) \times \mathbb{N}$, obtained by replacing each of its atomic subexpression $\ell \in \mathbb{K} \cup M$ by a letter $(\ell, i) \in (\mathbb{K} \cup M) \times \mathbb{N}$ where i is a unique index (starting from 0 for the leftmost one) associated with each atomic subexpression. For instance, with the weighted expression $W = (2\cdot a + 3\cdot b)^* \cdot (a + 5\cdot b + 3)$, one associates the indexed expression $I(W) = ((2,0)\cdot(a,1)+(3,2)\cdot(b,3))^*\cdot((a,4)+(5,5)\cdot(b,6)+(3,7))$. From the indexed expression, one can recover the initial expression, by forgetting about the index. Formally, we let λ be the morphism from $((\mathbb{K}\cup M)\times\mathbb{N})^*$ to M such that $\lambda(x,n) = x$ if $x \in M$ and ε_M otherwise, and π be the morphism from $((\mathbb{K}\cup M)\times\mathbb{N})^*$ to $\mathbb{K}$ such that $\pi(x,n) = x$ if $x \in \mathbb{K}$ and 1 otherwise. For the above example, $I(W)_{\lambda,\pi} = ((\varepsilon_M\cdot 2)\cdot(a\cdot 1)+(\varepsilon_M\cdot 3)\cdot(b\cdot 1))^*\cdot((a\cdot 1)+(\varepsilon_M\cdot 5)\cdot(b\cdot 1)+(\varepsilon_M\cdot 3))$, which is equivalent to W. More generally, we obtain:

▶ **Lemma 14.** *For all weighted expressions W over M, $I(W)_{\lambda,\pi}$ is valid if and only if W is valid. When valid, they have the same semantics.*

We would like to conclude by combining this result with Lemma 13 and by using the pre-rationality of the monoid M, as in Theorem 9. However, $I(W)$ might not be unambiguous as expected, as shown by the example $W = (m^*)^*$, with $m \in M$, that gives rise to the (ambiguous) Kleene expression $I(W) = (((m,0))^*)^*$: indeed, the word $(m,0)(m,0)$ has several possible decompositions in the semantics of $I(W)$. To patch this last issue, we simply incorporate a dummy marker after each Kleene star as follows: from a weighted expression W, $\phi(W)$ is inductively defined by:

- if W is an atomic expression, $\phi(W) = W$;
- if $W = U + V$ then $\phi(W) = \phi(U) + \phi(V)$;
- if $W = U \cdot V$ then $\phi(W) = \phi(U) \cdot \phi(V)$;
- if $W = U^*$ then $\phi(W) = (\phi(U))^* \cdot 1$, with 1 being the neutral element of the semiring $\mathbb{K}$.

We directly obtain:

▶ **Lemma 15.** *Let W be a weighted expression. The Kleene expression $I(\phi(W))$ is unambiguous.*

We are now ready to conclude the proof of Theorem 12, moreover showing that for all weighted expressions W and $m \in M$, $[\![W]\!](m) = \sum_{\lambda(w)=m} \pi(w)\chi_{I(\phi(W))}(w)$. Indeed, operation $\phi(\cdot)$ does not change the semantics of an expression, and therefore, $\phi(W)$ is valid if and only if W is valid, in which case they share the same semantics. Using the result of Lemma 15, we can apply Lemma 14: W is valid if and only if $I(\phi(W))_{\lambda,\pi}$ is valid, in which case they are equivalent. Let $L = \mathcal{L}(I(\phi(W))) \cap \lambda^{-1}(m)$. Since M is pre-rational, L is a rational language, and $\sum_{w\in L} \pi(w)$ exists. Moreover,

$$\begin{aligned}
\sum_{w\in L} \pi(w) &= \sum_{\lambda(w)=m} \pi(w)\chi_{I(\phi(W))}(w) \\
&= [\![I(\phi(W))_{\lambda,\pi}]\!](m) && \text{(by Lemma 13)} \\
&= [\![\phi(W)]\!](m) && \text{(by Lemma 14)} \\
&= [\![W]\!](m) && (W \text{ and } \phi(W) \text{ are equivalent}).
\end{aligned}$$

5 A Kleene-Like Theorem

Our main result is the following Kleene-like theorem, stating the constructive equivalence between expressions and automata over a pre-rational monoid and weighted over a rationally additive semiring.

▶ **Theorem 16.** *Let $\mathbb{K}$ be a rationally additive semiring, and M be a pre-rational monoid. Let $s \in \mathbb{K}\langle\langle M \rangle\rangle$ be a series. Then s is the semantics of some $\mathbb{K}$-automaton over M if and only if it is the semantics of some $\mathbb{K}$-expression over M.*

The rest of this section is devoted to the proof of this theorem, that consists in constructive translations of automata into equivalent expressions, and vice versa.

From Automata to Expressions. The idea is to build a $\mathbb{K}$-expression from an unambiguous expression generating the accepting runs of the automaton. Let $\mathcal{A} = (Q, \Delta, I, F)$ be a $\mathbb{K}$-automaton over M. By applying the result of [5], we build an unambiguous Kleene expression E over Δ^* recognising the language $R_\mathcal{A}$ of the accepting runs of $\mathcal{A}$. By Lemma 13, that we can apply on E since $E_{\lambda_\mathcal{A},\pi_\mathcal{A}}$ is valid (by Theorem 12), we have

$$[\![E_{\lambda_\mathcal{A},\pi_\mathcal{A}}]\!](m) = \sum_{\lambda_\mathcal{A}(w)=m} \pi_\mathcal{A}(w)\chi_E(w) = \sum_{w\in R_\mathcal{A} | \lambda_\mathcal{A}(w)=m} \pi_\mathcal{A}(w) = [\![\mathcal{A}]\!](m).$$

the second equality coming from (1), since $\mathcal{L}(E) = R_\mathcal{A}$.

From Expressions to Automata. We have shown in the previous section how, from a $\mathbb{K}$-expression E over M, we can construct an unambiguous Kleene expression $I(\phi(E))$ and two morphisms λ and π from[4] $(\mathbb{K} \cup M) \times \mathbb{N}$ to respectively M and $\mathbb{K}$, such that $I(\phi(E))_{\lambda,\pi}$ is equivalent to E, and by Theorem 12, $[\![E]\!](m) = \sum_{\lambda(w)=m} \pi(w)\chi_{I(\phi(E))}(w)$. We let $\{0, \dots, n\}$ be the set of indices used in $I(\phi(E))$.

By [5], we can build (for instance, by considering the position automaton) from $I(\phi(E))$ an equivalent unambiguous Boolean automaton $\mathcal{A} = (Q, \Delta, I, F)$ with $\Delta \subseteq Q \times \big((\mathbb{K} \cup M) \times \mathbb{N}\big) \times Q$ its set of transitions labelled by indexed atomic elements appearing in E. Here, unambiguous means as usual that every accepted word in $\mathcal{A}$ is associated with a unique accepting run.

[4] As before, in fact, we work with a finite subset of this set.

From $\mathcal{A}$, we build a $\mathbb{K}$-automaton $\mathcal{B} = (Q \times \{0, \dots, n\}, \Delta', I \times \{0\}, F \times \{0, \dots, n\})$ over M with transitions defined as follows: for all transitions $(p, (m, i), q) \in \Delta$, with $m \in M$, we add the transition $((p, j), m, 1, (q, i)) \in \Delta'$, and for all transitions $(p, (k, i), q) \in \Delta$, with $k \in \mathbb{K}$, we add the transition $((p, j), \varepsilon_M, k, (q, i)) \in \Delta'$. The transfer of indices from letters to states enables us to obtain a bijection f from accepted words of $\mathcal{A}$ to accepting runs of $\mathcal{B}$. Moreover, this bijection preserves the labels and weights, meaning that for all $u = (x_0, i_0) \cdots (x_m, i_m)$ accepted by $\mathcal{A}$, we have $\lambda(u) = \lambda_{\mathcal{B}}(f(u))$, and $\pi(u) = \pi_{\mathcal{B}}(f(u))$. Therefore, by applying the change of variable $w = f(u)$, we obtain

$$[\![\mathcal{B}]\!](m) = \sum_{w \in R_{\mathcal{B}} \cap \lambda_{\mathcal{B}}^{-1}(m)} \pi_{\mathcal{B}}(w) = \sum_{u \in \mathcal{L}(I(\phi(E))) \cap \lambda^{-1}(m)} \pi(u) = \sum_{\lambda(u)=m} \pi(u) \chi_{I(\phi(E))}(u) = [\![E]\!](m).$$

6 Dealing with Ambiguity

We have already encountered ambiguity in the context of the Boolean semiring and free monoids. We now study this notion for weighted expressions and automata. To do so, we use the rationally additive semiring $(\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}, +, \times, 0, 1)$ where all infinite sums exist: in particular, the sum over a family containing an infinite number of non-zero values is ∞, and otherwise the sum is equal to the finite sum over the support of the family. We call this semiring the *counting semiring*.

▶ **Definition 17.** *Given a $\mathbb{K}$-expression W over the monoid M, the* ambiguity $\mathrm{amb}(W, m)$ of W at m *is a value in $\mathbb{N}_\infty$ defined inductively over W as follows:*

- *for $W = n \in M$, $\mathrm{amb}(n, m) = 1$ if $n = m$, and 0 otherwise;*
- *for $W = k \in \mathbb{K}$, $\mathrm{amb}(k, m) = 1$ if $m = \varepsilon_M$, and 0 otherwise;*
- *for $W = U + V$, $\mathrm{amb}(U + V, m) = \mathrm{amb}(U, m) + \mathrm{amb}(V, m)$;*
- *for $W = U \cdot V$, $\mathrm{amb}(U \cdot V, m) = \sum_{m_1 m_2 = m} \mathrm{amb}(U, m_1) \times \mathrm{amb}(V, m_2)$;*
- *for $W = U^*$, $\mathrm{amb}(U^*, m) = \sum_{n \in \mathbb{N}} \mathrm{amb}(U^n, m)$.*

An expression is called unambiguous *if its ambiguity at every point is at most 1.*

For instance, the expression $W = 2 \cdot a + 3 \cdot a \cdot a$ over the free monoid $\{a\}^*$ is unambiguous, while W^* has ambiguity 2 at the word $aaa = a \cdot aa = aa \cdot a$.

The attentive reader may have noticed that the ambiguity of W is exactly the semantics of W where every atomic weight of $\mathbb{K}$ is replaced with the unit 1 of $\mathbb{N}_\infty$. Given two rationally additive semirings $\mathbb{K}_1$ and $\mathbb{K}_2$, $\mathbb{K}_1 \times \mathbb{K}_2$ is also a rationally additive semiring with the natural component-wise operations. In particular, given a $\mathbb{K}$-expression W, we can define a $\mathbb{K} \times \mathbb{N}_\infty$-expression W' by replacing every weight $k \in \mathbb{K}$ appearing in W by $(k, 1) \in \mathbb{K} \times \mathbb{N}_\infty$. Then, the ambiguity of W at m is the second component of the weight $[\![W']\!](m)$.

▶ **Definition 18.** *Given a $\mathbb{K}$-automaton $\mathcal{A}$ over the monoid M, the* ambiguity of $\mathcal{A}$ at m *is a value in $\mathbb{N}_\infty$ defined as the number (potentially ∞) of runs with label m. An automaton is called* unambiguous *if its ambiguity at every point is at most* 1.

Just as for expressions, the ambiguity of an automaton may be viewed as the semantics of the automaton where the weights of transitions are replaced by the unit of $\mathbb{N}_\infty$. Given $\mathcal{A}$ over $\mathbb{K}$, we can define $\mathcal{A}'$ by replacing all weights $k \in \mathbb{K}$ of transitions by $(k, 1) \in \mathbb{K} \times \mathbb{N}_\infty$. Then the ambiguity of $\mathcal{A}$ at m is exactly the second component of $[\![\mathcal{A}']\!](m)$. Now we claim:

▶ **Theorem 19.** *Let $\mathbb{K}$ be a rationally additive semiring, M be a pre-rational monoid, $s \in \mathbb{K}\langle\langle M \rangle\rangle$, and $k \in \mathbb{N}$. Then, s is the semantics of a $\mathbb{K}$-automaton over M of ambiguity k if and only if it is the semantics of a $\mathbb{K}$-expression over M of ambiguity k.*

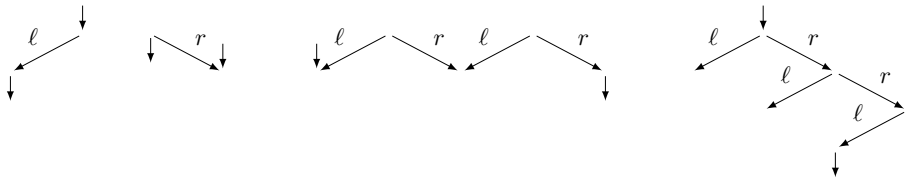


Figure 1 Munn bi-rooted trees of the elements of $\mathcal{I}(\{\ell, r\})$: ℓ, $\bar{r}$, $\bar{\ell}r\bar{\ell}r$, and $(\ell\bar{\ell}r)^2\ell$.

Proof. The procedures of section 5 to go from expressions to automata and back, over a pre-rational monoid M, preserve ambiguity. Indeed, the two constructions used to prove Theorem 16 do not introduce new weights. Thus, starting from a $\mathbb{K}$-expression W, one considers the $\mathbb{K} \times \mathbb{N}_\infty$-expression W' defined above. Transforming W' into an automaton preserves the semantics, and all the transitions have a second component equal to 1. Thus, the second component of the semantics, which is preserved, is exactly the ambiguity of the automaton. Forgetting about the second component, we get the result. Note that converting W to W' is not actually a necessary step to build the automaton, it is simply a mental crutch to make the argument simpler. Symmetrically when going from automata to expressions, the transformation does not introduce new weights and thus preserves ambiguity. ◀

7 Free Inverse Monoids and Applications to Walking Automata

We conclude this article by demonstrating why our model is able to encompass and reason about the usual models of two-way automata and tree-walking automata. To do so, we consider the free inverse monoid, as it was observed by Pécuchet [18] to be linked with this model. Dicky and Janin even gave in [9, Theorem 3.21] the equivalence in the boolean case between two-way automata and regular expressions, using this monoid.

Let A be a finite alphabet, and $\overline{A} = \{\overline{a} \mid a \in A\}$ be a copy of A. We define the function $\dagger\colon (A \cup \overline{A})^* \to (A \cup \overline{A})^*$ inductively as: $\varepsilon^\dagger = \varepsilon$, $(ua)^\dagger = \overline{a}u^\dagger$, and $(u\overline{a})^\dagger = au^\dagger$.

▶ **Definition 20.** *The* free inverse monoid $\mathcal{I}(A)$ *generated by a finite alphabet A is the quotient of $(A \cup \overline{A})^*$ by the following equivalence relations:*

- *"$x^\dagger$ and x are pseudo-inverses": for all $x \in (A \cup \overline{A})^*$, $xx^\dagger x = x$, and $x^\dagger xx^\dagger = x^\dagger$;*
- *"idempotent elements commute": for all $x, y \in (A \cup \overline{A})^*$: $xx^\dagger yy^\dagger = yy^\dagger xx^\dagger$.*

Notice that $xx^\dagger$ are indeed idempotent elements of the free inverse monoid, since $(xx^\dagger)(xx^\dagger) = (xx^\dagger x)x^\dagger = xx^\dagger$.

The elements of this monoid are conveniently represented via tree-like structures, the Munn bi-rooted trees [17]. They are directed graphs, whose underlying undirected graph is a tree, and two special nodes are marked, the *initial* and the *final* one. Examples of elements of the monoid with their Munn tree representation are given in Figure 1. Note that if you see $a \in A$ as the traversal of an edge labelled by a, and $\overline{a}$ its traversal in reverse, an element of $(A \cup \overline{A})^*$ describes a complete walk over the graph of the corresponding element of $\mathcal{I}(A)$.

With this tree representation in mind, we see that every element of $\mathcal{I}(A)$ has finitely many prefixes, since such a prefix is a subtree of x, with the same initial node. Thanks to Lemma 2, we obtain

▶ **Proposition 21.** *The free inverse monoid is pre-rational.*

We can thus apply our results on this pre-rational monoid, for instance by considering expressions. In the Boolean semiring, for example, the expression $(\ell \cdot \bar{\ell} \cdot r)^* \cdot \ell$ describes the language of Munn bi-rooted trees that are "right-combs" (see the rightmost tree of Figure 1),

when considering ℓ to be left children, and r right ones. The initial node is at the top while the final one is the farthest away from it. We can add weights to this expression: in the tropical semiring $(\mathbb{Z} \cup \{-\infty, +\infty\}, \sup, +, -\infty, 0)$, the unambiguous expression $(\ell \cdot \bar{\ell} \cdot r \cdot 1)^* \cdot \ell$ associates with a comb the length of its rightmost branch. More generally, the expression $W = \left[\sum_{a \in A} \left(a \cdot 1 + \bar{a} \cdot (-1)\right)\right]^*$ computes the (signed) length of the path linking the initial and final nodes in any Munn bi-rooted tree over alphabet A: each tree is associated with the difference between the number of positive letters of A and the number of negative letters of $\bar{A}$ of the unique acyclic path linking the initial node to the final node. On the trees of Figure 1, these lengths are respectively 1, -1, 0, 3. They represent the difference of "levels" in-between the initial and final nodes. Each tree is associated with many decompositions in the semantics of the expression W, but all of them have the same weight (and the chosen semiring has an idempotent sum operation).

Two-way Automata. Over an alphabet A, we can consider the free inverse monoid $\mathcal{I}(A \uplus \{\vdash, \dashv\})$, with two fresh symbols $\vdash$ and $\dashv$ that will help us distinguish the leftmost and rightmost letters of the word. To model two-wayness, only certain elements of $\mathcal{I}(A \uplus \{\vdash, \dashv\})$ are of interest, namely elements of $\vdash A^* \dashv$, that have linear Munn bi-rooted trees with the initial node at the leftmost position, and the final node at the rightmost one. The Munn bi-rooted tree representation of such an element is given in Figure 2.

We thus consider weighted automata and expressions over $\mathcal{I}(A)$ with weights in $\mathbb{K}$, a rationally additive semiring, and restrict our attention to words of $\vdash A^* \dashv$. From an automata perspective, this is a way to define the usual model of two-way automata, a forward movement of a two-way automaton being simulated by reading of a letter in A while a backward movement is simulated by reading a letter in $\bar{A}$. Indeed, our model of weighted automata over $\mathcal{I}(A)$ can also be simulated by the usual two-way weighted automata, since non-atomic elements of the monoid can be split into atomic elements. Therefore, in this specific context, Theorem 16 gives a new way to express the semantics of two-way weighted automata (over a rationally additive semiring) by using expressions.

Consider for example the function that maps a word $\vdash w \dashv$ with $w = w_0 \cdots w_{n-1} \in \{a, b\}^*$ to the set of words $\{(w_{n-1} \cdots w_0)^k \mid k \in \mathbb{N}\}$. Considering the semiring of regular languages, a weighted expression describing this function is

$$\left(\vdash \cdot (a+b)^* \cdot \dashv \cdot \overline{\dashv} \cdot (\overline{a} \cdot \{a\} + \overline{b} \cdot \{b\})^* \cdot \overline{\vdash}\right)^* \cdot \vdash \cdot (a+b)^* \cdot \dashv.$$

Notice the last pass over the word that allows one to finish the reading on the rightmost position, i.e. the final node.

Consider the alphabet $A = \{\mathbf{0}, \mathbf{1}\}$. For a word $w \in A^*$, let $w_{|2}$ denote the rational number between 0 and 1 that is written as $0.w$ in binary. Then, consider the following weighted expression with weights in $(\mathbb{Q}_+ \cup \{+\infty\}, +, \times, 0, 1)$:

$$W = \vdash \cdot \left(\mathbf{0} \cdot \frac{1}{2} + \mathbf{1} \cdot \frac{1}{2}\right)^* \cdot \mathbf{1} \cdot \frac{1}{2} \cdot (\mathbf{0} + \mathbf{1})^* \cdot \dashv.$$

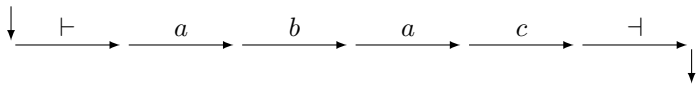


Figure 2 Munn bi-rooted tree of the "word" $\vdash abac \dashv$.

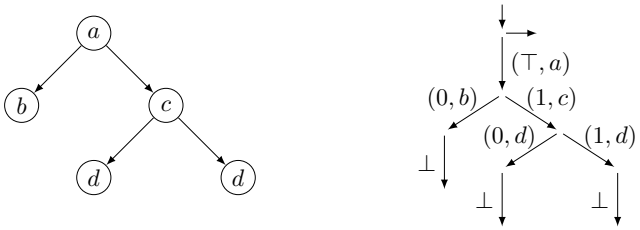


Figure 3 A binary tree, and its encoding in $\mathcal{I}(A')$.

It associates with a word $\vdash w \dashv$ the value $w_{|2}$, since it non-deterministically chooses a position i labelled by $\mathbf{1}$ in w and computes the value $1/2^i$. By considering the expression

$$(W \cdot \overline{\dashv} \cdot (\overline{\mathbf{0}} + \overline{\mathbf{1}})^* \cdot \overline{\vdash})^* \cdot W.$$

that consists in repeating the computation of W any number of times (at least once), with a reset of the word in-between, we associate with a word $\vdash w \dashv$ the value $\sum_{n=1}^{\infty} w_{|2}^n = w_{|2}/(1 - w_{|2})$.

Tree-Walking Automata. Another model captured by our approach is the one of tree-walking automata. These are automata whose head moves on the nodes of a rooted tree of a bounded arity m. As for words before, we can encode such trees labelled with a finite alphabet A by elements of $\mathcal{I}(A')$ with an extended alphabet $A' = (\{0, \dots, m-1\} \cup \{\top\}) \times A \cup \{\bot\}$. In elements of $\mathcal{I}(A')$, nodes contain no information, only edges do. The idea is thus to simulate the root of a tree labelled with a by a single node labelled with $(\top, a)$; the i-th child of a node, labelled with $a \in A$, will be simulated with a node of label (i, a); finally, under each leaf of the tree, we add a node labelled with $\bot$. The root of the tree will be both the initial and the final node of the encoding, simulating a tradition of tree-walking automata to start and end in the root of the tree (without loss of generality).

As an example, consider the binary tree on the left of Figure 3. It is modelled by the following element of $\mathcal{I}(A')$, obtained from the Munn bi-rooted tree represented on the right by a depth-first search: $(\top, a)(0, b)\bot\overline{\bot}\,\overline{(0, b)}\,(1, c)(0, d)\bot\overline{\bot}\,\overline{(0, d)}\,(1, d)\bot\overline{\bot}\,\overline{(1, d)}\,\overline{(1, c)}\,\overline{(\top, a)}$.

When restricting the semantics of weighted automata and expressions to elements of $\mathcal{I}(A')$ that are encoding of trees, Theorem 16 gives an interesting model of weighted expressions equivalent to weighted tree-walking automata over rationally additive semirings.

The depth-first search of a tree is describable by an unambiguous weighted expression (and thus also an unambiguous weighted automaton): letting (i, A) denote $\sum_{a \in A}(i, a)$, and restricting ourselves to trees with nodes of arity 0 or 2 to simplify the writing, we let

$$W_0 = (0, A)^* \cdot \bot\,, \quad W_1 = \overline{\bot} \cdot \overline{(1, A)}^*\,, \quad \text{and} \quad W_{\text{succ}} = W_1 \cdot \overline{(0, A)} \cdot (1, A) \cdot W_0.$$

The weighted expression W_0 finds the leftmost leaf; W_1 returns to the root from the rightmost leaf; and W_{succ} goes from a leaf to the next one in the depth-first search. Then, the depth-first search is implemented by the weighted expression $(\top, A) \cdot W_0 \cdot W_{\text{succ}}^* \cdot W_1 \cdot \overline{(\top, A)}$.

By Theorem 19, there exists an equivalent non ambiguous automaton, that thus visits the whole tree. Since it is possible to *reset* the tree, going back to the root, in a non ambiguous fashion, we can remove the requirement for the automata and the expressions to visit the whole tree while starting and ending at the root. This allows for more freedom in the models.

Taking advantage of this relaxation, it is possible to count the maximal number of occurrences of a letter a in branches of the tree, starting at the root of the tree, non-deterministically going down the chosen branch, and ending at the bottom: using the rationally additive semiring $(\mathbb{N} \cup \{-\infty, +\infty\}, \sup, +, -\infty, 0)$,

$$\big((\top, a) \cdot 1 + (\top, A \setminus \{a\})\big) \cdot \big((0, a) \cdot 1 + (0, A \setminus \{a\}) + (1, a) \cdot 1 + (1, A \setminus \{a\})\big)^* \cdot \bot.$$

8 Conclusion

We have given an application of our result to tree-walking automata. A natural extension consists in investigating other kinds of structure like Mazurkiewicz traces or grids.

Our approach *is* able to capture tree-walking automata, however it is intrinsically more of a tree-*generating* automaton model. Over trees it does not make a huge difference but it does if we try to extend this approach to more general graph-walking automata models. A natural way to define weighted automata over graphs is to take the sum of the weights of all paths over a given graph (in a sense already explored in [16], but limiting itself to non-looping runs). This means that a given path in the automaton can be a run in different graphs, which is not compatible with our approach of generating monoid elements.

One possible research direction would be to consider so-called SD-expressions introduced by Schützenberger (see [10]). These expressions were shown to coincide with star-free expressions with the advantage of not using the complement (instead restricting the languages over which the Kleene star can be applied, namely to prefix codes with bounded synchronisation delay) which means it can be applied to the quantitative setting. Indeed, in [7], the authors extended the result to transducers and showed that these expressions correspond to aperiodic transducers. These expressions are naturally adapted to the unambiguous setting (maybe this restriction can be overcome) but it would be interesting to study their expressive power in the context of pre-rational monoids.

A final direction would be to use logics instead of expressions, to describe in a less operational way the behaviour of weighted automata over monoids. Promising results have already been obtained in specific contexts, like non-looping automata walking (with pebbles) on words, trees or graphs [4], but a cohesive point of view via monoids is still lacking.

References

1 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *CSL-LICS '14*, pages 9:1–9:10. ACM, 2014.

2 Nicolas Baudru, Louis-Marie Dando, Nathan Lhote, Benjamin Monmege, Pierre-Alain Reynier, and Jean-Marc Talbot. Weighted automata and expressions over pre-rational monoids, October 2021. `arXiv:2110.12395`.

3 Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. *International Journal of Foundations of Computer Science*, 31(6):843–873, 2020.

4 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Logical characterization of weighted pebble walking automata. In Thomas A. Henzinger and Dale Miller, editors, *Proceedings of the 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)*, Vienna, Austria, July 2014. ACM. `doi:10.1145/2603088.2603118`.

5 R. Book, S. Even, Sheila A. Greibach, and G. Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, 20(2):149–153, 1971.

6 Christian Choffrut and Bruno Guillon. An algebraic characterization of unary two-way transducers. In *MFCS 2014*, volume 8634 of *LNCS*, pages 196–207. Springer, 2014.

7 Luc Dartois, Paul Gastin, and Shankara Narayanan Krishna. SD-regular transducer expressions for aperiodic transformations. *CoRR*, 2021. arXiv:2101.07130.

8 Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular transducer expressions for regular transformations. In *LICS 2018*, pages 315–324. ACM, 2018.

9 Anne Dicky and David Janin. Two-way automata and regular languages of overlapping tiles. *Fundamenta Informaticae*, 142:1–33, 2015.

10 Volker Diekert and Tobias Walter. Characterizing classes of regular languages using prefix codes of bounded synchronization delay. In *ICALP 2016*, volume 55 of *LIPIcs*, pages 129:1–129:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

11 Zoltán Ésik and Werner Kuich. Rationally additive semirings. *Journal of Universal Computer Science*, 8(2):173–183, 2002.

12 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *SIGLOG News*, 3(3):4–19, 2016.

13 Bruno Guillon. *Two-wayness: Automata and Transducers*. PhD thesis, Université Paris-Diderot and Universita di Milano, 2016.

14 Sylvain Lombardy. Two-way representations and weighted automata. *RAIRO - Theor. Inf. and Appl.*, 50(4):331–350, 2016. doi:10.1051/ita/2016026.

15 Sylvain Lombardy and Jacques Sakarovitch. The validity of weighted automata. *Internat. J. Algebra Comput.*, 23:863–913, 2013.

16 Benjamin Monmege. *Specification and Verification of Quantitative Properties: Expressions, Logics, and Automata*. Phd, ENS Cachan, France, 2013.

17 Walter D Munn. Free inverse semigroups. *Proceedings of the London Mathematical Society*, 3(3):385–404, 1974.

18 Jean-Pierre Pécuchet. Automates boustrophedon, semi-groupe de birget et monoide inversif libre. *RAIRO Theor. Inf. and Appl.*, 19(1):71–100, 1985.

19 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

20 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

Optimal Strategies in Concurrent Reachability Games

Benjamin Bordais
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Patricia Bouyer
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Stéphane Le Roux
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Abstract

We study two-player reachability games on finite graphs. At each state the interaction between the players is concurrent and there is a stochastic Nature. Players also play stochastically. The literature tells us that 1) Player B, who wants to avoid the target state, has a positional strategy that maximizes the probability to win (uniformly from every state) and 2) from every state, for every $\varepsilon > 0$, Player A has a strategy that maximizes up to ε the probability to win. Our work is two-fold.

First, we present a double-fixed-point procedure that says from which state Player A has a strategy that maximizes (exactly) the probability to win. This is computable if Nature's probability distributions are rational. We call these states *maximizable*. Moreover, we show that for every $\varepsilon > 0$, Player A has a positional strategy that maximizes the probability to win, exactly from maximizable states and up to ε from sub-maximizable states.

Second, we consider three-state games with one main state, one target, and one bin. We characterize the *local interactions* at the main state that guarantee the existence of an optimal Player A strategy. In this case there is a positional one. It turns out that in many-state games, these local interactions also guarantee the existence of a uniform optimal Player A strategy. In a way, these games are well-behaved by design of their elementary bricks, the local interactions. It is decidable whether a local interaction has this desirable property.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory

Keywords and phrases Concurrent reachability games, Game forms, Optimal strategies

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.7

Related Version *Full Version*: https://arxiv.org/abs/2110.14724

1 Introduction

Stochastic concurrent games. Games on graphs are an intensively studied mathematical tool, with wide applicability in verification and in particular for the controller synthesis problem, see for instance [16, 1]. We consider two-player stochastic concurrent games played on finite graphs. For simplicity (but this is with no restriction), such a game is played over a finite bipartite graph called an arena: some states belong to Nature while others belong to the players. Nature is stochastic, and therefore assigns a probabilistic distribution over the players' states. In each players' state, a local interaction between the two players (called Player A and Player B) happens, specified by a two-dimensional table. Such an interaction is resolved as follows: Player A selects a probability distribution over the rows of the table while Player B selects a probability distribution over the columns of the table; this results into a distribution over the cells of the table, each one pointing to a Nature state of the graph. An example of game arena is given in Figure 1: circle states are players' while square states are Nature's; note that dashed arrows assign only probability 1 to a next state in this example (but in general could give probabilities to several states).

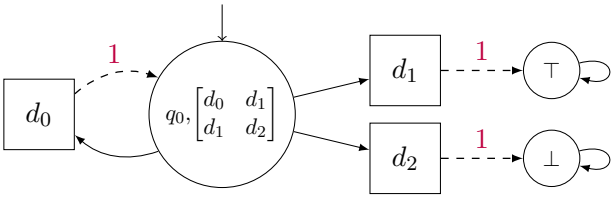


Figure 1 The game starts in q_0 with two actions available for each player. Player A wins if the state $\top$ is reached.

$$\begin{bmatrix} x & y \\ y & z \end{bmatrix}$$

Figure 2 The local interaction at q_0 up to a renaming of the outcomes.

Globally, the game proceeds as follows: starting at an initial state q_0, the two players play the local interaction of the current state, and the joint choice determines (stochastically) the next Nature state of the game, itself moving randomly to players' states; the game then proceeds subsequently from the new players' state. The way players make choices is given by strategies, which, given the sequence of states visited so far (the so-called history), assign local strategies for the local interaction of the state the game is in. For application in controller synthesis, strategies will correspond to controllers, hence it is desirable to have strategies simple to implement. We will be in particular interested in strategies which are *positional*, that is, strategies which only depend on the current state of the game, not on the whole history. When each player has fixed a strategy (say $\mathsf{s_A}$ for Player A and $\mathsf{s_B}$ for Player B), this defines a probability distribution $\mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}$ over infinite sequences of states of the game. The objectives of the two players are opposite (we assume a zero-sum setting): together with the game, a measurable set W of infinite sequences of states is fixed; the objective of Player A is then to maximize the probability of W while the objective of Player B is to minimize this probability.

Back to the example of Figure 1, assume Player A (resp. B) plays the first row (resp. column) with probability p_A (resp. p_B), then the probability to move to $\top$ is $p_\mathsf{A} + p_\mathsf{B} - 2p_\mathsf{A}p_\mathsf{B}$. If Player A repeatedly plays the same strategy at q_0 with $p_\mathsf{A} < 1$, then the probability to reach $\top$ will lie between p_A and 1, depending on Player B; however, if she plays $p_\mathsf{A} = 1$, then by playing $p_\mathsf{B} = 1$, Player B enforces staying in q_0, hence reaching $\top$ with probability 0.

Values and (almost-)optimal strategies. As mentioned above, Player A wants to maximize the probability of W, while Player B wants to minimize this probability. Formally, given a strategy $\mathsf{s_A}$ for Player A, its value is measured by $\inf_{\mathsf{s_B}} \mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(W)$, and Player A wants to maximize that value. Dually, given a strategy $\mathsf{s_B}$ for Player B, its value is measured by $\sup_{\mathsf{s_A}} \mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(W)$, and Player B wants to minimize that value. Following Martin's determinacy theorem for Blackwell games [13], it actually holds that when W is Borel, then the game has a *value* given by

$$\chi_{q_0} = \sup_{\mathsf{s_A}} \inf_{\mathsf{s_B}} \mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(W) = \inf_{\mathsf{s_B}} \sup_{\mathsf{s_A}} \mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(W)$$

While this ensures the existence of almost-optimal strategies (that is, ε-optimal strategies for every $\varepsilon > 0$) for both players, it says nothing about the existence of optimal strategies, which are strategies achieving χ_{q_0}. In general, as already mentioned in [8], optimal strategies may not exist. Indeed assuming a reachability objective with target $\top$, the game in Figure 1 is such that $\chi_{q_0} = 1$, however Player A can only achieve $1 - \varepsilon$ for every $\varepsilon > 0$ by playing repeatedly at q_0 the first row of the table with probability $1 - \varepsilon$ and the second row with probability ε, but Player A cannot achieve 1.

Our setting. In this paper we focus on reachability games, that is, W is a reachability condition. They are a special case of recursive games (where targets are assigned payoffs), as studied in [8]. As such, they enjoy several nice properties: (i) Player A has positional almost-optimal strategies; (ii) Player B has positional optimal strategies [7]. These properties are specific to reachability games (or slight generalizations thereof), and this is for instance not the case of Büchi games, see [7, Thm. 2].

Our goal is to study *maximizable* and *sub-maximizable* states in (reachability) games: maximizable (resp. sub-maximizable) states are states from which optimal strategies exist (resp. no optimal strategies exist). Our contributions are then mostly twofolds:

1. We characterize via a double-fixed-point procedure maximizable and sub-maximizable states. This characterization cautiously analyzes when and why no optimal strategies will exist. Back to the example of Figure 1, we realize that no optimal strategy exists since at the limit of ε-optimal strategies, i.e. when Player A plays the first row almost-surely, Player B can enforce cycling back to q_0, hence disabling state $\top$.This simple analysis close to the target has to be propagated carefully in the game, in which some strategies which are designated as risky (since they ultimately lead to such a situation) have to be avoided.

   As a byproduct of our construction, we have Theorem 28, which establishes that one can build almost-optimal positional strategies, which are actually optimal where they can be. This refines the result of [8] which did not ensure optimality where it could.

   A consequence of that construction is that maximizable and sub-maximizable states can be computed under slight assumptions, and that witness positional strategies can be computed as well. For these results we rely on Tarski's decidability result of the theory of the reals [15].

   We also show that our result cannot be extended to games with countably many states by exhibiting such a game in which an optimal strategy exists, but there is no optimal positional strategy.

2. Local interactions played by the players are abstracted into game forms, where cells of the matrix are now seen as variables (some of them being equal). For instance, the game form associated with state q_0 in the running example has three outcomes: x, y and z, and it is given in Figure 2. Game forms can be seen as elementary bricks that can be used to build games on graphs. We can embed such a brick into various three-states games with one main state, one target, and one bin (as is done in Figure 1 for the interaction of Figure 2). We characterize the *local interactions* at the main state that guarantee the existence of an optimal Player A strategy. In this case there is a positional one. It turns out that in many-state games, these local interactions also guarantee the existence of a uniform optimal Player A strategy. In a way, these games are well-behaved by design of their elementary bricks, the local interactions. It is decidable whether a local interaction has this desirable property.

   Importantly we exhibit a simple condition on game forms which ensures the above: determined game forms as studied in [2] do satisfy the condition. The latter game forms generalize turn-based local interactions (where each players' state is controlled by a unique player – that is, the matrix defining the local interaction has a single row or a single column). We therefore recover the fact that stochastic turn-based reachability games admit optimal positional strategies, which was shown in [14, 4, 19].

Additional details and complete proofs are available in the arXiv version of this paper [3].

Related work. In [6], the authors characterize using fixed points as well states with value 1: sure-winning states (all generated plays satisfy the reachability condition – as if no probabilities were involved), almost-sure winning states (that is, maximizable states with value 1) and limit-sure winning states (that is, sub-maximizable states with value 1). Our work generalizes this result with states with arbitrary values.

There are many works dedicated to the study of stochastic turn-based games. These games enjoy more properties. Indeed, in parity stochastic turn-based games, Player A always has an optimal pure positional strategy [14, 4, 19]. These results do not extend in general to infinite (turn-based) arenas (even when they are finitely-branching): optimal strategies may not exist, and when they exist, they may require infinite memory [12].

2 Preliminaries

Consider a non-empty set Q. The *support* $\mathsf{Supp}(\mu)$ of a function $\mu : Q \to [0,1]$ corresponds to set of non-0s of the function: $\mathsf{Supp}(\mu) = \{q \in Q \mid \mu(q) \in]0,1]\}$. A *discrete probabilistic distribution* over a non-empty set Q is a function $\mu : Q \to [0,1]$ such that its support $\mathsf{Supp}(\mu)$ is countable and $\sum_{x \in Q} \mu(x) = 1$. The set of all distributions over the set Q is denoted $\mathcal{D}(Q)$. We also consider the product order on vectors $\preceq : \mathbb{R}^n \times \mathbb{R}^n$ defined for any $n \in \mathbb{N}$ by, for all $v, v' \in \mathbb{R}^n$, we have $v \preceq v' \Leftrightarrow \forall i \in [\![1,n]\!],\ v(i) \leq v'(i)$. For $v \in \mathbb{R}^n$ and $x \in \mathbb{R}$, the notation $v + x$ refers to the vector $v' \in \mathbb{R}^n$ such that, for all $i \in [\![1,n]\!]$, we have $v'(i) = v(i) + x$.

3 Game Forms

We recall the definition of game forms which informally are 2-dim. tables with variables.

▶ **Definition 1** (Game form and game in normal form). *A* game form *is a tuple* $\mathcal{F} = \langle \mathsf{St_A}, \mathsf{St_B}, \mathsf{O}, \varrho \rangle$ *where* $\mathsf{St_A}$ *(resp.* $\mathsf{St_B}$*) is the non-empty set of (pure) strategies available to Player* A *(resp.* B*),* O *is a non-empty set of possible outcomes, and* $\varrho : \mathsf{St_A} \times \mathsf{St_B} \to \mathsf{O}$ *is a function that associates an outcome to each pair of strategies. When the set of outcomes* O *is equal to* $[0,1]$*, we say that* $\mathcal{F}$ *is a* game in normal form*. For a valuation* $v \in [0,1]^{\mathsf{O}}$ *of the outcomes, the notation* $\mathcal{F}^v$ *refers to the game in normal form* $\langle \mathsf{St_A}, \mathsf{St_B}, [0,1], v \circ \varrho \rangle$*. A game form* $\mathcal{F} = \langle \mathsf{St_A}, \mathsf{St_B}, \mathsf{O}, \varrho \rangle$ *is* finite *if the set of pure strategies* $\mathsf{St_A} \cup \mathsf{St_B}$ *is finite.*

In the following, the game form $\mathcal{F}$ will always refer to the tuple $\langle \mathsf{St_A}, \mathsf{St_B}, \mathsf{O}, \varrho \rangle$ unless otherwise stated. Furthermore, we will be interested in valuations of the outcomes in the interval $[0,1]$. Informally, Player A (the rows) tries to maximize the outcome, whereas Player B (the columns) tries to minimize it.

▶ **Definition 2** (Outcome of a game in normal form). *Consider a game in normal form* $\mathcal{F} = \langle \mathsf{St_A}, \mathsf{St_B}, [0,1], \varrho \rangle$*. The set* $\mathcal{D}(\mathsf{St_A})$ *corresponds to the set of mixed strategies available to Player* A*, and analogously for Player* B*. For a pair of mixed strategies* $(\sigma_\mathsf{A}, \sigma_\mathsf{B}) \in \mathcal{D}(\mathsf{St_A}) \times \mathcal{D}(\mathsf{St_B})$*, the outcome* $\mathsf{out}_\mathcal{F}(\sigma_\mathsf{A}, \sigma_\mathsf{B})$ *in* $\mathcal{F}$ *of the strategies* $(\sigma_\mathsf{A}, \sigma_\mathsf{B})$ *is defined as:* $\mathsf{out}_\mathcal{F}(\sigma_\mathsf{A}, \sigma_\mathsf{B}) := \sum_{a \in \mathsf{St_A}} \sum_{b \in \mathsf{St_B}} \sigma_\mathsf{A}(a) \cdot \sigma_\mathsf{B}(b) \cdot \varrho(a,b) \in [0,1]$.

The definition of the value of a game in normal form follows:

▶ **Definition 3** (Value of a game in normal form and optimal strategies). *Consider a game in normal form* $\mathcal{F} = \langle \mathsf{St_A}, \mathsf{St_B}, [0,1], \varrho \rangle$ *and a strategy* $\sigma_\mathsf{A} \in \mathcal{D}(\mathsf{St_A})$ *for Player* A*. The* value *of strategy* σ_A*, denoted* $\mathsf{val}_\mathcal{F}(\sigma_\mathsf{A})$ *is equal to:* $\mathsf{val}_\mathcal{F}(\sigma_\mathsf{A}) := \inf_{\sigma_\mathsf{B} \in \mathcal{D}(\mathsf{St_B})} \mathsf{out}_\mathcal{F}(\sigma_\mathsf{A}, \sigma_\mathsf{B})$*, and analogously for Player* B*, with a* sup *instead of an* inf*. When* $\sup_{\sigma_\mathsf{A} \in \mathcal{D}(\mathsf{St_A})} \mathsf{val}_\mathcal{F}(\sigma_\mathsf{A}) = \inf_{\sigma_\mathsf{B} \in \mathcal{D}(\mathsf{St_B})} \mathsf{val}_\mathcal{F}(\sigma_\mathsf{B})$*, it defines the* value *of the game* $\mathcal{F}$*, denoted* $\mathsf{val}_\mathcal{F}$*.*

Note that von Neuman's minimax theorem [18] ensures it does as soon as the game $\mathcal{F}$ is finite. A strategy $\sigma_{\mathsf{A}} \in \mathcal{D}(\mathsf{St}_{\mathsf{A}})$ ensuring $\mathsf{val}_{\mathcal{F}} = \mathsf{val}_{\mathcal{F}}(\sigma_{\mathsf{A}})$ is called optimal. *The set of all optimal strategies for Player* A *is denoted $\mathsf{Opt}_{\mathsf{A}}(\mathcal{F}) \subseteq \mathcal{D}(\mathsf{St}_{\mathsf{A}})$, and analogously for Player* B. *Von Neuman's minimax theorem ensures the existence of optimal strategies (for both players).*

As it will be useful in Section 7, we define a least fixed point operator in a game form given a partial valuation of the outcomes.

▶ **Definition 4** (Total valuation induced by a partial valuation). *For a game form $\mathcal{F}$ and a partial valuation $\alpha : \mathsf{O} \setminus E \to [0,1]$ for some $E \subseteq \mathsf{O}$, we define the map $f_\alpha^{\mathcal{F}} : [0,1] \to [0,1]$ by, for all $y \in [0,1]$: $f_\alpha^{\mathcal{F}}(y) := \mathsf{val}_{\mathcal{F}^{\alpha[y]}}$ where $\alpha[y] : \mathsf{O} \to [0,1]$ is such that $\alpha[y][E] = \{y\}$ and $\alpha[y]|_{\mathsf{O}\setminus E} = \alpha$. The map f_α has a least fixed point (by monotonocity), denoted $v_\alpha \in [0,1]$. The valuation $\tilde{\alpha} \in [0,1]^{\mathsf{O}}$ induced by the partial valuation α is then equal to $\tilde{\alpha} = \alpha[v_\alpha]$.*

4 Concurrent stochastic games

In this section, we define the formalism we use throughout this paper for concurrent graph games, strategies and values.

▶ **Definition 5** (Stochastic concurrent games). *A finite* stochastic concurrent arena *$\mathcal{C}$ is a tuple $\langle A, B, Q, \mathsf{D}, \delta, \mathsf{dist}\rangle$ where A (resp. B) is the non-empty finite set of actions of Player* A *(resp.* B*), Q is the non-empty finite set of states,* D *is the non-empty set of Nature states, $\delta : Q \times A \times B \to \mathsf{D}$ is the transition function, $\mathsf{dist} : \mathsf{D} \to \mathcal{D}(Q)$ is the distribution function. A* concurrent reachability game *is a pair $\langle \mathcal{C}, \top\rangle$ where $\top \in Q$ is a target state (for Player* A*). It is supposed to be a self-looping sink: for all $a \in A$ and $b \in B$, we have $\mathsf{Supp}(\delta(\top, a, b)) = \{\top\}$.*

In the following, the arena $\mathcal{C}$ will always refer to the tuple $\langle A, B, Q, \mathsf{D}, \delta, \mathsf{dist}\rangle$ unless otherwise stated, and $\top$ to the target in the game $\langle \mathcal{C}, \top\rangle$, that we assume fixed in the rest of the definitions. Let us now consider a crucial tool in our study: the notion of local interaction. These are game forms induced by the transition function δ in states of the game.

▶ **Definition 6** (Local interaction). *The* local interaction *at state $q \in Q$ is the game form $\mathcal{F}_q := \langle A, B, \mathsf{D}, \delta(q, \cdot, \cdot)\rangle$. That is, the strategies available for Player* A *(resp.* B*) are the actions in A (resp. B) and the outcomes are the Nature states.*

Local interactions also allow us to define the probability transition to go from one state to another, given two local strategies.

▶ **Definition 7** (Probability transition). *Consider a state $q \in Q$ and two local strategies $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}) \in \mathcal{D}(A) \times \mathcal{D}(B)$ in the game form $\mathcal{F}_q$. Let $q' \in Q$. The probability $p^{q,q'}(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}})$ to go from q to q' if the players opt for strategies σ_{A} and σ_{B} is equal to the outcome of the game form $\mathcal{F}_q$ with the value of a Nature state $d \in \mathsf{D}$ equal to the probability to go from d to q', i.e. it is given by the valuation $\mathsf{dist}(\cdot)(q') \in [0,1]^{\mathsf{D}}$. That is: $p^{q,q'}(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}) := \mathsf{out}_{\mathcal{F}_q^{\mathsf{dist}(\cdot)(q')}}(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}})$.*

Let us now look at the strategies we consider in such concurrent games.

▶ **Definition 8** (Strategies). *A Player* A *strategy is a map $\mathsf{s}_{\mathsf{A}} : Q^+ \to \mathcal{D}(A)$. It is said to be* positional *if, for all $\pi = \rho \cdot q \in Q^+$, we have $\mathsf{s}_{\mathsf{A}}(\pi) = \mathsf{s}_{\mathsf{A}}(q)$: the strategy only depends on the current state. We denote by $\mathsf{S}_{\mathcal{C}}^{\mathsf{A}}$ and $\mathsf{PS}_{\mathcal{C}}^{\mathsf{A}}$ the set of all strategies and positional strategies respectively in arena $\mathcal{C}$ for Player* A*. The definitions are analogous for Player* B.

A pair of strategies then induces a probability measure over paths.

▶ **Definition 9** (Probability measure of paths given two strategies). *For a pair of strategies* $(\mathsf{s_A}, \mathsf{s_B}) \in \mathsf{S}^{\mathsf{A}}_{\mathcal{C}} \times \mathsf{S}^{\mathsf{B}}_{\mathcal{C}}$*, we denote by* $\mathsf{s}^{\pi}_{\mathsf{A}} : Q^+ \to \mathcal{D}(A)$ *the Player* A *residual strategy after* $\pi \in Q^+$ *is seen: for all* $\pi' \in Q^+$*,* $\mathsf{s}^{\pi}_{\mathsf{A}}(\pi') = \mathsf{s_A}(\pi \cdot \pi')$*. The residual strategy* $\mathsf{s}^{\pi}_{\mathsf{B}}$ *is defined analogously. Then, the probability of occurrence of a finite path* $\pi \in Q^+$ *is defined inductively. For all starting states* $q_0 \in Q$*, for all* $q \cdot \pi \in Q^+$*, if* $q \neq q_0$*, we set* $\mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(q) := 0$*. Furthermore,* $\mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(q_0) := 1$ *and for all* $q \cdot \pi \in Q^+$*, we set:*

$$\mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(q_0 \cdot q \cdot \pi) := p^{q_0,q}(\mathsf{s_A}(q_0), \mathsf{s_B}(q_0)) \cdot \mathbb{P}^{q}_{\mathsf{s}^{q_0}_{\mathsf{A}},\mathsf{s}^{q_0}_{\mathsf{B}}}(q \cdot \pi)$$

A probability measure $\mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}$ *is thus defined over the* σ*-algebra generated by cylinders (which are continuations of finite paths). Standardly (see e.g. [17]), infinite sequences of states visiting some subset* $Q' \subseteq Q$ *is measurable, and we note* $\mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(Q')$ *(resp.* $\mathbb{P}^{q_0}_{\mathsf{s_A},\mathsf{s_B}}(n, Q')$*) the probability to reach* Q' *(resp. in at most* n *steps) from state* q_0*.*

Finally, we can define what is the value of strategies (for both players) and of the game.

▶ **Definition 10** (Value of strategies and of the game). *The value* $\chi^{\mathcal{C}}_{\mathsf{s_A}}(q)$ *of a Player* A *strategy* $\mathsf{s_A}$ *from a state* $q \in Q$ *is equal to* $\chi^{\mathcal{C}}_{\mathsf{s_A}}(q) := \inf_{\mathsf{s_B} \in \mathsf{S}^{\mathsf{B}}_{\mathcal{C}}} \mathbb{P}^{q}_{\mathsf{s_A},\mathsf{s_B}}(\top)$*. The value* $\chi^{\mathcal{C}}_{\mathsf{A}}(q)$ *of the game for Player* A *from* q *is:* $\chi^{\mathcal{C}}_{\mathsf{A}}(q) := \sup_{\mathsf{s_A} \in \mathsf{S}^{\mathsf{A}}_{\mathcal{C}}} \chi^{\mathcal{C}}_{\mathsf{s_A}}(q)$*. It is analogous for Player* B*, by inverting the* inf *and* sup*. When equality of these two values holds, it defines the* value *at state* q*, denoted* $\chi^{\mathcal{C}}(q)$*:* $\chi^{\mathcal{C}}(q) := \chi^{\mathcal{C}}_{\mathsf{A}}(q) = \chi^{\mathcal{C}}_{\mathsf{B}}(q) \in [0,1]$*. The value of the game is then given by the valuation* $\chi^{\mathcal{C}} \in [0,1]^Q$*. Since the game is finite, [13] gives that this equality is always ensured. A strategy* $\mathsf{s_A} \in \mathsf{S}^{\mathsf{A}}_{\mathcal{C}}$ *such that* $\chi^{\mathcal{C}}_{\mathsf{s_A}}(q) = \chi^{\mathcal{C}}_{\mathsf{A}}(q)$ *(resp.* $\chi^{\mathcal{C}}_{\mathsf{s_A}}(q) \geq \chi^{\mathcal{C}}_{\mathsf{A}}(q) - \varepsilon$ *for some* $\varepsilon > 0$*) is called a Player* A optimal *strategy (resp.* ε-optimal*) from state* q*. If* $\chi^{\mathcal{C}}_{\mathsf{s_A}} = \chi^{\mathcal{C}}_{\mathsf{A}}$*, the strategy* $\mathsf{s_A}$ *is* uniformly *optimal. This is defined analogously for Player* B*. For a valuation* $v \in [0,1]^Q$ *of the states, a Player* A *strategy* $\mathsf{s_A} \in \mathsf{S}^{\mathsf{A}}_{\mathcal{C}}$ *such that* $v \preceq \chi^{\mathcal{C}}_{\mathsf{s_A}}$ *is said to* guarantee *the valuation* v*.*

Value of the game and least fixed point. In the context of a reachability game, the value of the game is the least fixed point (lfp) of an operator on valuations on states. We define this operator here.

▶ **Definition 11** (Valuation of the Nature states and operator on values). *For* $v \in [0,1]^Q$*, we define the valuation* $\mu_v \in [0,1]^{\mathsf{D}}$ *of the Nature states by* $\mu_v(d) := \sum_{q \in Q} \mathsf{dist}(d)(q) \cdot v(q)$ *for all* $d \in \mathsf{D}$*. For the operator* $\Delta : [0,1]^Q \to [0,1]^Q$*, for all valuations* $v \in [0,1]^Q$*, we set* $\Delta(v)(\top) := 1$ *and, for all* $q \neq \top \in Q$*, we set* $\Delta(v)(q) := \mathsf{val}_{\mathcal{F}^{\mu_v}_q}$*.*

As the operator Δ is monotonous, it has an lfp for the product order $\preceq$. This lfp gives the value of the game. Furthermore, Player B has an optimal positional strategy:

▶ **Theorem 12** ([8, 9]). *Let* m *denote the lfp of the operator* Δ*. Then:* $\chi^{\mathcal{C}} = \mathsf{m}$*. Furthermore, there exists a positional strategy* $\mathsf{s_B} \in \mathsf{PS}^{\mathcal{C}}_{\mathsf{B}}$ *for Player* B *ensuring* $\chi^{\mathcal{C}}_{\mathsf{s_B}} = \chi^{\mathcal{C}} = \mathsf{m}$*.*

Markov decision process induced by a positional strategy. Once a Player A positional strategy is fixed, we obtain a Markov decision process, which, informally, is a game where only one player (here, Player B) plays (against probabilistic transitions).

▶ **Definition 13** (Induced Markov decision process). *Consider a Player* A *positional strategy* $\mathsf{s_A} \in \mathsf{PS}^{\mathsf{A}}_{\mathcal{C}}$*. The* Markov decision process Γ *(MDP for short) induced by the strategy* $\mathsf{s_A}$ *is the triplet* $\Gamma := \langle Q, B, \iota \rangle$ *where* Q *is the set of states,* B *is the set of actions and* $\iota : Q \times B \to \mathcal{D}(Q)$ *is a map associating to a state and an action a distribution over the states. For all* $q \in Q$*,* $b \in B$ *and* $q' \in Q$*, we set* $\iota(q,b)(q') := p^{q,q'}(\mathsf{s_A}(q), b)$*.*

Note that the set of Player B strategies in an induced MDP Γ is the same as in the concurrent game $\mathcal{C}$. Furthermore, the useful objects in MDPs are the end components [5]: informally, sub-MDPs that are strongly connected.

▶ **Definition 14** (End component). *Consider a Player* A *positional strategy* $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^\mathsf{A}_\mathcal{C}$ *and consider the MDP* Γ *induced by that strategy. An* end component *(EC for short)* H *in* Γ *is a pair* (Q_H, β) *such that* $Q_H \subseteq Q$ *is a subset of states and* $\beta : Q_H \to \mathcal{P}(B) \setminus \emptyset$ *associates to each state a non-empty set of actions compatible with the EC* H *such that:*

- *for all* $q \in Q_H$ *and* $b \in \beta(q)$*, we have* $\mathsf{Supp}(\iota(q, b)) \subseteq Q_H$*;*
- *the underlying graph* (Q_H, E) *is strongly connected where* $(q, q') \in E$ *iff* $q' \in \mathsf{Supp}(\iota(q, \beta(q)))$*.*

We denote by $\mathsf{D}_H \subseteq \mathsf{D}$ *the set of Nature states compatible with the EC* H*:* $\mathsf{D}_H = \{d \in \mathsf{D} \mid \mathsf{Supp}(d) \subseteq Q_H\}$*. Note that, for all* $q \in Q_H$ *and* $b \in \beta(q)$*, we have* $\delta(q, \mathsf{Supp}(\mathsf{s}_\mathsf{A}(q)), b) \subseteq \mathsf{D}_H$*.*

The interest of ECs lies in the proposition below: in the MDP induced by a Player A strategy, for all Player B (positional) strategies (thus inducing a Markov chain), from all states, there is a non-zero probability to reach an EC from which it is impossible to exit.

▶ **Proposition 15.** *Consider a Player* A *positional strategy* $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^\mathsf{A}_\mathcal{C}$*. Let* $\mathcal{H}$ *denote the set of all ECs in the MDP induced by the strategy* s_A*. For all Player* B *strategies* $\mathsf{s}_\mathsf{B} \in \mathsf{PS}^\mathsf{B}_\mathcal{C}$*, there exists a subset of end components* $\mathcal{H}_{\mathsf{s}_\mathsf{B}} \subseteq \mathcal{H}$ *called* bottom strongly conneted components *(BSCC for short): for all* $H = (Q_H, \beta) \in \mathcal{H}_{\mathsf{s}_\mathsf{B}}$ *and* $q \in Q_H$*, we have* $\mathbb{P}^q_{\mathsf{s}_\mathsf{A}, \mathsf{s}_\mathsf{B}}(Q \setminus Q_H) = 0$*. Furthermore, if* $q \in Q$*, we have:* $\mathbb{P}^q_{\mathsf{s}_\mathsf{A}, \mathsf{s}_\mathsf{B}}(n, \cup_{H \in \mathcal{H}_{\mathsf{s}_\mathsf{B}}} H) > 0$ *where* $n = |Q|$*.*

5 Crucial proposition

We fix a concurrent reachability game $\langle \mathcal{C}, T \rangle$ and a valuation $v \in [0,1]^Q$ of the states that Player A wants to guarantee. That is, she seeks a strategy s_A ensuring that for all $q \in Q$, it holds $\chi^\mathcal{C}_{\mathsf{s}_\mathsf{A}}(q) \geq v(q)$. In particular, when $v = \mathsf{m}$, such a strategy s_A would be optimal. We state a sufficient condition for Player A positional strategies to ensure such a property.

Consider a Player A positional strategy $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^\mathcal{C}_\mathsf{A}$. The probability distribution chosen by this strategy only depends on the current state. In fact, this strategy is built with one (local) strategy per local interaction: for all state $q \in Q$, $\mathsf{s}_\mathsf{A}(q) \in \mathcal{D}(A)$ is a strategy in the game form $\mathcal{F}_q$. As Player A wants to guarantee the valuation v, the valuation of interest of the outcomes of the game form $\mathcal{F}_q = \langle A, B, \mathsf{D}, \delta(q, \cdot, \cdot) \rangle$ is $\mu_v \in [0,1]^\mathsf{D}$ – lifting the valuation v to the Nature states. To ensure that $\chi^\mathcal{C}_{\mathsf{s}_\mathsf{A}}(q) \geq v(q)$, one may think that it suffices to choose $\mathsf{s}_\mathsf{A}(q)$ so that its value in the game in normal form $\mathcal{F}^{\mu_v}_q$ is at least $v(q)$, that is: $\mathsf{val}_{\mathcal{F}^{\mu_v}_q}(\mathsf{s}_\mathsf{A}(q)) \geq v(q)$. In that case, the strategy s_A is said to locally dominate the valuation v:

▶ **Definition 16** (Strategy locally dominating a valuation). *A Player* A *positional strategy* $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^\mathsf{A}_\mathcal{C}$ locally dominates *the valuation* v *if, for all* $q \in Q$*, we have:* $\mathsf{val}_{\mathcal{F}^{\mu_v}_q}(\mathsf{s}_\mathsf{A}(q)) \geq v(q)$*.*

However, this is not sufficient in the general case, as examplified in Figure 1. For the valuation $v = \chi^\mathcal{C}$ such that $v(q_0) = v(\top) = 1$ and $v(\bot) = 0$, a Player A positional strategy s_A that plays the first row in $\mathcal{F}_{q_0}$ with probability 1 ensures that $\mathsf{val}_{\mathcal{F}^{\mu_v}_{q_0}}(\mathsf{s}_\mathsf{A}(q_0)) = 1 \geq v(q_0)$. However, we have seen that it does not ensure that $\chi^\mathcal{C}_{\mathsf{s}_\mathsf{A}}(q_0) = 1$ since, if Player B always plays the first column, the game indefinitely loops in q_0. The issue is that, in the MDP induced by the strategy s_A, the trivial end component $\{q_0\}$ is a trap, as it does not intersect the target set $\top$ – and therefore, the probability to reach $\top$ from q_0 is equal to 0 – whereas $\chi^\mathcal{C}(q_0) > 0$. In fact, as soon as this issue is avoided, if the strategy s_A locally dominates the valuation v, the desired property on s_A holds. Indeed:

▶ **Proposition 17.** *Consider a Player* A *positional strategy* $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^\mathsf{A}_\mathcal{C}$ *locally dominating* v*, and assume that* $v \preceq \mathsf{m}$*. Assume that for all end components* $H = (Q_H, \beta)$ *in the MDP induced by the strategy* s_A*, if* $Q_H \neq \{\top\}$*, for all* $q_H \in Q_H$*, we have* $\chi^\mathcal{C}(q_H) = 0$ *(in other words, for all* $q \in Q$*, if* $\chi^\mathcal{C}_{\mathsf{s}_\mathsf{A}}(q) = 0$ *then* $\chi^\mathcal{C}(q) = 0$*). In that case, for all* $q \in Q$*, we have* $\chi^\mathcal{C}_{\mathsf{s}_\mathsf{A}}(q) \geq v(q)$ *(i.e. the strategy* s_A *guarantees the valuation* v*).*

Proof Sketch. Consider some $\varepsilon > 0$ and, for $x \in \{\varepsilon, \varepsilon/2\}$, the valuations $v_x = v - x \in [0,1]^Q$. We show that s_A guarantees v_ε. As this holds for all $\varepsilon > 0$, it follows that s_A guarantees v. Consider an arbitrary positional strategy s_B for Player B. Let κ_A be a Player A strategy guaranteeing $v_{\varepsilon/2}$ in $n \geq 0$ steps from every state (which exists since $v_{\varepsilon/2} \prec \mathsf{m}$) and a strategy κ_B for Player B optimal against κ_A. So $\mathbb{P}^q_{\kappa_\mathsf{A},\kappa_\mathsf{B}}(n, \top) \geq v_{\varepsilon/2}(q)$ for all $q \in Q$. Now, for all $l \geq 0$, we consider the strategy s^l_A that plays s_A l times and then plays κ_A (and similarly for a strategy s^l_B for Player B). As s_A locally dominates v, it also locally dominates $v_{\varepsilon/2}$ which is obtained from v by translation. Therefore, for any state $q \in Q$, if the local strategy $\mathsf{s}_\mathsf{A}(q)$ is played in q, then the convex combination of the values of the successors of q w.r.t. the valuation $v_{\varepsilon/2}$ is at least $v_{\varepsilon/2}(q)$. In other words, the probability to reach $\top$ from q in $1+n$ steps if the strategy s^1_A is played is at least $v_{\varepsilon/2}(q)$: $\mathbb{P}^q_{\mathsf{s}^1_\mathsf{A},\mathsf{s}^1_\mathsf{B}}(1+n, \top) \geq v_{\varepsilon/2}(q)$. In fact, by induction, this holds for all $l \geq 0$: $\mathbb{P}^q_{\mathsf{s}^l_\mathsf{A},\mathsf{s}^l_\mathsf{B}}(l+n, \top) \geq v_{\varepsilon/2}(q)$. Now, with strategies s^l_A and s^l_B, consider the state of the game after l steps: either it is in a BSCC (w.r.t. s_A and s_B) or it is not. For a sufficiently large l, the probability not to have reached a BSCC is as close to 0 as we want. Furthermore, for a state q_H in a BSCC H that is not $\{\top\}$, by assumption, we have that $\chi^\mathcal{C}(q_H) = 0$, hence $\mathbb{P}^{q_H}_{\kappa_\mathsf{A},\kappa_\mathsf{B}}(\top) = 0$. In addition, if the state is in the trivial BSCC $\{\top\}$, then $\top$ is reached. Hence, for l large enough, the two probabilities $\mathbb{P}^q_{\mathsf{s}^l_\mathsf{A},\mathsf{s}^l_\mathsf{B}}(l+n, \top)$ and $\mathbb{P}^q_{\mathsf{s}^l_\mathsf{A},\mathsf{s}^l_\mathsf{B}}(l, \top)$ are as close to one another as we want. Finally, note that the strategies $\mathsf{s}^l_\mathsf{A}, \mathsf{s}^l_\mathsf{B}$ behave exactly like the strategies $\mathsf{s}_\mathsf{A}, \mathsf{s}_\mathsf{B}$ in the first l steps. That is, for l large enough, and $q \in Q$, we have $\mathbb{P}^q_{\mathsf{s}_\mathsf{A},\mathsf{s}_\mathsf{B}}(\top) \geq \mathbb{P}^q_{\mathsf{s}_\mathsf{A},\mathsf{s}_\mathsf{B}}(l, \top) = \mathbb{P}^q_{\mathsf{s}^l_\mathsf{A},\mathsf{s}^l_\mathsf{B}}(l, \top) \geq \mathbb{P}^q_{\mathsf{s}^l_\mathsf{A},\mathsf{s}^l_\mathsf{B}}(l+n, \top) - \varepsilon/2 \geq v_{\varepsilon/2}(q) - \varepsilon/2 = v_\varepsilon(q)$. ◀

Fix a Player A positional strategy s_A locally dominating the valuation v and let Γ be the MDP induced by s_A. For s_A to guarantee the valuation v, it suffices to ensure that any EC in Γ that is not the trivial EC $\{\top\}$ has all its states of value 0. It does not necessarily hold for s_A (recall the explanations before Proposition 17). However, we do have the following: fix an EC H in Γ. Then, all the states H have the same value w.r.t. the valuation v. It is stated in the proposition below.

▶ **Proposition 18.** *Consider a Player* A *positional strategy* $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^\mathsf{A}_\mathcal{C}$ *locally dominating a valuation* $v \in [0,1]^Q$*. For all EC* $H = (Q_H, \beta)$ *in the MDP induced by the strategy* s_A*, there exists* $v_H \in [0,1]$ *such that, for all* $q \in Q_H$*, we have* $v(q) = v_H$*. Furthermore, for all* $q \in Q_H$*, we have* $\mathsf{val}_{\mathcal{F}^{\mu_v}_q}(\mathsf{s}_\mathsf{A}(q)) = v(q)$*.*

6 Positional optimal and ε-optimal strategies

The aim of this section is, given a concurrent reachability game, to determine exactly from which states Player A has an optimal strategy. This, in turn, will give that whenever she has an optimal strategy, she has one that is positional which therefore extends Everett [8] (the existence of positional ε-optimal strategies). We fix a concurrent reachability game $\langle \mathcal{C}, \top \rangle$ for the rest of this section. Let us first introduce some terminology.

▶ **Definition 19** (Maximizable and sub-maximizable states)**.** *A state* $q \in Q$ *from which Player* A *has (resp. does not have) an optimal strategy is called* maximizable *(resp.* sub-maximizable*). The set of such states is denoted* MaxQ_A *(resp.* $\mathsf{SubMaxQ}_\mathsf{A}$*).*

The value of that game is given by the vector $\mathsf{m} \in [0,1]^Q$ (from Definition 11). We want to build an optimal (and positional) strategy for Player A when possible. To be optimal, a Player A positional strategy $\mathsf{s_A}$ has to play optimally at each local interaction $\mathcal{F}_q$ (for $q \in Q$) with regard to the valuation $\mu_{\mathsf{m}} \in [0,1]^{\mathsf{D}}$ (lifting the valuation m to Nature states). However, it is not sufficient in general: in the snow-ball game of Figure 1, when Player A plays optimally in $\mathcal{F}_{q_0}$ w.r.t. the valuation μ_{m} (that is, plays the first line with probability 1), Player B can enforce the play never to leave the state $q_0 \neq \top$. Hence, locally, we want to have strategies that not only play optimally but, regardless of the choice of Player B, have a non-zero probability to get closer to the target $\top$. Such strategies will be called *progressive strategies*. To properly define them, we introduce the following notation.

▶ **Definition 20** (Optimal action). *Let $q \in Q$ be a state of the game. Consider the game in normal form $\mathcal{F}_q^{\mu_{\mathsf{m}}}$. For all strategies $\sigma_{\mathsf{A}} \in \mathcal{D}(\mathsf{St_A})$, we define the set $B_{\sigma_{\mathsf{A}}}$ of* optimal actions *w.r.t. the strategy σ_{A} by $B_{\sigma_{\mathsf{A}}} := \{b \in B \mid \mathsf{out}_{\mathcal{F}_q^{\mu_{\mathsf{m}}}}(\sigma_{\mathsf{A}}, b) = \mathsf{val}_{\mathcal{F}_q^{\mu_{\mathsf{m}}}}(\sigma_{\mathsf{A}})\}$.*

In Figure 3, the set $B_{\sigma_{\mathsf{A}}}$ of optimal actions w.r.t. the strategy σ_{A} are represented in bold purple: the weighted values of these actions is the value of the strategy: $1/2$.

We can now define the set of *progressive* strategies.

▶ **Definition 21** (Progressive strategies). *Consider a state $q \in Q$ and a set of states $\mathsf{Gd} \subseteq Q$ that Player A wants to reach. The set of Nature states $\mathsf{Gd_D} \subseteq \mathsf{D}$ corresponds to the Nature states with a non-zero probability to reach the set Gd: $\mathsf{Gd_D} := \{d \in \mathsf{D} \mid \mathsf{Supp}(\mathsf{dist}(d)) \cap \mathsf{Gd} \neq \emptyset\}$. Then, the set of* progressive strategies $\mathsf{Prog}_q(\mathsf{Gd})$ *at state q w.r.t. Gd is defined by $\mathsf{Prog}_q(\mathsf{Gd}) := \{\sigma_{\mathsf{A}} \in \mathsf{Opt_A}(\mathcal{F}_q^{\mu_{\mathsf{m}}}) \mid \forall b \in B_{\sigma_{\mathsf{A}}},\ \delta(q, \mathsf{Supp}(\sigma_{\mathsf{A}}), b) \cap \mathsf{Gd_D} \neq \emptyset\}$.*

In Figure 3, the Nature states in $\mathsf{Gd_D}$ are arbitrarily chosen for the example and circled in green. The depicted strategy is progressive as, for all bold purple actions, there is a green-circled state in the support of the strategy (the circled $3/4$).

However, in an arbitrary game, some states may be sub-maximizable. In that case, playing optimally implies avoiding these states. Given a set $\mathsf{Bd} \subseteq Q$ of states to avoid, an optimal strategy that has a non-zero probability to reach that set of states Bd is called *risky*.

▶ **Definition 22** (Risky strategies). *Let $q \in Q$ be a state of the game and $\mathsf{Bd} \subseteq Q$ be a set of sub-maximizable states. The corresponding set of Nature states $\mathsf{Bd_D} \subseteq \mathsf{D}$ is defined similarly to $\mathsf{Gd_D}$ in Definition 21: $\mathsf{Bd_D} := \{d \in \mathsf{D} \mid \mathsf{Supp}(\mathsf{dist}(d)) \cap \mathsf{Bd} \neq \emptyset\}$. Then, the set of* risky strategies $\mathsf{Risk}_q(\mathsf{Bd})$ *at state q w.r.t. Bd is defined by $\mathsf{Risk}_q(\mathsf{Bd}) := \{\sigma_{\mathsf{A}} \in \mathsf{Opt_A}(\mathcal{F}_q^{\mu_{\mathsf{m}}}) \mid \exists b \in B_{\sigma_{\mathsf{A}}},\ \delta(q, \mathsf{Supp}(\sigma_{\mathsf{A}}), b) \cap \mathsf{Bd_D} \neq \emptyset\}$.*

In Figure 3, the set of Nature states $\mathsf{Bd_D}$ are also arbitrarily chosen for the example and circled in red. The strategy σ_{A} is not risky since no red-squared state appears in the intersection of the support of σ_{A} and the purple actions in B_{σ_A}.

In fact, we want for local strategies to be *efficient*, that is both progressive and not risky.

▶ **Definition 23** (Efficient strategies). *Let $q \in Q$ be a state of the game and $\mathsf{Gd}, \mathsf{Bd} \subseteq Q$ be sets of states. The set of* efficient strategies $\mathsf{Eff}_q(\mathsf{Gd}, \mathsf{Bd})$ *at state q w.r.t. Gd and Bd is defined by $\mathsf{Eff}_q(\mathsf{Gd}, \mathsf{Bd}) := \mathsf{Prog}_q(\mathsf{Gd}) \setminus \mathsf{Risk}_q(\mathsf{Bd})$.*

In Figure 3, the strategy σ_{A} is efficient as it is both progressive and not risky.

We can now compute inductively the set of maximizable and sub-maximizable states. First, given a set of sub-maximizable states Bd, we define iteratively below a set of *secure* states w.r.t. Bd, there are the states with a non-zero probability to get closer to the target $\top$ while avoiding the set Bd. The construction is illustrated in Figure 4.

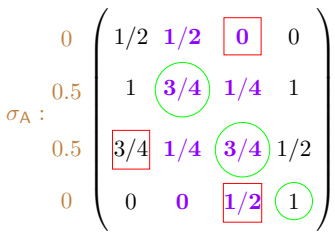


Figure 3 A game in normal form with an optimal strategy depicted in brown on the left. Its value is $1/2 = 1/2 \cdot 3/4 + 1/2 \cdot 1/4$.

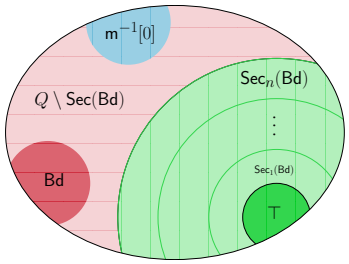


Figure 4 The construction of Definition 24 of the set of states $\mathsf{Sec}(\mathsf{Bd})$: it is the reunion of the blue and green vertical stripe areas.

▶ **Definition 24** (Secure states). *Consider a set of states* $\mathsf{Bd} \subseteq Q$. *We set* $\mathsf{Sec}_0(\mathsf{Bd}) := \{\top\}$ *and, for all* $i \geq 0$, $\mathsf{Sec}_{i+1}(\mathsf{Bd}) := \mathsf{Sec}_i(\mathsf{Bd}) \cup \{q \in Q \setminus \mathsf{Bd} \mid \mathsf{Eff}_q(\mathsf{Sec}_i(\mathsf{Bd}), \mathsf{Bd}) \neq \emptyset\}$. *The set* $\mathsf{Sec}(\mathsf{Bd})$ *of states* secure *w.r.t.* Bd *is:* $\mathsf{Sec}(\mathsf{Bd}) := \cup_{n\in\mathbb{N}} \mathsf{Sec}_n(\mathsf{Bd}) \cup \mathsf{m}^{-1}[0]$.

Note that, as the game $\mathcal{C}$ is finite, this procedure ends in at most $n = |Q|$ steps. Furthermore, the states of value 0 are added since any state of value 0 is maximizable. The interest of this construction lies in the lemma below: if all states in Bd are sub-maximizable, then all states in $Q \setminus \mathsf{Sec}(\mathsf{Bd})$ also are.

▶ **Lemma 25.** *Assume that a set of states* Bd *is such that* $\mathsf{Bd} \subseteq \mathsf{SubMaxQ}_\mathsf{A}$. *Then, the set of states* $Q \setminus \mathsf{Sec}(\mathsf{Bd})$ *is such that* $Q \setminus \mathsf{Sec}(\mathsf{Bd}) \subseteq \mathsf{SubMaxQ}_\mathsf{A}$ *(these correspond to the red horizontal stripe areas in Figure 4).*

Proof Sketch. For an arbitrary Player A strategy $\mathsf{s}_\mathsf{A} \in \mathsf{S}^\mathsf{A}_\mathcal{C}$ to be optimal, it roughly needs, on all relevant paths, to be optimal. More precisely, on any finite path $\pi = \pi' \cdot q \in Q^+$ with a non-zero probability to occur if Player B plays (locally) optimal actions against the strategy s_A (called a relevant path), the strategy s_A needs to play an optimal (local) strategy in the local interaction $\mathcal{F}_q$ and it[1] has to be optimal from q in the reachability game. Therefore, on all relevant paths, the strategy s_A, locally, has to play optimal strategies that are not risky. However, in any local interaction of a state $q \in Q \setminus \mathsf{Sec}(\mathsf{Bd})$, there is no efficient strategies available to Player A. Therefore, if the game starts from a state $q \in Q \setminus \mathsf{Sec}(\mathsf{Bd})$ an optimal strategy s_A for Player A (which therefore is locally optimal but not progressive) would allow Player B to ensure staying in the set $Q \setminus \mathsf{Sec}(\mathsf{Bd})$ while playing optimal actions. In that case, the game never leaves the set $Q \setminus \mathsf{Sec}(\mathsf{Bd})$, which induces a value of 0, whereas $\chi^\mathcal{C}(q) > 0$ since $q \notin \mathsf{Sec}(\mathsf{Bd})$. Thus, there is no optimal strategy for Player A from a state in $Q \setminus \mathsf{Sec}(\mathsf{Bd})$. ◀

We define inductively the set of bad states (which, in turn, will correspond to the set of sub-maximizable states) below.

▶ **Definition 26** (Set of sub-maximizable states). *Let* $\mathsf{Bad}_0 := \emptyset$ *and, for all* $i \geq 0$, $\mathsf{Bad}_{i+1} := Q \setminus \mathsf{Sec}(\mathsf{Bad}_i)$. *Then, the set* Bad *of bad states is equal to* $\mathsf{Bad} := \cup_{n\in\mathbb{N}} \mathsf{Bad}_n$ *for* $n = |Q|$.

Note that, as in the case of the set of secure states, since the game $\mathcal{C}$ is finite, this procedure ends in at most $n = |Q|$ steps. Lemma 25 ensures that the set of states Bad is included in $\mathsf{SubMaxQ}_\mathsf{A}$. In addition, we have that there exists a Player A positional strategy optimal from all states q in its complement $\mathsf{Sec}(\mathsf{Bad}) = Q \setminus \mathsf{Bad}$, as stated in the lemma below.

[1] In fact, the residual strategy $\mathsf{s}^{\pi'}_\mathsf{A}$.

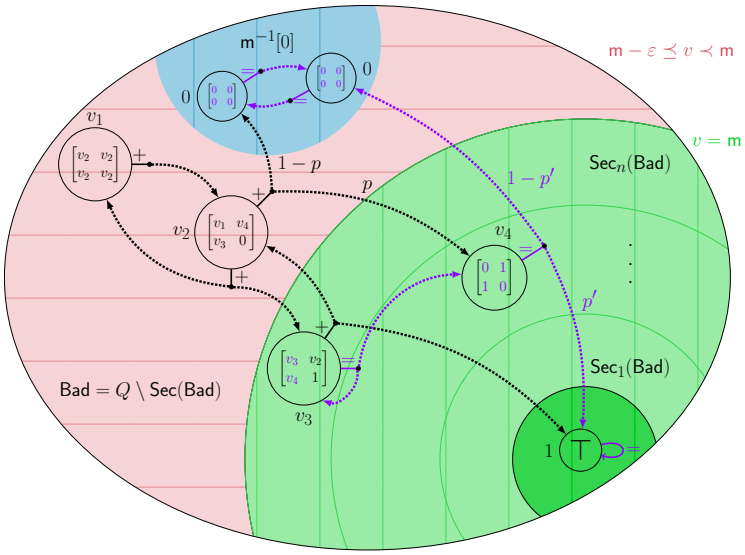


Figure 5 An illustration of the proof of Lemma 27 on the MDP induced by the strategy s_A. Labels $v_1, \dots, v_4$ is the value of the corresponding states given by the valuation v.

▶ **Lemma 27.** *For all $\varepsilon > 0$, there exists a positional strategy $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^{\mathcal{C}}_{\mathsf{A}}$ s.t.:*
- *for all $q \in \mathsf{Sec}(\mathsf{Bad})$, we have $\chi^{\mathcal{C}}_{\mathsf{s}_\mathsf{A}}(q) = \mathsf{m}(q)$;*
- *for all $q \in \mathsf{Bad}$, we have $\chi^{\mathcal{C}}_{\mathsf{s}_\mathsf{A}}(q) \geq \mathsf{m}(q) - \varepsilon$.*

In particular, it follows that $\mathsf{Sec}(\mathsf{Bad}) \subseteq \mathsf{MaxQ}_\mathsf{A}$.

Proof Sketch. To prove this lemma, we define a Player A positional strategy $\mathsf{s}_\mathsf{A} \in \mathsf{PS}^{\mathsf{A}}_{\mathcal{C}}$, a valuation $v \in [0,1]^Q$ of the states, prove that the strategy s_A locally dominates that valuation and prove that the only EC compatible with s_A that is not the target has value 0. This will show that is guarantees the valuation v by applying Proposition 17. As we want the strategy s_A to be optimal from all secure states, we consider a partial valuation v such that $v|_{\mathsf{Sec}(\mathsf{Bad})} := \mathsf{m}|_{\mathsf{Sec}(\mathsf{Bad})}$ (we will define it later on Bad). Then, on all secure states $q \in \mathsf{Sec}_i(\mathsf{Bad})$, we set $\mathsf{s}_\mathsf{A}(q)$ to be an efficient strategy w.r.t. $\mathsf{Sec}_{i-1}(\mathsf{Bad})$ and Bad, i.e. $\mathsf{s}_\mathsf{A}(q) \in \mathsf{Eff}_q(\mathsf{Sec}_{i-1}(\mathsf{Bad}), \mathsf{Bad})$. In particular, $\mathsf{s}_\mathsf{A}(q)$ is optimal in the game form $\mathcal{F}_q$ w.r.t. the valuation μ_m. However, we know that no strategy can be optimal from states in Bad. Hence, we consider a valuation v that is ε-close to the valuation m on states in Bad for a well-chosen $\varepsilon > 0$. This ε is chosen so that the value of the local strategy $\mathsf{s}_\mathsf{A}(q)$ for $q \in \mathsf{Sec}(\mathsf{Bad})$ is at least $v(q)$ w.r.t. the valuation μ_v[2]. We can now define the valuation v and the strategy s_A on Bad such that the value of $\mathsf{s}_\mathsf{A}(q)$ in $\mathcal{F}_q$ w.r.t. μ_v is greater than $v(q)$: $\mathsf{val}_{\mathcal{F}_q^{\mu_v}}(\mathsf{s}_\mathsf{A}(q)) > v(q)$ (this requires a careful use the fact that the operator Δ from Section 4 is 1-Lipschitz). The valuation v and the strategy s_A are now completely defined on Q. By definition, the strategy s_A locally dominates the valuation v.

The MDP induced by the strategy s_A is schematically depicted in Figure 5. The different split arrows appearing in the figure correspond to the actions (or columns in the local interactions) available to Player B. Black +-labeled-split arrows correspond to the actions of Player B that increase the value of v (i.e. in a state q, such that the convex combination –

[2] Specifically, ε has to be chosen smaller than the smallest difference between the values of an optimal actions $b \in B_{\mathsf{s}_\mathsf{A}(q)}$ and a non-optimal action $b \in B_{\mathsf{s}_\mathsf{A}(q)}$.

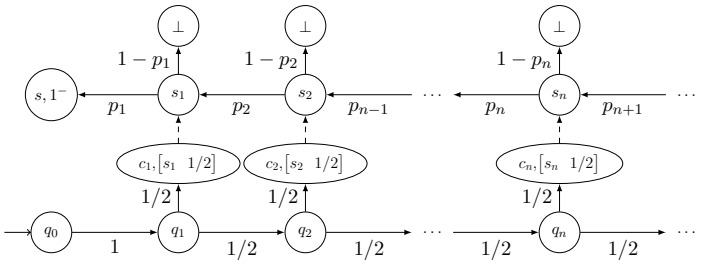


Figure 6 An infinite concurrent reachability game $\mathcal{C}$ (the Nature states are omitted). The probabilities p_k are such that, for all $i \geq 1$, the value of the state s_i is $\chi^{\mathcal{C}}(s_i) = \Pi_{k=1}^{i} p_k = (1/2+1/2^i)$.

w.r.t. to the probabilities chosen by the strategy s_A – of the values w.r.t. v of the successor states of q is greater than $v(q)$). For instance, we have $v_2 < p \cdot v_4 + (1-p) \cdot 0$, where the probability $p \in [0,1]$ is set by the strategy s_A. On the other hand, purple $=$-labeled-split arrows correspond to the actions whose values do not increase the value of the state. For instance $v_4 = (1-p') \cdot 0 + p' \cdot 1$. We can see that the only split arrows exiting states in Bad (the red horizontal stripe area) are black (since $\mathsf{val}_{\mathcal{F}_q^{\mu_v}}(\mathsf{s}_\mathsf{A}(q)) > v(q)$ for all $q \in \mathsf{Bad}$). However, from a secure state $q \in \mathsf{Sec}(\mathsf{Bad})$ (the green and blue vertical stripe areas) there are also purple split arrows. Note that, in these secure states $q \in \mathsf{Sec}(\mathsf{Bad})$, purple split arrows correspond to the optimal actions $B_{\mathsf{s}_\mathsf{A}(q)}$ at the local interaction $\mathcal{F}_q$. Furthermore, these split arrows cannot exit the set of secure states $\mathsf{Sec}(\mathsf{Bad})$ since the local strategy $\mathsf{s}_\mathsf{A}(q)$ is not risky.

We can then prove that the strategy s_A guarantees the valuation v by applying Proposition 17: since s_A locally dominates the valuation v, it remains to show that all the ECs different from $\{\top\}$ have only states of value 0. In the figure, this corresponds to having ECs only in the blue upper circle and dark green bottom right inner circle areas. In fact, Proposition 18 gives that any state q in an EC ensures $\mathsf{val}_{\mathcal{F}_q^{\mu_v}}(\mathsf{s}_\mathsf{A}(q)) = v(q)$, which implies that no state in Bad can be in an EC. This can be seen in the figure between the states of value v_1 and v_2: because of the black arrow from v_1 to v_2, we necessarily have $v_1 < v_2$. Then, v_2 cannot loop (with probability one) to v_1 since this would imply $v_2 < v_1$. As all the split arrows are black for states in Bad, no EC can appear in this region. Furthermore, the optimal actions in the secure states always have a non-zero probability to get closer to the target $\top$. In the figure, this corresponds to the fact that there is always one tip of a purple split arrow that goes down in the $(\mathsf{Sec}_i(\mathsf{Bad}))_{i\in\mathbb{N}}$ hierarchy (since the strategy $\mathsf{s}_\mathsf{A}(q)$ is progressive): in the example, from v_3 to v_4 and from v_4 to the target $\top$. Therefore, the only loop (with probability one) that can occur in the set $(\mathsf{Sec}_i(\mathsf{Bad}))_{i\in\mathbb{N}}$ is at the target $\top$. We conclude by applying Proposition 17. ◀

Overall, we obtain the theorem below summarizing the results proved in this section.

▶ **Theorem 28.** *In a concurrent reachability game $\langle \mathcal{C}, \top \rangle$, we have $\mathsf{Bad} = \mathsf{SubMaxQ}_\mathsf{A}$ and $\mathsf{Sec}(\mathsf{Bad}) = \mathsf{MaxQ}_\mathsf{A}$. Furthermore, for all $\varepsilon > 0$, there is a Player A positional strategy s_A optimal from all states in MaxQ_A and ε-optimal from all states in $\mathsf{SubMaxQ}_\mathsf{A}$.*

Infinite arenas. In this paper, we only consider finite arenas and the constructions we have exhibited and results we have shown hold in that setting. Note that Theorem 28 does not hold on infinite arenas (i.e. with an infinite number of states): Figure 6 depicts an infinite concurrent reachability game where the state q_0 is maximizable but, from q_0, Player A does not have any positional optimal strategy. Indeed, in state s is plugged the game of Figure 1,

whose value is 1 but Player A does not have an optimal strategy. Then, for all $i \geq 0$, the probability to reach s from s_i is equal to $v_i = (1/2 + 1/2^i) > 1/2$. Hence, if Player A plays an $0 < \varepsilon_i$-optimal strategy in s such that $(1 - \varepsilon_i) \cdot q_i > 1/2$, then the value of the state s_i is greater than $1/2$. In that case, in the states c_i, Player B plays the second columns obtaining the value $1/2$. This induces that the value in all states q_i is $1/2$. However, this is only possible if Player A has (infinite) memory, since the greater the index i considered, the smaller the value of ε_i needs to be to ensure $(1 - \varepsilon_i) \cdot q_i \geq 1/2$ while still ensuring $\varepsilon_i > 0$ (since Player A does not have an optimal strategy from s). In particular, for any Player A positional strategy s_A from q_0 that is $0 < \varepsilon$-optimal in s, the value – w.r.t. the strategy s_A – of all states s_i for indexes i such that $(1 - \varepsilon) \cdot q_i < 1/2$ is smaller than $1/2$. In which case, Player B plays the first column in c_i, thus obtaining a value smaller than $1/2$. It follows that the value of all states $(q_n)_{n\geq 0}$ – w.r.t. the strategy s_A – is smaller than $1/2$. Hence, any Player A positional strategy is not optimal from q_0. Note that, when considering MDPs instead of two-player games, optimal strategies need not exist but when they do there necessarily are positional ones (see for instance [10]).

Computing the set of maximizable states. Finally, consider the problem, given a finite concurrent reachability game, to effectively compute the set of maximizable and sub-maximizable states (assuming the probability distribution of the Nature states are rational). In fact, this can be done by using the theory of the reals.

▶ **Definition 29** (First-order theory of the reals). *The* first-order theory of the reals *(denoted* FO-$\mathbb{R}$*) corresponds to the well-formed sentences of first-order logic (i.e. with universal and existential quandtificators), also involving logical combinations of equalities and inequalities of real polynomials, with integer coefficients.*

The first-order theory of the reals is decidable [15], i.e. determining if a given formula belonging to that theory is true is decidable. Now, let us consider a finite concurrent reachability game $\mathcal{C}$ and a state $q \in Q$. It is possible to encode, with an FO-$\mathbb{R}$ formula, that the state q is maximizable, i.e. $q \in \mathsf{MaxQ}_\mathsf{A}$. First, note that, given two positional strategies s_A and s_B for both players, it is possible to compute the value of the game with the theory of reals: it amounts to finding the least fixed point of the operator Δ with the strategies of both players fixed. Then, q being maximizable, denoting $u := \chi^{\mathcal{C}}(q) \in [0, 1]$ its value, is equivalent to having a Player A positional strategy ensuring at least u (against all Player B positional strategies) and no Player A positional strategy ensures more than u (as ε-optimal positional strategies always exists for Player A [8]). This can be expressed in FO-$\mathbb{R}$. The theorem below follows.

▶ **Theorem 30.** *In a finite concurrent reachability game with rational distributions, the set of maximizable states is computable.*

7 Maximizable states and game forms

In the previous section, we were given a concurrent reachability game and we considered a construction to compute exactly the sets of maximizable and sub-maximizable states. It is rather cumbersome as it requires two nested fixed point procedures. Now, we would like to have a structural condition ensuring that if a game is built correctly (i.e. built from reach-maximizable local interactions), then all states are maximizable. More specifically, in this section, we characterize exactly the *reach-maximizable* game forms, that is the game forms such that every reachability game built with these game forms as local interactions have only maximizable states.

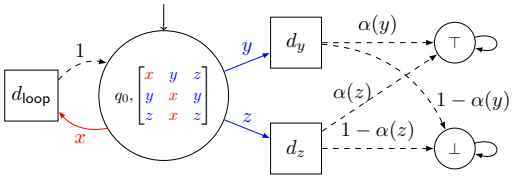


Figure 7 The three-state reachability game $\langle \mathcal{C}_{(\mathcal{F},\alpha)}, \top \rangle$ built from the game form $\mathcal{F}$ for some partial valuation $\alpha : \mathsf{O} \setminus E \to [0,1]$ with $E = \{x\}$.

$$\mathcal{F} = \begin{bmatrix} x & y & z \\ y & x & y \\ z & x & z \end{bmatrix}$$

Figure 8 The game form that constitutes the local interaction in the state q_0.

First, let us characterize a necessary condition for game forms to be reach-maximizable. We want for reach-maximizable game forms to behave properly when used individually. That is, from a game form $\mathcal{F}$ and a partial valuation $\alpha : \mathsf{O} \setminus E \to [0,1]$ of the outcomes, we define a three-state reachability game $\langle \mathcal{C}_{(\mathcal{F},\alpha)}, \top \rangle$. Note that such games were previously studied in [11]. We illustrate this construction on an example.

▶ **Example 31.** In Figure 7, a three-state reachability game $\langle \mathcal{C}_{(\mathcal{F},\alpha)}, \top \rangle$ is built from a game form $\mathcal{F} = \langle \mathsf{St_A}, \mathsf{St_B}, \{x,y,z\}, \varrho \rangle$ – with ϱ depicted in Figure 8 – and a partial valuation $\alpha : \{y,z\} \to [0,1]$. We have a one-to-one correspondence between the outcomes of the game form $\mathcal{F}$ and the Nature states of the reachability game $\langle \mathcal{C}_{(\mathcal{F},\alpha)}, \top \rangle$ via the bijection $g : \{x,y,z\} \to \mathsf{D}$ such that $g(x) = d_{\mathsf{loop}}$ and for $u \in \{y,z\}$, $g(u) = d_u$. Furthermore, in the reachability game $\langle \mathcal{C}_{(\mathcal{F},\alpha)}, \top \rangle$, we have $\mathsf{m}(\top) = 1$ and $\mathsf{m}(\bot) = 0$. Therefore, for $u \in \{y,z\}$, we have $\mu_{\mathsf{m}} \circ g(u) = \alpha(u)$. In fact, this game is built so that $v_\alpha = \mathsf{m}(q_0)$ and $\mu_{\mathsf{m}} = \tilde{\alpha} \circ g^{-1}$ (recall that $\tilde{\alpha}$ is the (total) valuation induced by the partial valuation α from Definition 4).

Let us now determine at which condition on the pair $(\mathcal{F}, \alpha)$ is the starting state q_0 maximizable in $\mathcal{C}_{(\mathcal{F},\alpha)}$. If we have $v_\alpha = \mathsf{m}(q_0) = 0$, the state q_0 is maximizable in any case. Now, assume that $v_\alpha = \mathsf{m}(q_0) > 0$. Recall the construction of the previous section, specifically the set of secure states w.r.t. a set of bad states (Definition 24). Initially, $\mathsf{Bad}_0 = \emptyset$, so we want for the state q_0 to be in $\mathsf{Sec}(\emptyset)$, i.e. we want (and need) an efficient strategy in the state q_0 where the set of good states Gd is the target $\mathsf{Gd} = \{\top\}$ and the set of bad states is empty. In that case, the set of efficient strategies coincide with the set of progressive strategies. Thus, q_0 is maximizable if and only if $\mathsf{Prog}_{q_0}(\{\top\}) \neq \emptyset$. We assume for simplicity that $\alpha(y), \alpha(z) > 0$, hence the set Nature states $\mathsf{Gd_D}$ with a non-zero probability to reach $\top$ is $\{g(y), g(z)\} \subseteq \mathsf{D}$. By definition of Prog (Definition 21), $\mathsf{Prog}_{q_0}(\{\top\}) \neq \emptyset$ amounts to have an optimal strategy σ_{A} in $\mathcal{F}_{q_0}^{\mu_{\mathsf{m}}}$ such that, for all $b \in B_{\sigma_{\mathsf{A}}} : \delta(q_0, \mathsf{Supp}(\sigma_{\mathsf{A}}), b) \cap \{g(y), g(z)\} \neq \emptyset$ or, equivalently, $\delta(q_0, \mathsf{Supp}(\sigma_{\mathsf{A}}), b) \not\subseteq \{g(x)\}$. In terms of $\mathcal{F}$ and α, the state q_0 is maximizable if and only if there is an optimal strategy σ_{A} in $\mathcal{F}^{\tilde{\alpha}}$ such that, for all $b \in B_{\sigma_{\mathsf{A}}} : \varrho(\mathsf{Supp}(\sigma_{\mathsf{A}}), b) \not\subseteq \{x\} = E$ if the partial valuation α is defined as $\alpha : \mathsf{O} \setminus E \to [0,1]$ for $\mathsf{O} = \{x,y,z\}$ and $E = \{x\}$.

This suggests the definition below of *reach-maximizable game form* w.r.t. a partial valuation.

▶ **Definition 32** (Reach-maximizable game forms w.r.t. a partial valuation)**.** *Consider a game form $\mathcal{F}$ and a partial valuation of the outcomes $\alpha : \mathsf{O} \setminus E \to [0,1]$. The game form $\mathcal{F}$ is* reach-maximizable *w.r.t. the partial valuation α if $v_\alpha = 0$ or there exists an optimal strategy $\sigma_{\mathsf{A}} \in \mathsf{Opt_A}(\mathcal{F}^{\tilde{\alpha}})$ such that for all $b \in B_{\sigma_{\mathsf{A}}}$, we have $\varrho(\mathsf{Supp}(\sigma_{\mathsf{A}}), b) \not\subseteq E$. Such strategies are said to be* reach-maximizing *w.r.t. α.*

This definition was chosen to ensure the lemma below.

▶ **Lemma 33.** *Consider a game form $\mathcal{F}$ and a partial valuation of the outcomes $\alpha : \mathsf{O} \setminus E \to [0,1]$. The initial state (and thus all states) in the three-state reachability game $\mathcal{C}_{(\mathcal{F},\alpha)}$ is maximizable if and only if the game form $\mathcal{F}$ is reach-maximizable w.r.t. the partial valuation α.*

The definition of reach-maximizable game form is then obtained via a universal quantification over the partial valuations considered.

▶ **Definition 34** (Reach-maximizable game form)**.** *Consider a game form $\mathcal{F} = \langle \mathsf{St_A}, \mathsf{St_B}, \mathsf{O}, \varrho \rangle$. It is a* reach-maximizable *(RM for short) game form if it is reach-maximizable w.r.t. all partial valuations $\alpha : \mathsf{O} \setminus E \to [0,1]$.*

Lemma 33 gives that RM game forms behave properly when used individually, such as in three-state reachability games. Let us now look at how such game forms behave collectively, that is we consider concurrent reachability games where all local interactions are RM. In fact, in such a setting, all states are maximizable. This is stated in the lemma below.

▶ **Lemma 35.** *Consider a concurrent reachability game $\langle \mathcal{C}, \top \rangle$ and assume that all local interactions are RM game forms. Then, all states are maximizable: $Q = \mathsf{MaxQ_A}$.*

Proof Sketch. We show that $Q = \mathsf{MaxQ_A}$ by showing that $\mathsf{Bad} = \emptyset$, which is equivalent since, by Theorem 28, we have $\mathsf{Bad} = \mathsf{SubMaxQ_A} = Q \setminus \mathsf{MaxQ_A}$. That is, we consider the iterative construction of the set of sub-maximizable states of the previous section and we show that $\mathsf{Bad}_1 = Q \setminus (\mathsf{Sec}(\mathsf{Bad}_0)) = \emptyset = \mathsf{Bad}_0$ (see Definition 26), which induces that $\mathsf{Bad} = \emptyset$. Let us assume towards a contradiction that $Q \setminus (\mathsf{Sec}_n(\emptyset) \cup \mathsf{m}^{-1}[0]) \neq \emptyset$ for $n = |Q|$. Since $\mathsf{Risk}_q(\emptyset) = \emptyset$ for all $q \in q$, any efficient strategy in a state q w.r.t. to the sets $\mathsf{Sec}_n(\emptyset)$ and $\emptyset$ is in fact a progressive strategy w.r.t. the set $\mathsf{Sec}_n(\emptyset)$. Hence, the goal is to exhibit such a progressive strategy in a state $q \in Q \setminus \mathsf{Sec}(\emptyset)$, thus showing a contradiction with the fact that $q \notin \mathsf{Sec}(\emptyset)$. We consider the states with the greatest value – w.r.t. m – as we can hope that they are the more likely to have progressive strategies. That is, for $x := \max_{q \in Q \setminus \mathsf{Sec}_n(\emptyset)} \mathsf{m}(q) > 0$ the maximum of m, we set $Q_x := \mathsf{m}^{-1}[x] \setminus \mathsf{Sec}_n(\emptyset) \neq \emptyset$ the set of states realizing that maximum. We want to use the assumption that all local interactions are RM. That is, we need to define a partial valuation on the outcomes of the local interactions, i.e. on Nature states. First, let us define its domain. We can find intuition in the example of the three-state reachability game in Figure 7: the outcome that is not valued by the partial valuation considered is the Nature state looping on the state q_0. Note that its value w.r.t. μ_{m} is the same as the value of the state q_0 w.r.t. m. In our case, we consider the set of Nature states D_x realizing this value x that cannot reach the set $\mathsf{Sec}_n(\emptyset)$, that is $\mathsf{D}_x := \mu_{\mathsf{m}}^{-1}[x] \setminus \mathsf{Sec}_n(\emptyset)_{\mathsf{D}}$. Then, we define the partial valuation of the Nature states $\alpha : \mathsf{D} \setminus \mathsf{D}_x \to [0,1]$ by $\alpha := \mu_{\mathsf{m}}|_{\mathsf{D} \setminus \mathsf{D}_x}$. Now, we can show that there exists a state $q \in Q_x$ such that $\tilde{\alpha} = \mu_{\mathsf{m}}$ in the game form $\mathcal{F}_q$. By maximality of x, we can prove that any local strategy σ_A in $\mathcal{F}_q$ that is reach-maximizing w.r.t. the partial valuation α of the outcomes of $\mathcal{F}_q$ is a progressive strategy w.r.t. $\mathsf{Sec}_n(\emptyset)$ in $\mathcal{F}_q$. Equivalently, σ_{A} is efficient w.r.t. $\mathsf{Sec}_n(\emptyset)$ and $\emptyset$. Hence the contradiction with the fact that $q \notin \mathsf{Sec}(\emptyset)$. ◀

Overall, we obtain the theorem below.

▶ **Theorem 36.** *For a set of game forms $\mathcal{G}$, all states in all concurrent reachability games with local interactions in $\mathcal{G}$ are maximizable if and only if all game forms in $\mathcal{G}$ are RM.*

Deciding if game forms are RM. Consider the following decision problem RMGF: given a game form, decide if it is a RM game form. We proved Theorem 30 by showing that the fact that a state is maximizable in a concurrent reachability game can be encoded in the

theory of the reals (FO-$\mathbb{R}$). Since Lemma 33 ensures that a game form $\mathcal{F}$ is RM w.r.t. a partial valuation α if and only if the initial state in the three-state reachability game $\mathcal{C}_{(\mathcal{F},\alpha)}$ is maximizable, it follows that, via a universal quantification over partial valuations, the fact that a game form is RM can be encoded in the theory of the reals. Note that it can also be encoded directly from the definition of RM game form. We obtain the theorem below.

▶ **Proposition 37.** *The problem* RMGF *is decidable.*

Determined game forms and RM game forms. In [2], the authors have studied a problem similar to the one we considered in this section: determining the game forms ensuring that, when used as local interaction in a concurrent game (with an arbitrary Borel winning condition), the game is determined (i.e. either of the players has a winning strategy). The authors have shown that these game forms exactly correspond to *determined* game forms. These roughly correspond to game forms where, for all subsets of outcomes $E \subseteq \mathsf{O}$, there is either of line of outcomes in E or a column of outcomes in $\mathsf{O} \setminus E$, as formally defined below.

▶ **Definition 38** (Determined game forms)**.** *Consider a game form* $\mathcal{F} = \langle \mathsf{St_A}, \mathsf{St_B}, \mathsf{O}, \varrho \rangle$*. It is* determined *if, for all subsets of outcomes* $E \subseteq \mathsf{O}$*, either there exists some* $a \in \mathsf{St_A}$ *such that* $\varrho(a, \mathsf{St_B}) \subseteq E$ *or there exists some* $b \in \mathsf{St_B}$ *such that* $\varrho(\mathsf{St_A}, b) \subseteq \mathsf{O} \setminus E$*.*

In fact, they proved an equivalence between turn-based games and concurrent games using determined game forms as local interactions, which holds also when the game is stochastic. In fact, positional optimal strategies exists for both players in turn-based reachability games [4], it is also the case in concurrent reachability games with determined local interactions. This result, combined with Theorem 36 gives immediately that determined game forms are RM. Interestingly, determined game forms can also be characterized with the least fixed point operator as in the proposition below.

▶ **Proposition 39.** *A game form* $\mathcal{F}$ *is* determined *if and only if, for all partial valuations* $\alpha : \mathsf{O} \setminus E \to [0,1]$ *of the outcomes, we have* $v_\alpha = f_\alpha^{\mathcal{F}}(0)$*. In particular, this implies that all determined game forms are RM.*

8 Future Work

In this paper we give a double-fixed-point procedure to compute maximizable and submaximizable states in a stochastic concurrent reachability (finite) game. Our procedure yields *de facto* positional witnesses for the strategies. As further natural work, we seek studying more general objectives. It is however interesting to notice that, as mentioned in the introduction, it will not be so easy since even Büchi games do not enjoy positional almost optimal strategies [7, Theorem 2].

We also plan to better grasp RM game forms, and understand what are RM game forms for the two players, or analyze the complexity of the RMGF problem.

References

1 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. *Handbook of Model Checking*, chapter Graph games and reactive synthesis, pages 921–962. Springer, 2018.

2 Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. From local to global determinacy in concurrent graph games. Technical Report abs/2107.04081, CoRR, 2021. `arXiv:2107.04081`.

3 Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. Optimal strategies in concurrent reachability games. *CoRR*, abs/2110.14724, 2021. `arXiv:2110.14724`.

4 Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. Quantitative stochastic parity games. In *Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 121–130. SIAM, 2004.

5 Luca de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

6 Luca de Alfaro, Thomas Henzinger, and Orna Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3):188–217, 2007.

7 Luca de Alfaro and Rupak Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68:374–397, 2004.

8 Hugh Everett. Recursive games. *Annals of Mathematics Studies – Contributions to the Theory of Games*, 3:67–78, 1957.

9 Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

10 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Strategy complexity of parity objectives in countable mdps. In *Proc. 31st International Conference on Concurrency Theory (CONCUR'20)*, volume 171 of *LIPIcs*, pages 39:1–39:17. Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.39.

11 Elon Kohlberg. Repeated games with absorbing states. *The Annals of Statistics*, pages 724–738, 1974.

12 Antonín Kučera. *Lectures in Game Theory for Computer Scientists*, chapter Turn-Based Stochastic Games, pages 146–184. Cambridge University Press, 2011.

13 Donald A. Martin. The determinacy of blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

14 Annabelle McIver and Carroll Morgan. Games, probability and the quantitative μ-calculus $qm\mu$. In *Proc. 9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, volume 2514 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2002.

15 James Renegar. On the computational complexity and geometry of the first-order theory of the reals. part iii: Quantifier elimination. *Journal of Symbolic Computation*, 13(3):329–352, 1992. doi:10.1016/S0747-7171(10)80005-7.

16 Wolfgang Thomas. Infinite games and verification. In *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 58–64. Springer, 2002. Invited Tutorial.

17 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.

18 John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton Univ. Press, Princeton, 1944.

19 Wiesław Zielonka. Perfect-information stochastic parity games. In *Proc. 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'04)*, volume 2987 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2004.

Finite-Memory Strategies in Two-Player Infinite Games

Patricia Bouyer
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France

Stéphane Le Roux
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France

Nathan Thomasset
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France

Abstract

We study infinite two-player win/lose games (A, B, W) where A, B are finite and $W \subseteq (A \times B)^\omega$. At each round Player 1 and Player 2 concurrently choose one action in A and B, respectively. Player 1 wins iff the generated sequence is in W. Each history $h \in (A \times B)^*$ induces a game (A, B, W_h) with $W_h := \{\rho \in (A \times B)^\omega \mid h\rho \in W\}$. We show the following: if W is in Δ_2^0 (for the usual topology), if the inclusion relation induces a well partial order on the W_h's, and if Player 1 has a winning strategy, then she has a finite-memory winning strategy. Our proof relies on inductive descriptions of set complexity, such as the Hausdorff difference hierarchy of the open sets.

Examples in Σ_2^0 and Π_2^0 show some tightness of our result. Our result can be translated to games on finite graphs: e.g. finite-memory determinacy of multi-energy games is a direct corollary, whereas it does not follow from recent general results on finite memory strategies.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Two-player win/lose games, Infinite trees, Finite-memory winning strategies, Well partial orders, Hausdorff difference hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.8

Related Version *Full Version*: https://arxiv.org/abs/2107.09945

1 Introduction

Two-player win/lose games have been a useful tool in various areas of logic and computer science. The two-player win/lose games in this article consist of infinitely many rounds. At each round i, Player 1 and Player 2 concurrently choose one *action* each, i.e. a_i and b_i in their respective sets A and B. Player 1 wins and Player 2 loses if the play $(a_0, b_0)(a_1, b_1)\dots$ belongs to a fixed $W \subseteq (A \times B)^\omega$. Otherwise Player 2 wins and Player 1 loses. We call W the *winning set* of Player 1, or the winning condition for Player 1. A *strategy* is a map that tells a player how to play after any finite *history* of actions played: a Player 1 (resp. 2) strategy is a map from $(A \times B)^*$ to A (resp. B). A strategy is *finite-memory* (FM) if the map can be implemented by a finite-state machine. Also, each history $h \in (A \times B)^*$ induces a game starting at h and taking the past into account, i.e. with winning set $W_h := \{\rho \in (A \times B)^\omega \mid h\rho \in W\}$.

For now we state a slightly weaker version of our main result: if A and B are finite, if the $(W_h)_{h \in (A \times B)^*}$ constitute a well partial order (wpo) for the inclusion, if $W \in \Delta_2^0$, i.e. in the usual cylinder topology W is a countable union of closed sets and a countable intersection of open sets, and if Player 1 has a winning strategy, then she has a finite-memory winning strategy. Of course, our result also applies to the turn-based version of such games.

On the proof of the main result. The proof of our main result relies on descriptive set theory. The Hausdorff-Kuratowski theorem (see, e.g., [5]) states that each set in Δ_2^0 can be expressed as an ordinal difference of open sets, and conversely. In general, this implies that properties of sets in Δ_2^0 may be proved by induction over the countable ordinals. Accordingly, we prove our main result by induction on W, but the inductive step suggested by the Hausdorff-Kuratowski theorem does not suit us completely. Instead, we mix it with a folklore alternative way of describing Δ_2^0 by induction. After this mix, our base case consists of the open sets, the first inductive step consists of *union with a closed set*, and the second inductive step of *open union*.

To prove our result, the base case, where W is open, amounts to reachability games, and the wpo assumption is not needed. The case where W is closed is easy, and it includes the multi-energy games (where Player 1 keeps all energy levels positive). Just above these, the case where W is the union of an open set and a closed set is harder to prove. It includes disjunctions of a reachability condition and a multi-energy condition. We will present this harder case in details because it shows part of the complexity of the full result.

Yet another representation of Δ_2^0. Above, we mentioned two hierarchies that describe Δ_2^0. In addition, this paper (re-)proves the folklore result that Π_2^0 corresponds to Büchi winning conditions and Σ_2^0 to co-Büchi. If labeling each history with 0 or 1, the Büchi (co-Büchi) condition requires that infinitely (only finitely) many 1's be seen on a branch/play. The Δ_2^0 sets are therefore the sets that can be expressed both by Büchi and co-Büchi conditions. This is possible exactly if on every infinite play not both 0 and 1 occurs infinitely often. This corresponds to the play's crossing only finitely many layers in the previous paragraph.

From the four representations and the layer intuition, we can conclude that Δ_2^0 sets are infinite Boolean combinations of open sets, of course in a restricted sense.

Tightness of the result. The collection of the countable unions of closed sets is called Σ_2^0, and the collection of their complements, i.e. the countable intersections of open sets, is called Π_2^0. So $\Delta_2^0 = \Sigma_2^0 \cap \Pi_2^0$. In this article we provide one example of a winning set W in Σ_2^0 and one example in Π_2^0 that satisfy the wpo assumption but not the FM-strategy sufficiency. Hence tightness.

Note that without the wpo assumption, even Turing-computable strategies may not suffice to win for closed winning sets: take a non-computable binary sequence ρ and a game where Player 1 wins iff she plays ρ. She has a winning strategy, but no computable ones.

Connections with graph games. Our game $\langle A, B, W\rangle$ can be seen as a one-state concurrent graph game where the winning condition is defined via the actions rather than the visited states. Winning strategies (resp. FM strategies) coincide in both models.

Alternatively, we can unfold any concurrent graph game into an infinite tree game $\langle A, B, W\rangle$ whose nodes are the histories of pairs of actions. Winning strategies coincide in both models. Moreover, an FM strategy in the tree game is, up to isomorphism, also an FM strategy in the graph game. The converse may not hold since, informally, the player may observe the current state only in the graph model. Nevertheless for finite graphs, the observation of the state can be simulated by an additional finite memory, i.e. the graph itself. To sum up, any FM result in our tree games can be translated into an FM result in graph games. Almost conversely, FM results in finite-graph games can be obtained from our tree games, possibly with non-optimal memory.

Related works and applications. The two articles [7] and [2] provide abstract criteria to show finite-memory determinacy in finite-graph games: [7] by Boolean combination of complex FM winning conditions with simple winning conditions defined via regular languages; [2] by characterizing, for a fixed memory, the winning conditions that yield, in all finite-graph games, FM determinacy via this fixed memory. The FM determinacy of multi-energy games is a corollary of neither, but as mentioned above, it is a direct corollary of our result. (Note that this specific result was already proved in [10].)

More generally on a finite-graph game, consider the conjunction or disjunction of a multi-energy winning condition and a Boolean combination of reachability conditions. This is in Δ_2^0 (actually low at some finite level of the hierarchy), this induces a wpo, so if Player 1 has a winning strategy she has an FM one.

However, the FM determinacy of Büchi games in finite-graph games is not a corollary of our result, because the Büchi conditions may not be in Δ_2^0. We mention three things about this. First, this determinacy does not contradict our tightness results: in Π_2^0 or Σ_2^0, FM determinacy holds when the corresponding labeling is regular. Second, in future work we plan to seek a general theorem having both this determinacy and our main result as special cases. Third, in finite-graph games, many winning conditions that yield memoryless or FM determinacy can be simulated by finite games, and therefore clopen winning conditions, i.e. Δ_1^0 instead of our more general Δ_2^0. For instance, see [1], [8], [4]. This suggests that our work could be used to prove more FM sufficiency results by reduction of finite-graph games to tree games with wpo winning condition in Δ_2^0.

Structure of the article. Section 2 defines our games and finite-memory strategies; Section 3 presents the related descriptive set theory; Section 4 presents our main result; Section 5 discusses tightness of our main result; Section 6 mentions possible future work.

2 Setting and definitions

We study two-player games, which consist in a tuple (A, B, W): A is the *action set* for Player 1, B is the action set for Player 2 and $W \subseteq (A \times B)^\omega$ is the *winning set*. Here we only consider finitely branching games, where A and B are both finite. Such a game is played in the following way: at each round, each player chooses an action from their respective action set in a concurrent way, thus producing a pair of actions in $A \times B$. The game then continues for infinitely many rounds, generating a *play* which consists in an infinite word in $(A \times B)^\omega$. Player 1 then wins if the generated play belongs to the winning set W, while Player 2 wins if it does not. In the following we will focus on Player 1.

To describe the Players' behavior in such a game, we use the concepts of *histories* and *strategies*. A history is a finite word in $(A \times B)^*$ and represents the state of the game after finitely many rounds. We call $\mathcal{H}$ the set of histories. For a (finite or infinite) word w and $k \in \mathbb{N}$, we denote by w_k the k-th letter of w and by $w_{<k}$ the prefix of w of length k. A strategy $s : \mathcal{H} \to A$ for Player 1 is a function that maps histories to actions and represents a behavior for Player 1: in history h, she will play action $s(h)$. Given a strategy s, a history h and a word $\beta \in B^\omega$, we call $\mathrm{out}(h, s, \beta)$ the only play where both players first play h, then Player 1 plays according to s and Player 2 plays the actions of β in order. This play is defined inductively as follows:

- for $k \leq |h|$, $\mathrm{out}(h, s, \beta)_{<k} = h_{<k}$;
- for $k > |h|$, $\mathrm{out}(h, s, \beta)_{<k} = \mathrm{out}(h, s, \beta)_{<k-1}(s(\mathrm{out}(h, s, \beta)_{<k-1}), \beta_{k-|h|-1})$.

We say that a play ρ is *compatible* with a given strategy s if there exists $\beta \in B^\omega$ such that $\rho = \text{out}(\varepsilon, s, \beta)$, where ε is the empty history. Similarly, a history is compatible with s if it is the finite prefix of a compatible play. We say that a strategy is *winning* if all the plays compatible with it belong to W: if Player 1 plays according to such a strategy, she is guaranteed to win. We can extend this concept to say that a strategy s is winning *from a history* h when for all $\beta \in B^\omega$ we have $\text{out}(h, s, \beta) \in W$ (if after history h Player 1 starts playing according to s then she will win). We call a *winning history* a history from which there exists a winning strategy.

We call a *tree* any subset of $(A \times B)^*$ which is closed by prefix, and a branch any (finite or infinite) sequence of elements $e_0 = \varepsilon \sqsubset e_1 \sqsubset e_2 \sqsubset ...$ of a tree. In particular, all histories compatible with a given strategy form a tree, which we call the *strategic tree* induced by the strategy. We makes extensive use of Kőnig's lemma [6], which states that if a finitely branching tree has no infinite branch then it is a finite tree. Specifically, we often use the derived result that if some family in a tree intersects all infinte branches of the tree then it has a finite subset that also does.

Given a history $h \in \Gamma$, we say that an action $a \in A$ is non-losing for h if for any action $b \in B$, $h(a, b)$ is a winning history. Among all histories, we are particularly interested in the set of histories along which Player 1 has only played non-losing actions: we call Γ this particular set. Notice that Player 1 has a winning strategy from any history $h \in \Gamma$, but that playing only non-losing actions for Player 1 might be a losing strategy.

A history $h \in \mathcal{H}$ *induces* a winning set W_h defined as $W_h = \{\rho \in (A \times B)^\omega \mid h\rho \in W\}$. The set W_h contains all the infinite continuations ρ such that $h\rho$ is a winning play.

Recall that we introduced strategies for Player 1 as functions mapping histories to actions in A. Among these strategies, we are particularly interested in those that can be described as finite machines: we call them *finite-memory* strategies. Let us introduce first the concept of *finite-memory decision machines*. A finite-memory decision machine is a tuple (M, σ, μ, m_0) such that:

- M is a finite set (the memory);
- $\sigma : M \to A$ is the decision function;
- $\mu : M \times (A \times B) \to M$ is the memory update function;
- $m_0 \in M$ is the initial memory state.

Given a finite-memory decision machine (M, σ, μ, m_0), we extend μ by defining $\mu(m, h)$ for $h \in \mathcal{H}$ in the following inductive way:

- $\mu(m, \varepsilon) = m$
- for all h in $\mathcal{H}$ and $(a, b) \in A \times B$, $\mu(m, h(a, b)) = \mu(\mu(m, h), (a, b))$.

For readability's sake, in case $m = m_0$ and the context is clear, we write $\mu(h)$ for $\mu(m, h)$.

A finite-memory decision machine (M, σ, μ, m_0) induces a strategy s for Player 1 defined for all $h \in \mathcal{H}$ as $s(h) = \sigma(\mu(m_0, h))$. We say that a strategy s is a finite-memory strategy if it is induced by some finite-memory decision machine, and abusively write $s = (M, \sigma, \mu, m_0)$ (identifying the finite-memory decision machine with the strategy it induces) when it is the case.

We often describe a finite-memory decision machine by only defining μ for the action pairs that are compatible with σ (i.e. for $m \in M$ we only define $\mu(m, (a, b))$ when $a = \sigma(m)$). Such a partial machine can be easily extended to a complete one, and contains all the relevant information to decide on the winning aspect of the strategy (or rather, strategies, as many different extensions are possible) it induces as it describes all plays compatible with itself.

3 Descriptive set theory

3.1 Open sets and the Borel hierarchy

Given a set C and a finite word $w \in C^*$, we call *cylinder* of w the set $\mathrm{cyl}(w) = \{w\rho \mid \rho \in C^\omega\}$. This set contains all the infinite words that start with w. In concordance with the usual cylinder topology on C^ω, the cylinders serve as the basis for the open sets, in the sense that we define as an *open set* any set that can be written as an arbitrary union of cylinders. We say that a family of words $\mathcal{F}$ is a *generating family* for an open set O if we have $O = \cup_{f\in\mathcal{F}}\mathrm{cyl}(f)$, that is, O is the set of all plays that have at least one finite prefix in $\mathcal{F}$.

These open sets allow to define a Borel algebra on C^ω as the smallest σ-algebra that contains all open sets. More precisely, the Borel algebra is the smallest collection of sets that contains the open sets and is closed under both countable union and complement (for more information about Borel sets, see [5]). This collection of sets can be organized into what is called the Borel hierarchy, which is defined for countable ordinals in the following way:

- Σ^0_1 is the collection of all the open sets;
- for all countable ordinals θ, Π^0_θ is the collection of sets whose complements are in Σ^0_θ;
- for all countable ordinals θ, Σ^0_θ is the collection of sets that can be defined as a countable union of sets belonging to lower levels of the hierarchy;
- finally, for all countable ordinals θ, Δ^0_θ is the collection of sets that are in both Σ^0_θ and Π^0_θ.

To illustrate, let us detail the lowest levels of the hierarchy:

- as per the definition, Σ^0_1 is the collection of all the open sets;
- the sets in Π^0_1 are the sets whose complement is an open set, we call them the *closed* sets;
- Σ^0_2 contains the sets which can be written as a countable union of closed sets;
- Π^0_2 contains the sets which complement can be written as a countable union of closed sets: by properties of the complement, these are the sets that can be written as a countable intersection of open sets;
- finally, Δ^0_2 is the collection of sets that can be written both as a countable union of closed sets and as a countable intersection of open sets.

In the following we will focus on the collection of sets Δ^0_2.

3.2 The Hausdorff difference hierarchy

The Hausdorff difference hierarchy (see for instance [5]) provides us with a way of defining inductively all the sets in Δ^0_2. Formally, given an ordinal θ and an increasing sequence of open sets $(O_\eta)_{\eta<\theta}$, the set $D_\theta((O_\eta)_{\eta<\theta})$ is defined by:

$$\rho \in D_\theta((O_\eta)_{\eta<\theta}) \quad \Leftrightarrow \quad \begin{array}{l}\rho \in \cup_{\eta<\theta} O_\eta \text{ and the least } \eta \text{ such that } \rho \in O_\eta \\ \text{has parity opposite to that of } \theta.\end{array}$$

For any ordinal θ, we call $\mathcal{D}_\theta$ the collection of sets S such that there exists an increasing family of open sets $(O_\eta)_{\eta<\theta}$ such that $S = D_\theta((O_\eta)_{\eta<\theta})$. To illustrate, $\mathcal{D}_1$ is the collection of all the open sets, $\mathcal{D}_2$ is the collections of the sets that can be written as $O_1 \setminus O_0$ where O_1 and O_0 are two open sets (and hence contains the closed sets), $\mathcal{D}_3$ is the collection of the sets that can be written as $O_2 \setminus (O_1 \setminus O_0)$ where O_2, O_1 and O_0 are three open sets, etc.

The Hausdorff-Kuratowski theorem [5] then states that a set S belongs to Δ^0_2 if and only if there exists an ordinal θ such that $S \in \mathcal{D}_\theta$.

3.3 The fine Hausdorff hierarchy

In the spirit of the Hausdorff difference hierarchy, we propose another inductive way of defining the sets in Δ_2^0. This other hierarchy was already introduced in [9], and may have appeared earlier in the literature but to our knowledge has never been studied in similar depth. Most of the related results can however be considered folklore. First we introduce the concept of *open union*: we say that the union of a family of sets $(S_i)_{i\in I}$ is an open union if there exists a family of disjoint open sets $(O_i)_{i\in I}$ such that for all $i \in I$ have $S_i \subseteq O_i$. We denote such a union by $\biguplus_{i\in I} S_i$.

We then define inductively collections of sets Λ_θ and K_θ, with θ a positive ordinal, in the following way:

- a set S is in Λ_1 if and only if it is an open set;
- a set S is in K_θ if and only if its complement is in Λ_θ;
- a set S is in Λ_θ with $\theta > 1$ if and only if there exists a family of sets $(S_i)_{i\in I}$ such that for each i there exists $\eta_i < \theta$ such that $S_i \in \Lambda_{\eta_i} \cup K_{\eta_i}$ and we have $S = \biguplus_{i\in I} S_i$.

This definition is akin to the definition of the Borel hierarchy, with the exception that the union operation is replaced with an open union. We then prove the following theorem, which shows our hierarchy is a refinement of the Hausdorff difference hierarchy (and hence justifies its name):

▶ **Theorem 1** (folklore). *For all ordinals θ, we have $\mathcal{D}_\theta = \Lambda_\theta$.*

This theorem is naturally proven by induction on θ and requires intermediate results which help understand the nature of the two hierarchies. In particular, we have: (i) for all ordinals θ the collection $\mathcal{D}_\theta$ is closed under open union, (ii) for all ordinals θ the collection Λ_θ is closed by intersection with an open set, (iii) for all ordinals θ the two collections Λ_θ and K_θ are closed under intersection with a cylinder and (iv) for all limit ordinals θ the collection $\mathcal{D}_\theta$ is the collection of sets that can be written as the open unions of sets in $\cup_{\eta<\theta}\mathcal{D}_\eta$. One observation which proves pivotal for proving our main result on the existence of finite-memory strategies is that for all ordinals θ, all sets in K_θ can be written as the union of a closed set and a set that belongs to Λ_θ (if θ is a successor odinal, we can be even more precise as K_θ is the collection of all sets that can be written as the union of a closed set and a set in $\Lambda_{\theta-1}$). All the details surrounding these two views and the proof of theorem 1 can be found in [3]

3.4 The 0-1 eventually constant labelling

A third possible view of sets in Δ_2^0 is given via eventually constant labelling functions. We say that a labelling function $l : C^* \to \{0, 1\}$ is eventually constant if for all infinite words ρ in C^ω, the sequence $(l(\rho_{<n}))_{n\in\mathcal{N}}$ is eventually constant, which means that there exists a finite $k \in \mathbb{N}$ and $i \in \{0, 1\}$ such that for all $n \geq k$ we have $l(\rho_{<n}) = i$. We then call 1_l the set of infinite words $\rho \in C$ such that the set $\{n \mid l(\rho_{<n}) = 1\}$ is infinite. As we will see, the sets S that belong to Δ_2^0 are the sets such that there exists an eventually constant labelling function l such that $S = 1_l$.

3.5 Equivalence of representations

3.5.1 Representations of sets in Δ_2^0

As expressed by the following theorem, the Hausdorff difference hierarchy, fine Hausdorff hierarchy and eventually constant labelling functions actually define the same sets, which are exactly the sets that belong to Δ_2^0.

▶ **Theorem 2** (folklore). *Given a subset S of C^ω, the following propositions are equivalent:*
(1) *$S \in \Delta_2^0$;*
(2) *S belongs to the Hausdorff difference hierarchy;*
(3) *S belongs to the fine Hausdorff hierarchy;*
(4) *there exists an eventually constant labelling function l such that $S = 1_l$.*

The detailed proof can be found in [3], but we give some elements here: the Hausdorff-Kuratowski theorem [5] shows that (1) ⇔ (2), and Theorem 1 shows that (2) ⇔ (3). We prove that (3) ⇒ (4) by showing that the collection of sets of the form 1_l where l is an eventually constant labelling function is closed under both open union and complement, and finally prove that (4) ⇒ (1) by showing that all sets of the form 1_l where l is an eventually constant labelling function can be expressed both as countable intersection of open sets and countable union of closed sets.

3.5.2 Correspondence between Büchi/co-Büchi conditions and Π_2^0/Σ_2^0

Two much studied types of winning condition in computer science are the *Büchi* and *co-Büchi* conditions. Such winning conditions are given by a coloring function c that provides a color (elements in $\{0, 1\}$) for every history. In the case of a Büchi condition, a play is then winning if infinitely many of its prefixes are associated with the color 1 while in the case of a co-Büchi condition it is winning if finitely many of its prefixes are associated with the color 1 (Büchi and co-Büchi conditions are thus the complement of each other). As stated by the following lemma, whose proof can be found in [3], Büchi conditions actually describe the sets in Π_2^0:

▶ **Lemma 3.** *A subset S of C^ω belongs to Π_2^0 if and only if it can be expressed as a Büchi condition.*

A trivial corollary is that co-Büchi conditions describe the sets in Σ_2^0:

▶ **Corollary 4.** *W belongs to Σ_2^0 if and only if it can be expressed as a co-Büchi condition.*

4 On the existence of finite-memory winning strategies when the winning set belongs to the Hausdorff difference hierarchy

Our aim is to exhibit conditions on W that ensure Player 1 has a finite-memory winning strategy when some winning strategy exists.

Consider a game (A, B, W) where the winning set W belongs to Δ_2^0. We introduce a new hypothesis on the induced winning sets of this game: the set inclusion relation, denoted by $\subseteq$, induces a well partial order (wpo) on the winning sets induced by the histories in Γ. That is, for any sequence $(h_n)_{n\in\mathbb{N}}$ of histories in Γ, there exists $k < l$ such that $W_{h_k} \subseteq W_{h_l}$. A known property of well partial orders which we will use is that any set $S \subseteq \Gamma$ contains a finite subset M such that for all $h \in S$ there exists $m \in M$ such that $W_m \subseteq W_h$. The set of winning sets induced by the histories in M effectively functions as a finite set of under-approximations for the winning sets induced by the histories in S.

Such hypotheses might seem exotic and restrictive, but are effectively satisfied for well-studied classes of games, such as energy games or multi-energy games played on graphs (see for instance [10]), or games with a winning condition expressed as a boolean combination of reachability/safety conditions. Indeed, in the first case the induced winning sets are isomorphic to the cartesian product of the state space and $\mathbb{N}^k$, where k is the number of energy dimensions, while in the second case they are isomorphic to the cartesian product of the state space and the set of possible valuations.

Under these specific conditions, we prove that Player 1 always has a finite-memory winning strategy when she has a winning strategy:

▶ **Theorem 5.** *Assume that W belongs to Δ_2^0 and $\subseteq$ induces a well partial order on $\{W_h \mid h \in \Gamma\}$. If Player 1 has a winning strategy from ε, then she also has a finite-memory one.*

Given a set S in the Hausdorff difference hierarchy, the rank of S is the least ordinal θ such that $S \in \mathcal{D}_\theta$. We prove Theorem 5 by a transfinite induction on the rank of W.

First notice that the inclusion of induced winning sets has the nice property of being preserved by the addition of a suffix, which is formally expressed by the following lemma:

▶ **Lemma 6.** *If $W_h \subseteq W_{h'}$ then for all $(a, b) \in A \times B$ we have $W_{h(a,b)} \subseteq W_{h'(a,b)}$.*

▶ **Corollary 7.** *If $W_h \subseteq W_{h'}$ then all non-losing actions of h are also non-losing for h'.*

4.1 Proof for open sets

We begin by the case where W is an open set (W has rank 1), generated by a set $\mathcal{F}$ of histories.

▶ **Lemma 8.** *If W is an open set and $\varepsilon \in \Gamma$, then Player 1 has a finite-memory winning strategy from ε.*

Proof. Suppose that there exists a winning strategy s from ε. Then consider the strategic tree T induced by s, and consider the tree $T' = T \setminus \{h \in \mathcal{H} \mid \exists f \in \mathcal{F}, f \sqsubset h\}$, where $\sqsubset$ is the strict prefix relation. Since s is winning, there is no infinite branch in T'. By Kőnig's lemma, this means that T' is finite and by definition all maximal elements (with regards to $\sqsubseteq$, the prefix relation) of T' belong to $\mathcal{F}$. T' can then serve as the memory of a finite-memory winning strategy $s_f = (T', \sigma, \mu, m_0)$ defined by:

- for $t \in T' \setminus \mathcal{F}$, $\sigma(t) = s(t)$;
- for $t \in T' \cap \mathcal{F}$, we set $\sigma(t)$ as any action $a \in A$;
- for $t \in T'$ and $b \in B$, $\mu(t, (\sigma(t), b)) = t(\sigma(t), b)$ if $t \notin \mathcal{F}$ and $\mu(t, (\sigma(t), b)) = t$ if $t \in \mathcal{F}$;
- $m_0 = \varepsilon$.

The strategy s_f works by simply following s alongside the branches of T' until it reaches a history in $\mathcal{F}$, and is thus winning. ◀

4.2 Proof for closed sets

We now focus on the study of the case where the winning set W is a closed set. In that case, the plays ρ such that $\rho \in W$ are precisely the plays for which all finite prefixes h are such that $W_h \neq \emptyset$. As a consequence, any strategy playing non-losing actions for Player 1 is a winning strategy: such a strategy only generates histories h in Γ, and in particular $W_h \neq \emptyset$.

Furthermore, if $\subseteq$ induces a well partial order on the partial winning sets, then:

($*$) there exists a finite subset M of Γ such that for all $h \in \Gamma$ there exists $m \in M$ such that $W_m \subseteq W_h$;

($**$) any play ρ has two finite prefixes ρ_0 and ρ_1 such that $\rho_0 \sqsubset \rho_1$ and $W_{\rho_0} \subseteq W_{\rho_1}$.

These two observations are the basis for two different approaches to prove the next lemma.

▶ **Lemma 9.** *If W is a closed set, if $\subseteq$ induces a well partial order on $(W_h)_{h \in \Gamma}$ and if $\varepsilon \in \Gamma$, then Player 1 has a finite-memory winning strategy from ε.*

Proofs using the two approaches. Consider indeed a game (A, B, W) where W is a closed set, $\subseteq$ induces a well partial order on the partial winning sets associated with the winning histories and such that ε is a winning history. We consider a strategy s that is a winning strategy for Player 1.

The first approach, derived from observation $(*)$, consists in building a finite-memory strategy (M, σ, μ, m_0) with memory set M in the following way:

- for $m \in M$, we let $\sigma(m)$ be any non-losing action from m,
- for $m \in M$ and $b \in B$, since $\sigma(m)$ is non-losing we know that $m(\sigma(m), b) \in \Gamma$, which means that there exists $m' \in M$ such that $W_{m'} \subseteq W_{m(\sigma(m),b)}$; we then let $\mu(m, (\sigma(m), b)) = m'$;
- finally $m_0 \in M$ is chosen such that we have $W_{m_0} \subseteq W_\varepsilon$.

Informally, we have as our memory the set M which contains under-approximations for all winning sets induced by the histories of Γ. We use the transition function to maintain an under-approximation of the "real" induced winning set associated to the current history, and play according to this under-approximation. By Corollary 7, this ensures that we always play a non-losing action, which is enough to guarantee the win because W is a closed set.

The second approach is derived from observation $(**)$. Consider the winning strategy s and its associated strategic tree T_s. Along every infinite branch ρ of T_s, there exist two histories h, h' such that $h \sqsubset h'$ and $W_h \subseteq W_{h'}$. Consider then the tree T_s^f obtained by pruning T_s along these histories: $T_s^f = \{h' \in T_s \mid \forall h \in T_s, h \sqsubset h' \Rightarrow W_h \not\subseteq W_{h'}\}$. By Kőnig's lemma, T_s^f is a finite tree. We call $\mathcal{P}$ the set $\{h \in T_s \mid h \notin T_s^f \text{ and } h' \sqsubset h \Rightarrow h' \in T_s^f\}$ of the minimal elements (with regards to the prefix relation) of T_s that do not belong to T_s^f. We then build a finite-memory strategy (M', σ, μ, m_0) in the following way:

- $M' = T_s^f$
- for $m \in M'$, we let $\sigma(m) = s(m)$,
- for $m \in M'$ and $(a, b) \in A \times B$ such that $a = \sigma(m) = s(m)$,
  - if $m(a, b) \in T_s^f$ then we let $\mu(m, (a, b)) = m(a, b)$,
  - else by construction we have $m(a, b) \in \mathcal{P}$ and there exists $m' \in T_s^f$ such that $m' \sqsubset m(a, b)$ and $W_m \subseteq W_{m(a,b)}$; we then let $\mu(m, (a, b)) = m'$,
- finally $m_0 = \varepsilon$.

Informally, this approach consists in playing according to s until we reach a history whose induced winning set is bigger than one we already met. We then forget the current history and continue playing as if we were in the history with the smaller induced winning set. This second approach also ensures that the memory consists of an under-approximation of the "real" induced winning set, and hence by Corollary 7 it guarantees that the resulting strategy is non-losing, and thus winning since W is a closed set. ◀

4.3 Limitations to the above approaches

Until now, we have studied the lowest levels of the Hausdorff difference hierarchy, focusing on the cases where the winning sets belongs to Λ_1, the open sets, and K_1, the closed sets. We will explain later how to handle the case for Λ_2 and for now turn our attention to K_2, as it proves pivotal to the understanding of our method.

The sets in K_2 are the sets that can be written as the union of a closed set and an open set. Informally, this means that Player 1 can win in two different ways, by ensuring that either the generated play lies in the closed set or they reach a history which belongs to the generating family of the open set.

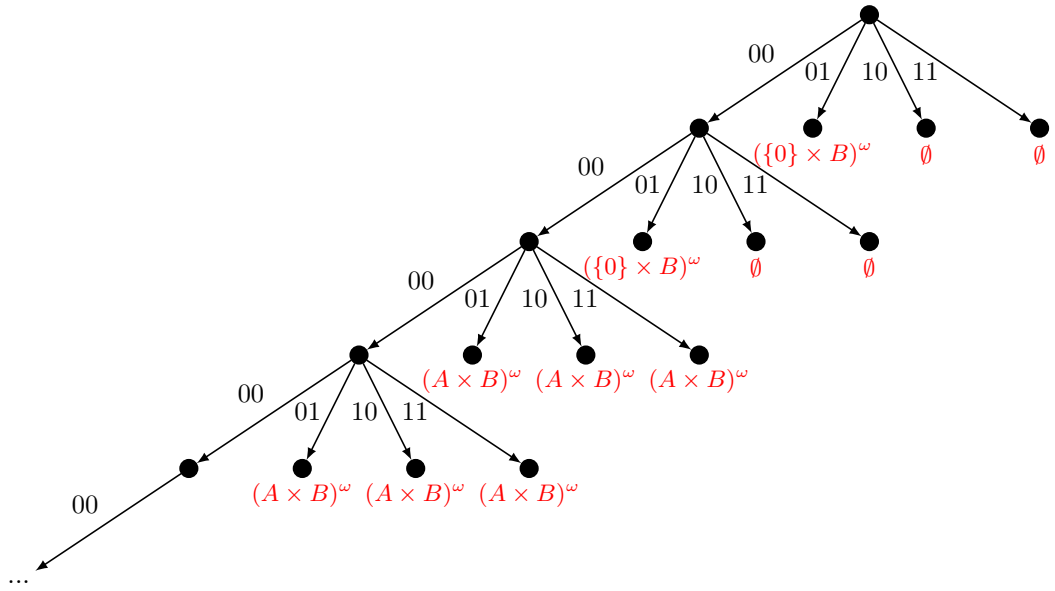


Figure 1 A game for which the naive algorithm does not work. For the sake of concision, the pairs of actions in $A \times B$ are written as two-letter words. In red is the partial winning set of the corresponding history.

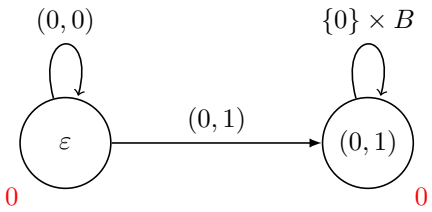


Figure 2 The finite-memory strategy generated by approaches $(*)$ and $(**)$ for the game represented in Figure 1. Each state is labeled by the history associated to it, and in red is the action associated to that state.

The first condition is akin to a safety objective (Player 1 manages to never go out of a certain region) while the second condition is akin to a reachability objective (Player 1 meets a certain given condition at a finite time and it suffices to ensure the win). As shown by the following example, the two simple approaches we detailed previously for closed sets do not suffice here:

▶ **Example 10.** Consider the game (A, B, W) with $A = B = \{0, 1\}$ and $W = (0,0)^*(0,1)(\{0\} \times B)^\omega + (0,0)^2(0,0)^*(\{1\} \times B + A \times \{1\})(A \times B)^\omega$. This game is described in Figure 1. In other words, Player 1 has two ways to win:

- either the players play $(0,1)$ or $(0,0)(0,1)$ and Player 1 then only has to play action 0 forever;
- or the players play $(0,0)$ twice and then one player plays action 1, reaching a point where all possible continuations are winning for Player 1.

As W can be expressed via a regular expression, it induces finitely many partial winning sets, which means that $\subseteq$ trivially induces a well partial order over said partial winning sets. Moreover, W can be expressed as the union of an open set and a closed set (the closed set corresponds to the first item above, while the open set corresponds to the second item), and hence belongs to the Hausdorff difference hierarchy (more precisely it belongs to K_2).

Moreover, one can easily check that $W_\varepsilon \subseteq W_{(0,0)}$. As a consequence, both approach $(*)$ and approach $(**)$ yield the finite-memory strategy described in Figure 2. This strategy is not winning for Player 1, as if Player 2 always plays action 0 it will generate the play $(0,0)^\omega$, which does not belong to W.

4.4 Proofs for sets in K_2

To better understand how the proof works in the general case, we propose here to study the basic case of sets in K_2. As we have seen previously, the two approaches that worked well for the case where the winning set is a closed set do not suffice in that case. Nevertheless, we prove the following result:

▶ **Theorem 11.** *If W is in K_2, if $\subseteq$ induces a well partial order on $(W_h)_{h\in\Gamma}$ and if $\varepsilon \in \Gamma$, then Player 1 has a finite-memory winning strategy from ε.*

Let us suppose then that W is in K_2: as already mentioned, W is the union of a closed set C and an open set O. We let $\mathcal{F}$ be the generating family of O and denote by $\mathrm{Pref}(C)$ the set of histories which have at least one continuation in C, that is $\mathrm{Pref}(C) = \{h \in \mathcal{H} \mid \exists \rho, h\rho \in C\}$.

As a preliminary observation, recall we already know how to handle the case when the winning set is open. The method also works well for the general case when Player 1 is able to reach O by herself (that is, she have a winning strategy for O). We also know that finding a finite-memory non-losing strategy for Player 1 is always possible (see for instance the method $(*)$ for the case where the winning set is closed). As a consequence, a simple method one would be tempted to try would be the following:

- follow some non-losing strategy as long as the current history belongs to $\mathrm{Pref}(C)$;
- as soon as we detect we have left $\mathrm{Pref}(C)$, play some finite-memory winning strategy to reach a history in $\mathcal{F}$ (this is possible because if we have played in a non-losing fashion so far and the current history does not belong to $\mathrm{Pref}(C)$, then the only way to win from there is to produce a play that belongs to O).

This method should produce a finite-memory winning strategy, however it relies on the assumption that one is able to detect whether or not the current history belongs to $\mathrm{Pref}(C)$ using only finite memory. This assumption does not rely on any solid ground, which makes this method incorrect. We propose another construction of a finite-memory winning strategy, which does not need to detect when the current history stops belonging to $\mathrm{Pref}(C)$, but which ensures that $\mathcal{F}$ will be reached if it were the case (despite not knowing it).

Consider indeed a history $\overline{h}$ in $\Gamma \cap \mathrm{Pref}(C)$ and a history h in Γ such that $h \notin \mathrm{Pref}(C)$ and $W_{\overline{h}} \subseteq W_h$. We call $TC(\overline{h}, h)$ the set $\{hl \mid \overline{h}l \in \mathrm{Pref}(C)\}$ consisting of the finite continuations from $\overline{h}$ that belong to $\mathrm{Pref}(C)$, but rooted in h. Notice that for all infinite continuations ρ such that $\overline{h}\rho \in C$, we have $\overline{h}\rho \in W$, which means that $h\rho \in W$ (since $W_{\overline{h}} \subseteq W_h$) and hence that $h\rho \in O$ since $h \notin \mathrm{Pref}(C)$ (which means that all winning continuations of h belong to O because they cannot belong to C). We thus know that $TC(\overline{h}, h)$ contains a family of histories $\mathcal{F}_{\overline{h},h}$ included in $\mathcal{F}$ (or in the case where h itself has a strict prefix in $\mathcal{F}$, we set $\mathcal{F}_{\overline{h},h} = \{h\}$) and such that all infinite branches of $TC(\overline{h}, h)$ have a finite prefix in $\mathcal{F}_{\overline{h},h}$, and by Kőnig's lemma we know this family is finite. We call $\mathrm{depth}(\overline{h}, h)$ the maximal length of the elements in $\mathcal{F}_{\overline{h},h}$. Intuitively, this means that, if the current history were h but Player 1 only knew of its under-approximation $\overline{h}$, she could ensure the win by following a play whose finite prefixes l were such that $\overline{h}l \in \mathrm{Pref}(C)$ for $\mathrm{depth}(\overline{h}, h)$ steps. This however still requires to compute the value of $\mathrm{depth}(\overline{h}, h)$ and hence to know of h. However, as stated by the following lemma, the value of $\mathrm{depth}(\overline{h}, h)$ for all eligible h is bounded.

▶ **Lemma 12.** *For all* $\overline{h}$ *in* $\mathrm{Pref}(C)$, $\{\mathit{depth}(\overline{h}, h) \mid h \in \Gamma, h \notin \mathrm{Pref}(C), W_{\overline{h}} \subseteq W_h\}$ *is bounded.*

The proof of this lemma can be found in appendix [3], and makes use of the well partial order hypothesis.

For $\overline{h}$ in $\Gamma \cap \mathrm{Pref}(C)$, we will then call $\mathrm{depth}(\overline{h})$ the upper bound of $\mathrm{depth}(\overline{h}, h)$ for h meeting the criteria described above. The idea is the following: if the current history is $h \in \Gamma$, but Player 1 only knows of its under-approximation $\overline{h}$, and then plays some finite continuation l of length $\mathrm{depth}(\overline{h})$ (which is independent of h) such that $\overline{h}l \in \mathrm{Pref}(C)$, then she ensured the win as hl has a finite prefix in $\mathcal{F}$. This is formally stated in the following lemma:

▶ **Lemma 13.** *Let* $\overline{h} \in \mathrm{Pref}(C)$ *and* $h \notin \mathrm{Pref}(C)$ *such that* $W_{\overline{h}} \subseteq W_h$. *Let* $\rho \in (A \times B)^\omega$ *such that for all finite prefixes* l *of* ρ *such that* $|l| \leq \mathit{depth}(\overline{h})$ *we have* $\overline{h}l \in \mathrm{Pref}(C)$. *Then* $h\rho \in O$.

Proof. Let l be the finite prefix of ρ of length $\mathrm{depth}(\overline{h})$. As $\mathrm{depth}(\overline{h}, h) \leq \mathrm{depth}(\overline{h})$ we know that hl has a prefix in $\mathcal{F}$, hence the result. ◀

Consider now a finite family $(h_i)_{i \in I}$ of histories in $\Gamma \setminus \mathrm{Pref}(C)$ such that for all $h \in \mathrm{Pref}(C)$ there exists $i \in I$ such that $W_{h_i} \subseteq W_h$. For all $i \in I$ there exists a finite-memory decision machine $(M_i, \sigma_i, \mu_i, m_i)$ associated with a of finite-memory strategy $(s_i)_{i \in I}$ such that s_i wins from h_i. As a consequence, for all $h \in \Gamma \setminus \mathrm{Pref}(C)$ there exists $i \in I$ such that s_i wins from h. Consider also a finite family $(\overline{h}_j)_{j \in J}$ of histories in $\Gamma \cap \mathrm{Pref}(C)$ indexed by $J \subseteq \mathbb{N}$ such that for all histories $\overline{h}$ in $\mathrm{Pref}(C) \cap \Gamma$ there exists $j \in J$ such that $W_{\overline{h}_j} \subseteq W_{\overline{h}}$. For all $j \in J$, let $T_j = \{\overline{h}_j l \mid |l| \leq \mathrm{depth}(\overline{h}_j)\}$. Up to renaming, we can suppose that the T_j's are disjoint from one another. We build our finite-memory winning strategy $s = (M, \sigma, \mu, m_0)$ in the following way:

- $M = \cup_{i \in I} M_i \cup \cup_{j \in J} T_j$;
- for $m \in M_i$ we let $\sigma(m) = \sigma_i(m)$;
- for $t \in T_j$ we let $\sigma(t)$ be any non-losing action from t;
- for $m \in M_i$ and $(a, b) \in A \times B$ we let $\mu(m, (a, b)) = \mu_i(m, (a, b))$;
- for $t \in T_j$ and $(a, b) \in A \times B$ such that $a = \sigma(t)$:
  - if $t(a, b) \in T_j$ then $\mu(t, (a, b)) = t(a, b)$;
  - else if $t(a, b) \in \Gamma \setminus \mathrm{Pref}(C)$ then there exists $i \in I$ such that s_i wins from $t(a, b)$: we let $\mu(t, (a, b)) = m_i$;
  - else if $t(a, b) \in \Gamma \cap \mathrm{Pref}(C)$ then there exists $j \in J$ such that $W_{\overline{h}_j} \subseteq W_{t(a,b)}$ and we let $\mu(h, (a, b)) = \overline{h}_j$;
- $m_0 = \overline{h}_j$ where $j \in J$ is such that $W_{\overline{h}_j} \subseteq W_\varepsilon$.

We prove this finite-memory strategy is winning for Player 1. To this end, let us first show that for all compatible histories h such that $\mu(h) \in T_j$ for some $j \in J$, the memory state $\mu(h)$ provides an under-approximation of the winning set induced by h:

▶ **Lemma 14.** *If* $\mu(h) \in T_j$ *for some* $j \in J$ *then we have* $W_{\mu(h)} \subseteq W_h$.

Proof. The proof is by induction on h. First we have $\mu(\varepsilon) = m_0$ and per the definition $W_{m_0} \subseteq W_\varepsilon$. Now consider $h \in \mathcal{H}$ such that $\mu(h) \in T_j$ for some $j \in J$ and $W_{\mu(h)} \subseteq W_h$. Let $(a, b) \in A \times B$ such that $\mu(h(a, b)) \in T_{j'}$ for some $j' \in J$. Then,

- if $\mu(h)(a, b) \in T_j$ then we have $\mu(h(a, b)) = \mu(h)(a, b)$ and the desired result follows by Lemma 6,

- else we must have $\mu(h)(a,b) \in \Gamma \cap \mathrm{Pref}(C)$ (else we would not have $\mu(h(a,b)) \in T_{j'}$) and $\mu(h(a,b)) = \overline{h}_{j'}$ with j' such that $W_{\overline{h}_{j'}} \subseteq W_{\mu(h)(a,b)}$, and $W_{\mu(h)(a,b)} \subseteq W_{h(a,b)}$ once again by Lemma 6, which ensures the result. ◀

As a consequence of Lemma 14, when h is such that $\mu(h) \in T_j$ for some $j \in J$ then we have $W_{\mu(h)} \subseteq W_h$. As a consequence, for all $(a,b) \in A \times B$ we have $W_{\mu(h)(a,b)} \subseteq W_{h(a,b)}$. This ensures that if $\mu(h)(a,b) \in \Gamma \setminus \mathrm{Pref}(C)$ and the strategy s_i wins from $\mu(h)(a,b)$ then s_i also wins from $h(a,b)$, which explains why our finite-memory strategy is winning. Formally, we have the following lemma:

▶ **Lemma 15.** *If $\rho \in (A \times B)^\omega$ is compatible with s and there exists $k \in \mathbb{N}$ and $i \in I$ such that $\mu(\rho_{\leq k}) = m_i$ then $\rho \in W$.*

Proof. Without loss of generality, we can suppose that k is the smallest integer such that $\mu(\rho_{\leq k}) \notin \cup_{j \in J} T_j$. We want to prove there exists a history h such that $W_h \subseteq W_{\rho_{\leq k}}$ and s_i is winning from h, as this will yield the desired result. Our candidate for h is $\mu(\rho_{\leq k-1})(a,b)$ where $(a,b) = \rho_k$.

- By Lemma 14 we know that $W_{\mu(\rho_{\leq k-1})} \subseteq W_{\rho_{\leq k-1}}$, and thus $W_{\mu(\rho_{\leq k-1})(a,b)} \subseteq W_{\rho_{\leq k-1}}$.
- Furthermore, we know per the definition of s that s_i is winning from $\mu(\rho_{\leq k-1}(a,b))$ since $\mu(\rho_{\leq k}) = m_i$.

Finally, a trivial induction shows that for any $\beta \in B^\omega$ we have $\mathrm{out}(\rho_{\leq k}, s, \beta) = \mathrm{out}(\rho_{\leq k}, s_i, \beta)$, and since s_i is winning from $\rho_{\leq k}$ we have $\rho \in W$. ◀

Finally, with the help of Lemma 14 and Lemma 15, we can prove the following, which concludes the proof of Theorem 11.

▶ **Lemma 16.** *s is winning from ε.*

Proof. Let us consider $\rho \in (A \times B)^\omega$ compatible with s. We want to show that $\rho \in W$. If there exists $k \in \mathbb{N}$ such that $\mu(\rho_{\leq k}) \in \cup_{i \in I} M_i$ then Lemma 15 suffices to conclude. Suppose then that for all $k \in \mathbb{N}$ we have $\mu(\rho_{\leq k}) \in \cup_{j \in J} T_j$. If $\rho \in C$ then obviously we have $\rho \in W$, so let us suppose that there exists $n_0 \in \mathbb{N}$ such that $\rho_{\leq n_0} \notin \mathrm{Pref}(C)$. Due to the construction of s, there exists some $n_1 \geq n_0$ and some $j \in J$ such that $\mu(\rho_{\leq n_1}) = \overline{h}_j$. By Lemma 14 we then have $W_{\mu(\rho_{\leq n_1})} \subseteq W_{\rho_{\leq n_1}}$. Notice that we also have $\rho_{\leq n_1} \notin \mathrm{Pref}(C)$. Finally, let l be the prefix of $\rho_{>n_1}$ of length $\mathrm{depth}(\overline{h}_j)$. By construction and since we do not have $\mu(\rho_{\leq k}) \in \cup_{i \in I} M_i$ for any k, for all $k \leq \mathrm{depth}(\overline{h}_j)$ we have $\mu(\rho_{\leq n_1 + k}) = \overline{h}_j l_{\leq k} \in \mathrm{Pref}(C)$, and hence by application of Lemma 13 we can conclude that $\rho \in O$ and thus $\rho \in W$. ◀

We illustrate our method on the game described in Figure 1:

▶ **Example 17.** Consider once again the game represented in 1. We recall here that we have $A = B = \{0,1\}$ and $W = (0,0)^*(0,1)(\{0\} \times B)^\omega + (0,0)^2(0,0)^*(\{1\} \times B + A \times \{1\})(A \times B)^\omega$. We let $C = (0,1)(\{0\} \times B)^\omega + (0,0)(0,1)(\{0\} \times B)^\omega$ and $O = (0,0)^2(0,0)^*[(0,1) + (1,0) + (1,1)](A \times B)^\omega$, and we have $W = C \cup O$. One can check easily that C is a closed set and O is an open set generated by the family of histories $\mathcal{F} = (0,0)^2(0,0)^*[(0,1) + (1,0) + (1,1)]$.

The histories in $\mathrm{Pref}(C)$ are of three different types: ε, whose induced winning set is W, $(0,0)$, whose induced winning set is included in W, and all the other histories in $\mathrm{Pref}(C)$, whose induced winning set is $(\{0\} \times B)^\omega$. We choose two histories in $\mathrm{Pref}(C)$ that provide under-approximations for the open sets induced by all elements of $\mathrm{Pref}(C)$: ε and $(0,1)$. Except for histories which already have a prefix in $\mathcal{F}$, no history out of $\mathrm{Pref}(C)$ induces a winning set that includes $W_{(0,1)}$, which means that $\mathrm{depth}((0,1)) = 0$. However W_ε is included in all winning sets induced by the histories in $(0,0)^2(0,0)^*$. Since $(0,0)^2(0,0)^*(0,0)(0,1) \subseteq \mathcal{F}$,

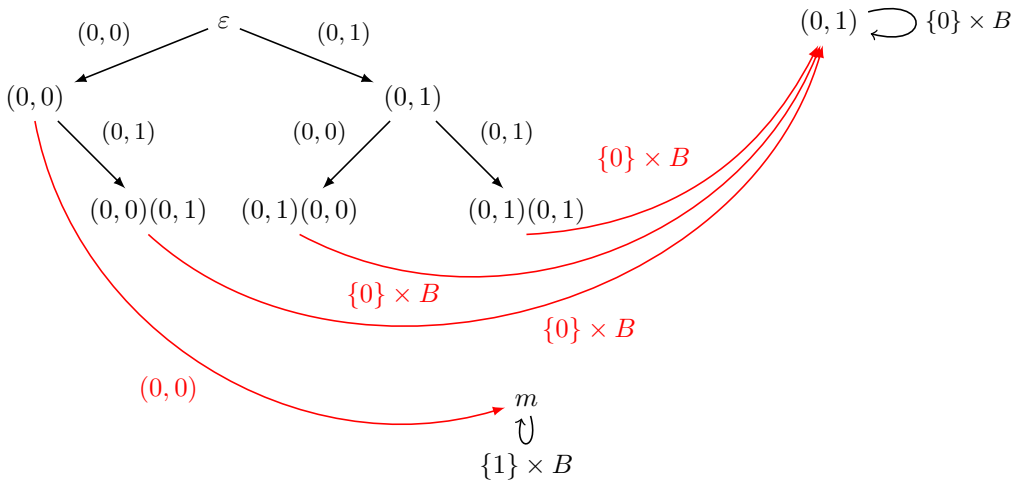


Figure 3 The finite-memory strategy for the game represented in Figure 1.

we have $\text{depth}(\varepsilon) = 2$. The only continuation of length 2 from ε that goes out of $\text{Pref}(C)$ is $(0,0)(0,0)$, and the finite-memory strategy we chose that reaches O from $(0,0)(0,0)$ is one where Player 1 always plays action 1.

4.5 Proof for sets in Λ_θ

We study now the case where $W \in \Lambda_\theta$ for an ordinal $\theta > 1$ and Theorem 5 is true for all winning sets belonging to Λ_η or K_η for all $\eta < \theta$. As always, we suppose that Player 1 has a winning strategy from ε and we want to show she has a finite-memory winning strategy.

As $W \in \Lambda_\theta$, there exists a family of sets $(S_i)_{i\in I}$ such that $W = \Cup_{i\in I} S_i$ and for each i there exists $\theta_i < \theta$ such that $S_i \in \Lambda_{\theta_i}$ or $S_i \in K_{\theta_i}$. Furthermore, there exists a disjoint family of open sets $(O_i)_{i\in I}$ such that for each $i \in I$ we have $S_i \subseteq O_i$. Finally, for each i the set O_i is generated by a family of histories $\mathcal{F}_i$.

Consider then a winning strategy s for Player 1 and let T_s be its induced strategic tree. Consider the tree $T'_s = T_s \setminus \{h \in \mathcal{H} \mid \exists i \in I, \exists f \in \mathcal{F}_i, f \sqsubset h\}$. Since s is winning, all infinite branches of T_s belongs to O_i for some i, and hence T'_s does not have any infinite branch. By Kőnig's lemma, this means that T'_s is a finite tree. Let us consider the set $\mathcal{L}$ of maximal elements (with regards to the prefix relation) in T'_s. By construction all histories h in $\mathcal{L}$ are such that there exists a unique (because the O_i's are disjoint from one another) $i \in I$ such that $h \in \mathcal{F}_i$. This means that the winning plays which have h as a prefix are included in S_i and hence for $h \in \mathcal{L}$ we have $W_h = h^{-1}(S_i \cap \text{cyl}(h))$. Since both Λ_{θ_i} and K_{θ_i} are closed under intersection with a cylinder (see [3]) we know that $W_h \in \Lambda_{\theta_i}$ or $W_h \in K_{\theta_i}$ (depending on whether S_i belongs to Λ_{θ_i} or K_{θ_i}). Moreover, since all histories in $\mathcal{L}$ belong to T_s and s is winning we know that $\mathcal{L} \subseteq \Gamma$. This means that we have all the hypotheses we need to apply the induction hypothesis to any $l \in \mathcal{L}$ (namely, $l \in \Gamma$ and $W_l \in \Lambda_\eta$ or $W_l \in K_\eta$ for some $\eta < \theta$), and hence for all $l \in \mathcal{L}$ there exists a finite-memory strategy $s_l = (M_l, \sigma_l, \mu_l, m_l)$ that wins from l (up to renaming, we suppose that the M_l's are disjoint from one another and from T_s).

We will build a finite-memory strategy $s_f = (M, \sigma, \mu, m_0)$ for Player 1 in the following way:

- $M = T'_s \cup \uplus_{l \in \mathcal{L}} M_l$;
- $\sigma(m) = s(m)$ for $m \in T'_s$, and $\sigma(m) = \sigma_l(m)$ for $m \in M_l$;
- $\mu(m, (a, b)) = m(a, b)$ if $m \in T'_s$ and $m(a, b) \neq l$ for all $l \in \mathcal{L}$ (where $a = \sigma(m)$);
- $\mu(m, (a, b)) = m_l$ if $m \in T'_s$ and $m(a, b) = l$ for some $l \in \mathcal{L}$ (where $a = \sigma(m)$);
- $\mu(m, (a, b)) = \mu_l(m(a, b))$ if $m \in M_l$;
- $m_0 = \varepsilon$ if $\varepsilon \neq l$ for all $l \in \mathcal{L}$, and $m_0 = m_l$ if there exists $l \in \mathcal{L}$ such that $l = \varepsilon$.

The idea behind this construction is the following: we follow the winning strategy s until we reach some $l \in \mathcal{L}$, from which we know the strategy s_l is winning. It remains only to emulate s_l from this point onwards to guarantee the win. The formal proof that this strategy is winning can be found in [3].

4.6 Proof for sets in K_θ

The full induction step for K_θ is more complex than for K_2, in term of overall structure as well as detail-level technicalities. Especially, it involves nested inductions on the Hausdorff difference hierarchy. We do not present it here, but provide a detailed proof in appendix [3].

5 Tightness of the result

We explore the tightness of our result. In particular, the winning sets of the two games in Example 18 below are in Π^0_2 and Σ^0_2, respectively, just above Δ^0_2 in the Borel hierarchy; the two games satisfy the well partial order assumption and Player 1 has winning strategies, but no finite-memory winning strategies. Note that the example in Π^0_2 is harder to define and deal with than the one in Σ^0_2. (See details in [3].)

▶ **Example 18.** For the counter-example in Π^0_2, let A be a finite set of at least two elements and w a disjunctive sequence on $A \times \{0\}$. We define the labeling function $l : (A \times \{0\})^* \to \{0, 1\}$ in the following way: $l(h) = 1$ if and only if there exists h_0 and h' such that $h = h_0 h'$ and h' is the longest factor of h that is also a prefix of w. Let then W be the set defined by $\rho \in W$ if and only if infinitely many prefixes h of ρ are such that $l(h) = 1$ and consider the game $(A, \{0\}, W)$.

For the counter-example in Σ^0_2, let A be a finite set of at least two elements and w an irregular word (i.e. a word with infinitely many different suffixes) in $(A \times \{0\})^\omega$. Let $W = \{\rho \in (A \times \{0\})^\omega \mid \exists h \in (A \times \{0\})^*, \rho = h\rho_0$ where ρ_0 is a suffix of $w\}$. W is the set of sequences which have a suffix in common with w. Consider then the game $(A, \{0\}, W)$.

We also considered a relaxation of the well partial order assumption, but found a counter-example with a closed winning set. (See details in [3].)

Finally, we studied the following statement "given a two-player game (A, B, W) where W belongs to the Hausdorff difference hierarchy and such that $\subseteq$ induces a well partial order on the induced winning sets for Player 1, if Player 2 has a winning strategy for ε then is it the case that he also has a finite-memory winning strategy?". When W is a closed set (and hence its complement an open set), the answer is obviously yes, but we found a counter-example where $W \in \Lambda_2$.

6 Conclusion

To conclude, we have proven the existence of finite-memory winning strategies under certain conditions on the winning set for Player 1. These conditions are met for well studied games such as energy games [10] or games where the winning condition is a Boolean combination of

reachability and safety objectives, which makes our result a generalization of known results on the topic. This result relies on descriptive set theory, in particular representation of sets in Δ_2^0.

We have also studied the tightness of our result. The results we currently have in this direction encourage us to think that our hypotheses are tight and that weakening them is no easy task. In the future, we want to extend these tightness results by exploring other possible hypotheses, as well as study infinitely branching games, when the action sets of the players are not finite.

References

1 Benjamin Aminof and Sasha Rubin. First-cycle games. *Inf. Comput.*, 254:195–216, 2017. doi:10.1016/j.ic.2016.10.008.

2 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, pages 24:1–24:22, 2020. doi:10.4230/LIPIcs.CONCUR.2020.24.

3 Patricia Bouyer, Stéphane Le Roux, and Nathan Thomasset. Finite-memory strategies in two-player infinite games. *CoRR*, abs/2107.09945, 2021. arXiv:2107.09945.

4 John Fearnley and Martin Zimmermann. Playing Muller games in a hurry. *Int. J. Found. Comput. Sci.*, 23(3):649–668, 2012. doi:10.1142/S0129054112400321.

5 Alexander Kechris. *Classical Descriptive Set Theory*. Springer, New York, NY, 1995. doi:10.1007/978-1-4612-4190-4.

6 Dénes Kőnig. Über eine Schlussweise aus dem Endlichen ins Unendliche (in german). *Acta Sci. Math. (Szeged)*, 1927.

7 Stéphane Le Roux, Arno Pauly, and Mickael Randour. Extending finite-memory determinacy by boolean combination of winning conditions. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, pages 38:1–38:20, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.38.

8 Robert McNaughton. Playing infinite games in finite time. In *A Half-Century of Automata Theory: Celebration and Inspiration*, pages 73–91. World Scientific, 2000.

9 Stéphane Le Roux. Infinite subgame perfect equilibrium in the Hausdorff difference hierarchy. In Mohammad Taghi Hajiaghayi and Mohammad Reza Mousavi, editors, *Topics in Theoretical Computer Science – The First IFIP WG 1.8 International Conference, TTCS 2015, Tehran, Iran, August 26-28, 2015, Revised Selected Papers*, volume 9541 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2015. doi:10.1007/978-3-319-28678-5_11.

10 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015. doi:10.1016/j.ic.2015.03.001.

Constructing the Space of Valuations of a Quasi-Polish Space as a Space of Ideals

Matthew de Brecht ✉
Kyoto University, Japan

Abstract

We construct the space of valuations on a quasi-Polish space in terms of the characterization of quasi-Polish spaces as spaces of ideals of a countable transitive relation. Our construction is closely related to domain theoretical work on the probabilistic powerdomain, and helps illustrate the connections between domain theory and quasi-Polish spaces. Our approach is consistent with previous work on computable measures, and can be formalized within weak formal systems, such as subsystems of second order arithmetic.

2012 ACM Subject Classification Mathematics of computing → Topology; Theory of computation → Probabilistic computation

Keywords and phrases Quasi-Polish spaces, space of valuations, domain theory, measure theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.9

Funding This work was supported by JSPS KAKENHI Grant Number 18K11166.

Acknowledgements We thank the reviewers for carefully reading this paper and providing feedback.

1 Introduction

Quasi-Polish spaces [2] are a class of well-behaved countably based sober spaces that includes Polish spaces, ω-continuous domains, and countably based spectral spaces. They can be interpreted via Stone-duality as the spaces of models of countably axiomatized propositional geometric theories [12, 1]. In [7] another characterization of quasi-Polish spaces was presented that is a natural generalization of the notion of an *abstract basis* for ω-continuous domains [8]. In this paper we use this latter characterization to extend domain theoretical work on probabilistic powerdomains to the study of valuations on quasi-Polish spaces.

Valuations are a substitute for Borel measures which are used in the denotational semantics of probabilistic programming languages [14] and in computable approaches to measure theory, probability theory, and randomness [19, 13, 18]. See R. Heckmann's excellent paper [11] for more on the theory of valuations, spaces of valuations, and integration[1]. Every valuation on a quasi-Polish space can be extended to a Borel measure [5], and this extension is unique if the valuation is locally finite [3]. Conversely, it is easy to see that the restriction of a Borel measure to the open sets is a valuation. Thus, in particular, there is a bijection between probabilistic valuations and probabilistic Borel measures on quasi-Polish spaces.

The main result in this paper is a construction of the space of valuations on a quasi-Polish space as a space of ideals of a transitive relation on a countable set (Theorem 13). Our construction is closely related to domain theoretical work on the probabilistic powerdomain (see [14] and [8, Section IV-9]). Along with the constructions of the upper and lower powerspaces of quasi-Polish spaces as spaces of ideals given in [4], our results demonstrate how some domain theoretic results generalize well to quasi-Polish spaces (see also [6] for more on the upper and lower powerspaces of quasi-Polish spaces).

[1] The valuations in this note correspond to the *Scott-continuous valuations* in [11].

An immediate corollary of our construction is that the space of valuations on a quasi-Polish space is again a quasi-Polish space, although this already follows from well-known results. A locale theoretic proof easily follows from S. Vickers' geometricity result in [20, Proposition 5] by using R. Heckmann's characterization of quasi-Polish spaces as countably presented locales [12]. A proof based on quasi-metrics, at least for the case of subprobabilistic valuations, follows from J. Goubault-Larrecq's work on continuous Yoneda-complete quasi-metric spaces in [9, Section 11] and his characterization of quasi-Polish spaces in [10, Theorem 8.18]. Independently, the first proof we found (which we presented at the Domains XII conference in August 2015) was largely based on M. Schröder's work in [19] on the space of (probabilistic) measures within the cartesian closed category QCB_0. That proof starts with the observation that the QCB_0 exponential $\mathbb{S}^{\mathbb{S}^X}$ is quasi-Polish whenever X is[2], then uses the cartesian closed structure of QCB_0 to show that $Y^{\mathbb{S}^X}$ is quasi-Polish whenever X and Y are, and finally observes that M. Schröder's construction of the space of valuations on X can be obtained as the equalizer of the continuous functions $\ell, r\colon \overline{\mathbb{R}}_+^{\mathbb{S}^X} \to \overline{\mathbb{R}}_+ \times \overline{\mathbb{R}}_+^{\mathbb{S}^X \times \mathbb{S}^X}$ defined as:

$$\begin{aligned} \ell(\nu) &= \big\langle \nu(\emptyset), \lambda\langle U, V\rangle.\nu(U) + \nu(V)\big\rangle, \text{ and} \\ r(\nu) &= \big\langle 0, \lambda\langle U, V\rangle.\nu(U \cup V) + \nu(U \cap V)\big\rangle. \end{aligned}$$

It follows that the space of valuations is quasi-Polish because the space of extended reals $\overline{\mathbb{R}}_+$ is quasi-Polish and the category of quasi-Polish spaces is closed under countable limits.

A nice characteristic of the construction we give in this paper is that it can be formalized within relatively weak formal systems. For example, our approach is related to C. Mummert's formalization of general topology within subsystems of second order arithmetic [15, 16, 17][3].

2 Main result

We let $\overline{\mathbb{R}}_+$ denote the positive extended reals (i.e., $[0, \infty]$) with the Scott-topology induced by the usual order. Given a topological space X, we let $\mathbf{O}(X)$ denote the lattice of open subsets of X with the Scott-topology.

▶ **Definition 1** (Valuations). *Let X be a topological space. A* valuation *on X is a continuous function $\nu\colon \mathbf{O}(X) \to \overline{\mathbb{R}}_+$ satisfying:*

1. $\nu(\emptyset) = 0$, *and* (strictness)
2. $\nu(U) + \nu(V) = \nu(U \cup V) + \nu(U \cap V)$. (modularity)

The space of valuations *on X is the set $\mathbf{V}(X)$ of all valuations on X with the* weak topology, *which is generated by subbasic opens of the form*

$$\langle U, q\rangle := \{\nu \in \mathbf{V}(X) \mid \nu(U) > q\}$$

with $U \in \mathbf{O}(X)$ and $q \in \overline{\mathbb{R}}_+ \setminus \{\infty\}$.

In this paper we will only consider the whole space of valuations $\mathbf{V}(X)$, but it is straightforward to modify our results for the subspaces of $\mathbf{V}(X)$ consisting of *probabilistic valuations* (i.e., valuations satisfying $\nu(X) = 1$) and *sub-probabilistic valuations* (i.e., valuations satisfying $\nu(X) \leq 1$).

---

[2] See [6] for a proof. The $\mathbb{S}$ here is the Sierpinski space, and the space $\mathbf{O}(\mathbf{O}(X))$ defined in [6] is homeomorphic to the QCB_0 exponential object $\mathbb{S}^{\mathbb{S}^X}$ when X is quasi-Polish.

[3] Note that C. Mummert's *MF-spaces* are in general $\mathbf{\Pi}^1_1$-complete spaces, whereas quasi-Polish spaces correspond to the $\mathbf{\Pi}^0_2$-level of the Borel hierarchy. This explains why $\mathbf{\Pi}^1_1 - \mathsf{CA}_0$ is required to prove MF-spaces are closed under G_δ-subsets, whereas our construction of $\mathbf{\Pi}^0_2$-subspaces of quasi-Polish spaces in Theorem 3 of [4] can be done within ACA_0.

Quasi-Polish spaces were introduced in [2]. In this paper we will define them using the following equivalent characterization from [7] (see also [4]).

▶ **Definition 2.** *Let $\prec$ be a transitive relation on $\mathbb{N}$. A subset $I \subseteq \mathbb{N}$ is an* ideal *(with respect to $\prec$) if and only if:*

1. $I \neq \emptyset$, (I is non-empty)
2. $(\forall a \in I)(\forall b \in \mathbb{N})\,(b \prec a \Rightarrow b \in I)$, ($I$ is a lower set)
3. $(\forall a, b \in I)(\exists c \in I)\,(a \prec c \,\&\, b \prec c)$. ($I$ is directed)

The collection $\mathbf{I}(\prec)$ of all ideals has the topology generated by basic open sets of the form $[n]_\prec = \{I \in \mathbf{I}(\prec) \mid n \in I\}$. A space is quasi-Polish *if and only if it is homeomorphic to $\mathbf{I}(\prec)$ for some transitive relation $\prec$ on $\mathbb{N}$.*

We often apply the above definition to other countable sets with the implicit assumption that it has been suitably encoded as a subset of $\mathbb{N}$.

Fix a transitive relation $\prec$ on $\mathbb{N}$ for the rest of this section. Let $\mathcal{B}$ be the (countable) set of all partial functions $r :\subseteq \mathbb{N} \to \mathbb{Q}_{>0}$ such that $dom(r)$ is finite, where $\mathbb{Q}_{>0}$ is the set of rational numbers strictly larger than zero.

▶ **Definition 3.** *Define the transitive relation $\prec_V$ on $\mathcal{B}$ as $r \prec_V s$ if and only if*

$$\sum_{b \in F} r(b) < \sum_{c \in \uparrow F \cap dom(s)} s(c)$$

for every non-empty $F \subseteq dom(r)$, where $\uparrow F = \{c \in \mathbb{N} \mid (\exists b \in F)\, b \prec c\}$.

Transitivity of $\prec_V$ follows from the transitivity of $\prec$. Note that if $dom(r) = \emptyset$ then $r \prec_V s$ for every $s \in \mathcal{B}$. We will sometimes use the fact that if $r \prec_V s$ and $b \in dom(r)$ then there is $c \in dom(s)$ with $b \prec c$.

▶ **Definition 4.** *Define $f_V \colon \mathbf{V}(\mathbf{I}(\prec)) \to \mathbf{I}(\prec_V)$ and $g_V \colon \mathbf{I}(\prec_V) \to \mathbf{V}(\mathbf{I}(\prec))$ as*

$$f_V(\nu) = \left\{ r \in \mathcal{B} \,\middle|\, \sum_{b \in F} r(b) < \nu\Big(\bigcup_{b \in F} [b]_\prec\Big) \text{ for every non-empty } F \subseteq dom(r) \right\},$$

$$g_V(I) = \lambda U. \bigvee \left\{ \sum_{b \in dom(r)} r(b) \,\middle|\, r \in I \text{ and } \bigcup_{b \in dom(r)} [b]_\prec \subseteq U \right\}.$$

We next prove a few lemmas which will be used to show that f_V and g_V are continuous inverses of each other.

▶ **Lemma 5.** *If $I \in \mathbf{I}(\prec_V)$, $r \in I$, and $A \subseteq dom(r)$, then $r|_A \in I$, where $r|_A$ is the partial function obtained by restricting the domain of r to A.*

Proof. Since I is directed there is $s \in I$ with $r \prec_V s$. Then clearly $r|_A \prec_V s$ hence $r|_A \in I$ because I is a lower set. ◀

▶ **Definition 6.** *Define the transitive binary relation $\prec_U$ on $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ (the set of finite subsets of $\mathbb{N}$) as $F \prec_U G$ if and only if $(\forall n \in G)\,(\exists m \in F)\, m \prec n$.*

We write $\mathbf{K}(X)$ for the space of saturated compact subsets of X (see [6]).

▶ **Lemma 7** (Lemma 9 & Theorem 10 of [4])**.** *Given $J \in \mathbf{I}(\prec_U)$, the set*

$$g_U(J) = \{I \in \mathbf{I}(\prec) \mid (\forall F \in J)(\exists m \in I)\, m \in F\}$$

is in $\mathbf{K}(\mathbf{I}(\prec))$. Furthermore, for any $S \subseteq \mathbb{N}$, $g_U(J) \subseteq \bigcup_{b \in S} [b]_\prec$ if and only if there is finite $F \subseteq S$ with $F \in J$.

▶ **Lemma 8.** *If $I \in \mathbf{I}(\prec_V)$ and $r \in I$, then there exists $s \in I$ with $r \prec_V s$ and $dom(r) \prec_U dom(s)$.*

Proof. Choose any $t \in I$ with $r \prec_V t$. Let s be the restriction of t to have $dom(s) = \{c \in dom(t) \mid (\exists b \in dom(r))\, b \prec c\}$. Clearly $r \prec_V s$ and $dom(r) \prec_U dom(s)$, and Lemma 5 implies $s \in I$. ◀

▶ **Lemma 9.** *Assume $I \in \mathbf{I}(\prec_V)$ and $r \in I$. Then there exists $K \in \mathbf{K}(\mathbf{I}(\prec))$ such that*

- *$K \subseteq \bigcup_{b \in dom(r)} [b]_\prec$, and*
- *For any finite $F \subseteq \mathbb{N}$, if $K \subseteq \bigcup_{b \in F} [b]_\prec$, then there is $s \in I$ with $r \prec_V s$ and $F \prec_U dom(s)$ and $K \subseteq \bigcup_{c \in dom(s)} [c]_\prec \subseteq \bigcup_{b \in F} [b]_\prec$.*

Proof. Fix $I \in \mathbf{I}(\prec_V)$ and $r \in I$. Using Lemma 8, we can find a $\prec_V$-ascending sequence $(r_i)_{i \in \mathbb{N}}$ in I with $r = r_0$ and $dom(r_i) \prec_U dom(r_{i+1})$ for each $i \in \mathbb{N}$. Then $J = \{F \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \mid (\exists i \in \mathbb{N})\, F \prec_U dom(r_i)\}$ is in $\mathbf{I}(\prec_U)$, hence $K = g_U(J) \in \mathbf{K}(\mathbf{I}(\prec))$ and $K \subseteq \bigcup_{b \in dom(r)} [b]_\prec$ by Lemma 7 and the fact that $dom(r) \in J$. Assume $F \subseteq \mathbb{N}$ is finite and $K \subseteq \bigcup_{b \in F} [b]_\prec$. Then $F \in J$ by Lemma 7, hence $F \prec_U dom(r_i)$ for some $i \in \mathbb{N}$. Since $\prec_U$ is transitive, we can assume without loss of generality that $i > 0$. Setting $s = r_i$, we have $s \in I$ and $r \prec_V s$ and $F \prec_U dom(s)$, and since $dom(s) \in J$ it follows from Lemma 7 that $K \subseteq \bigcup_{c \in dom(s)} [c]_\prec$. The claim $\bigcup_{c \in dom(s)} [c]_\prec \subseteq \bigcup_{b \in F} [b]_\prec$ follows from $F \prec_U dom(s)$. ◀

▶ **Lemma 10.** *Let $D \subseteq \mathbb{N}$ be finite, and let $\mathcal{P}_+(D)$ be the set of non-empty subsets of D. Define*

$$U_G = \bigcap_{b \in G} [b]_\prec$$
$$V_G = U_G \cap \bigcup_{b \in D \setminus G} [b]_\prec$$

for each $G \in \mathcal{P}_+(\mathcal{D})$. Let $P \subseteq \mathcal{P}_+(D)$ be an upper set (i.e., if $F \in P$ and $F \subseteq G \subseteq D$ then $G \in P$). If $\nu \in \mathbf{V}(\mathbf{I}(\prec))$ and $\nu(U_G) < \infty$ for each $G \in P$, then

$$\sum_{G \in P} (\nu(U_G) - \nu(V_G)) = \nu\left(\bigcup_{G \in P} U_G\right).$$

Proof. The proof is by induction on the size of P. It is trivial when $P = \emptyset$, so assume P is a non-empty upper set and that the lemma holds for all upper sets of size strictly less than P. If F is any minimal element of P, then

$$\begin{aligned} V_F &= \bigcup_{b \in D \setminus F} U_{F \cup \{b\}} \\ &= \bigcup_{G \in P \setminus \{F\}} U_{F \cup G} \\ &= U_F \cap \bigcup_{G \in P \setminus \{F\}} U_G, \end{aligned}$$

so the induction hypothesis and modularity yields

$$\begin{aligned}\sum_{G\in P}(\nu(U_G)-\nu(V_G)) &= \nu(U_F)-\nu(V_F)+\sum_{G\in P\setminus\{F\}}(\nu(U_G)-\nu(V_G))\\ &= \nu(U_F)-\nu\left(U_F\cap\bigcup_{G\in P\setminus\{F\}}U_G\right)+\nu\left(\bigcup_{G\in P\setminus\{F\}}U_G\right)\\ &= \nu\left(\bigcup_{G\in P}U_G\right).\end{aligned}$$

◀

▶ **Lemma 11.** *f_V is well-defined and continuous.*

Proof. We first show that $f_V(\nu)\in\mathbf{I}(\prec_V)$ for each $\nu\in\mathbf{V}(\mathbf{I}(\prec))$.

1. *($f_V(\nu)$ is non-empty).* The partial function with empty domain is in $f_V(\nu)$.
2. *($f_V(\nu)$ is a lower set).* Assume $r\prec_V s\in f_V(\nu)$. Let $F\subseteq dom(r)$ be non-empty, and define $G=\uparrow F\cap dom(s)$. Since $b\prec c$ implies $[c]_\prec\subseteq[b]_\prec$ it follows that $\bigcup_{c\in G}[c]_\prec\subseteq\bigcup_{b\in F}[b]_\prec$. Then

$$\begin{aligned}\sum_{b\in F}r(b) &< \sum_{c\in G}s(c) && \text{(because } r\prec_V s\text{)}\\ &< \nu\Big(\bigcup_{c\in G}[c]_\prec\Big) && \text{(because } s\in f_V(\nu)\text{)}\\ &\leq \nu\Big(\bigcup_{b\in F}[b]_\prec\Big) && \text{(because } \nu \text{ is monotonic),}\end{aligned}$$

   hence $r\in f_V(\nu)$.
3. *($f_V(\nu)$ is directed).* Our proof is related to the series of lemmas leading up to Theorem IV-9.16 in [8]. Assume $r_0,r_1\in f_V(\nu)$. For each $i\in\{0,1\}$ and non-empty $F\subseteq dom(r_i)$ fix some real number β^i_F satisfying

$$\sum_{b\in F}r_i(b)<\beta^i_F<\nu\left(\bigcup_{b\in F}[b]_\prec\right),$$

   and set

$$\beta=\min\left\{\frac{\beta^i_F-\sum_{b\in F}r_i(b)}{\sum_{b\in F}r_i(b)}\;\middle|\; i\in\{0,1\}\ \&\ \emptyset\neq F\subseteq dom(r_i)\right\}.$$

   Then $\alpha=1/(1+\beta/2)$ satisfies $0<\alpha<1$ and is such that

$$\sum_{b\in F}r_i(b)<\alpha\nu\left(\bigcup_{b\in F}[b]_\prec\right)$$

   for each $i\in\{0,1\}$ and non-empty $F\subseteq dom(r_i)$ (see Lemma IV-9.11 (iii) of [8]). Set $M=1+\sum_{b\in dom(r_0)}r_0(b)+\sum_{b\in dom(r_1)}r_1(b)$, and $D=dom(r_0)\cup dom(r_1)$. Let U_G and V_G be defined as in Lemma 10 for each non-empty $G\subseteq D$.
   We define a finite set $h(G)\subseteq\mathbb{N}$ and a function $s_G\colon h(G)\to\mathbb{Q}_>$ for each non-empty $G\subseteq D$ as follows. If $\nu(U_G)=\nu(V_G)$ then let $h(G)=\emptyset$ and let s_G be the empty function. Otherwise, the set

$$C=\{c\in\mathbb{N}\mid(\forall b\in D)\,[b\prec c\iff b\in G]\}$$

is non-empty because $\nu(U_G) > \nu(V_G)$ implies there is some ideal containing G which is not in V_G. If there is some $c \in C$ with $\nu([c]_\prec) = \infty$, then set $h(G) = \{c\}$ and define $s_G \colon h(G) \to \mathbb{Q}_>$ as $s_G(c) = M$. If no such $c \in C$ exists, then let $(c_i)_{i\in\mathbb{N}}$ be an enumeration of C and define

$$p_i = \nu([c_i]_\prec) - \nu\left([c_i]_\prec \cap \left(\bigcup_{k<i}[c_k]_\prec \cup V_G\right)\right).$$

Using modularity and a simple inductive argument, we have

$$\begin{aligned}\sum_{i\leq n} p_i &= \nu(\bigcup_{i\leq n}[c_i]_\prec) - \nu\left(\bigcup_{i\leq n}[c_i]_\prec \cap V_G\right)\\ &= \nu(\bigcup_{i\leq n}[c_i]_\prec \cup V_G) - \nu(V_G)\end{aligned}$$

for each $n \in \mathbb{N}$. Since $U_G = \bigcup_{i\in\mathbb{N}}[c_i]_\prec \cup V_G$ and ν is Scott-continuous, there is $n_0 \in \mathbb{N}$ with

$$\left(\frac{1+\alpha}{2}\right)\sum_{i\leq n_0} p_i \geq \alpha(\nu(U_G) - \nu(V_G))$$

if $\nu(U_G) < \infty$, and

$$\left(\frac{1+\alpha}{2}\right)\sum_{i\leq n_0} p_i \geq M$$

if $\nu(U_G) = \infty$. Define

$$h(G) = \{c_i \mid i \leq n_0 \,\&\, p_i > 0\}$$

and define $s_G \colon h(G) \to \mathbb{Q}_>$ so that $s_G(c_i)$ is a positive rational satisfying

$$\left(\frac{1+\alpha}{2}\right)p_i \leq s_G(c_i) < p_i.$$

Since $h(G) \cap h(G') \neq \emptyset$ implies $G = G'$, there is $s \in \mathcal{B}$ with

$$dom(s) = \bigcup\{h(G) \mid G \subseteq D\}$$

satisfying $s(c) = s_G(c)$ for the unique $G \subseteq D$ with $c \in h(G)$. From the construction of s, if $F \subseteq h(G)$ is non-empty then

$$\sum_{c\in F} s(c) < \nu(\bigcup_{c\in F}[c]_\prec) - \nu(\bigcup_{c\in F}[c]_\prec \cap V_G). \tag{1}$$

Furthermore, if $h(G) \neq \emptyset$, then $\nu(U_G) < \infty$ implies

$$\alpha\,(\nu(U_G) - \nu(V_G)) \leq \sum_{c\in h(G)} s(c), \tag{2}$$

and $\nu(U_G) = \infty$ implies

$$M \leq \sum_{c\in h(G)} s(c). \tag{3}$$

To show $s \in f_V(\nu)$, we must prove $\sum_{c\in F} s(c) < \nu(\bigcup_{c\in F}[c]_\prec)$ for each non-empty $F \subseteq dom(s)$. This clearly holds when $F = \{c\}$ is a singleton. Next, assume it holds for all sets of size less than or equal to n, and let F be a set of size $n+1$. We can assume $\nu(\bigcup_{c\in F}[c]_\prec) < \infty$, since otherwise the claim is trivial. Let $G \subseteq D$ be a set of minimal size satisfying $F \cap h(G) \neq \emptyset$. This implies that either $F \setminus h(G)$ is empty or else it satisfies the induction hypothesis. Furthermore, for any $c \in F \setminus h(G)$ there is $G' \subseteq D$ with $c \in h(G')$, and since the minimality of G implies $G' \not\subseteq G$, there is $b \in G' \setminus G$ with $b \prec c$, which implies $U_G \cap [c]_\prec \subseteq V_G$. Therefore,

$$\begin{aligned}
\sum_{c\in F} s(c) &= \sum_{c\in F\cap h(G)} s_G(c) + \sum_{c\in F\setminus h(G)} s(c) \\
&< \nu\Big(\bigcup_{c\in F\cap h(G)} [c]_\prec\Big) - \nu\Big(\bigcup_{c\in F\cap h(G)} [c]_\prec \cap V_G\Big) + \nu\Big(\bigcup_{c\in F\setminus h(G)} [c]_\prec\Big) \\
&\quad \text{(by (1) and the induction hypothesis)} \\
&\leq \nu\Big(\bigcup_{c\in F\cap h(G)} [c]_\prec\Big) - \nu\Big(\bigcup_{c\in F\cap h(G)} [c]_\prec \cap \bigcup_{c\in F\setminus h(G)} [c]_\prec\Big) \\
&\quad + \nu\Big(\bigcup_{c\in F\setminus h(G)} [c]_\prec\Big) \\
&\quad \text{(because } U_G \cap [c]_\prec \subseteq V_G \text{ for each } c \in F \setminus h(G)\text{)} \\
&= \nu\Big(\bigcup_{c\in F} [c]_\prec\Big),
\end{aligned}$$

which proves $s \in f_V(\nu)$.

Finally, we must show $r_0 \prec_V s$ and $r_1 \prec_V s$. Fix $i \in \{0,1\}$ and non-empty $F \subseteq dom(r_i)$. Set $P = \{G \subseteq D \mid G \cap F \neq \emptyset\}$ and note that $\uparrow F \cap dom(s) = \bigcup_{G\in P} h(G)$. If $\nu(U_G) < \infty$ for each $G \in P$, then using (2) and the fact that $G \neq G'$ implies $h(G) \cap h(G') = \emptyset$, we have

$$\begin{aligned}
\sum_{c\in\uparrow F\cap dom(s)} s(c) &\geq \sum_{G\in P} \alpha(\nu(U_G) - \nu(V_G)) \\
&= \alpha\nu\left(\bigcup_{G\in P} U_G\right) \qquad \text{(by Lemma 10)} \\
&= \alpha\nu\left(\bigcup_{b\in F} [b]_\prec\right) \\
&> \sum_{b\in F} r_i(b).
\end{aligned}$$

Otherwise, there is $G \in P$ with $\nu(U_G) = \infty$, so (3) implies

$$\sum_{c\in\uparrow F\cap dom(s)} s(c) \geq M > \sum_{b\in F} r_i(b).$$

This completes the proof that $f_V(\nu)$ is directed.

It only remains to show that f_V is continuous. Fix $r \in \mathcal{B}$. For each $F \subseteq dom(r)$ define $W_F = \bigcup_{b\in F}[b]_\prec$ and $q_F = \sum_{b\in F} r(b)$, and set $D = \{F \subseteq dom(r) \mid F \neq \emptyset\}$. Then $f_V(\nu) \in [r]_{\prec_V}$ if and only if

$$\nu \in \bigcap_{F\in D} \langle W_F, q_F\rangle,$$

hence f_V is continuous. ◀

▶ **Lemma 12.** *g_V is well-defined and continuous.*

Proof. We first show that $\nu = g_V(I)$ is a valuation for each $I \in \mathbf{I}(\prec_V)$.

1. $\nu(\emptyset) = 0$: Assume $U \in \mathbf{O}(\mathbf{I}(\prec))$ and $\nu(U) > 0$. Then there is $r_0 \in I$ and $b_0 \in dom(r_0)$ such that $[b_0]_\prec \subseteq U$ and $0 < r_0(b_0)$. Since I is directed, there is an infinite sequence $r_0 \prec_V r_1 \prec_V \cdots$ in I. Since $b_0 \in dom(r_0)$ and $r_0 \prec_V r_1$, there is $b_1 \in dom(r_1)$ with $b_0 \prec b_1$. Similarly, there must be $b_2 \in dom(r_2)$ with $b_1 \prec b_2$. This yields an infinite sequence $b_0 \prec b_1 \prec \cdots$, hence $\{c \in \mathbb{N} \mid (\exists i \in \mathbb{N})\, c \prec b_i\}$ is an element of $[b_0]_\prec \subseteq U$. Therefore, $U \neq \emptyset$.
2. $\nu(U) + \nu(V) = \nu(U \cup V) + \nu(U \cap V)$: We first show $\nu(U) + \nu(V) \leq \nu(U \cup V) + \nu(U \cap V)$. Let $r, s \in I$ be such that $(\forall b \in dom(r))\, [b]_\prec \subseteq U$ and $(\forall b \in dom(s))\, [b]_\prec \subseteq V$. Set

$$p_r = \sum_{b \in dom(r)} r(b), \qquad p_s = \sum_{b \in dom(s)} s(b).$$

Let $t \in I$ be a $\prec_V$-upper bound of r and s. Let

$$D_r = \{c \in dom(t) \mid (\exists b \in dom(r))\, b \prec c\},$$
$$D_s = \{c \in dom(t) \mid (\exists b \in dom(s))\, b \prec c\}.$$

Note that $c \in D_r \cap D_s$ implies $[c]_\prec \subseteq U \cap V$. Set

$$q_0 = \sum_{c \in D_r \setminus D_s} t(c), \qquad q_1 = \sum_{c \in D_s \setminus D_r} t(c), \qquad q_2 = \sum_{c \in D_r \cap D_s} t(c).$$

Then $r \prec_V t$ implies $p_r \leq q_0 + q_2$ and $s \prec_V t$ implies $p_s \leq q_1 + q_2$. Furthermore, using the fact $t \in I$, Lemma 5, and the definition of ν, we obtain $\nu(U \cup V) \geq q_0 + q_1 + q_2$ and $\nu(U \cap V) \geq q_2$, hence $p_r + p_s \leq \nu(U \cup V) + \nu(U \cap V)$. It follows that $\nu(U) + \nu(V) \leq \nu(U \cup V) + \nu(U \cap V)$.
Next we show $\nu(U \cup V) + \nu(U \cap V) \leq \nu(U) + \nu(V)$. Let $r, s \in I$ be such that $(\forall b \in dom(r))\, [b]_\prec \subseteq U \cup V$ and $(\forall b \in dom(s))\, [b]_\prec \subseteq U \cap V$. Let $K \subseteq \bigcup_{b \in dom(r)} [b]_\prec$ be as in Lemma 9. Since K is compact and $K \subseteq U \cup V$, there exists a finite set $F \subseteq \mathbb{N}$ with $K \subseteq \bigcup_{b \in F} [b]_\prec$ and such that each $b \in F$ satisfies $[b]_\prec \subseteq U$ or $[b]_\prec \subseteq V$. Apply Lemma 9 to get $t \in I$ with $r \prec_V t$ and $F \prec_U dom(t)$ and $K \subseteq \bigcup_{c \in dom(t)} [c]_\prec \subseteq \bigcup_{b \in dom(r)} [b]_\prec$. Next let $u \in I$ be a $\prec_V$-upper bound of t and s. By restricting the domain of u if necessary, we can assume that $(dom(t) \cup dom(s)) \prec_U dom(u)$, hence every $c \in dom(u)$ satisfies $[c]_\prec \subseteq U$ or $[c]_\prec \subseteq V$. Let u_0 be the restriction of u to have domain $dom(u_0) = \{b \in dom(u) \mid [b]_\prec \subseteq U\}$, and let u_1 be the restriction of u to have domain $dom(u_1) = \{b \in dom(u) \mid [b]_\prec \subseteq V\}$. Note that u_0 and u_1 are both in I by Lemma 5, and that $dom(u) = dom(u_0) \cup dom(u_1)$. Then using the fact that $r \prec_V u$ and $s \prec_V u$, we have

$$\begin{aligned}\sum_{b \in dom(r)} r(b) + \sum_{b \in dom(s)} s(b) &\leq \sum_{c \in dom(u)} u(c) + \sum_{c \in dom(u_0) \cap dom(u_1)} u(c) \\ &= \sum_{c \in dom(u_0)} u_0(c) + \sum_{c \in dom(u_1)} u_1(c) \\ &\leq \nu(U) + \nu(V).\end{aligned}$$

Therefore, $\nu(U \cup V) + \nu(U \cap V) \leq \nu(U) + \nu(V)$.
3. ν is a continuous function: Assume $U \in \mathbf{O}(\mathbf{I}(\prec))$ and $q \in \mathbb{Q}_{>0}$ and $\nu(U) > q$. Since $\mathbf{I}(\prec)$ is consonant (see [6]), it suffices to find $K \in \mathbf{K}(\mathbf{I}(\prec))$ such that $K \subseteq U$ and $\nu(W) > q$ whenever W is an open set containing K. By definition of $g_V(I)$, there must be $r \in I$

such that $(\forall b \in dom(r))\, [b]_\prec \subseteq U$ and $\sum_{b\in dom(r)} r(b) > q$. Now let $K \in \mathbf{K}(\mathbf{I}(\prec))$ be as in Lemma 9. Then $K \subseteq U$, and if $K \subseteq W$ then there is $s \in I$ with $r \prec_V s$ and $K \subseteq \bigcup_{c\in dom(s)} [c]_\prec \subseteq W$, hence $q < \sum_{c\in dom(s)} s(c) \leq \nu(W)$.

It only remains to show that g_V is continuous. Assume $g_V(I) \in \langle U, q\rangle$. Then there is $r \in I$ satisfying $(\forall b \in dom(r))\, [b]_\prec \subseteq U$ and $q < \sum_{b\in dom(r)} r(b)$. Then $I \in [r]_{\prec_V} \subseteq g_V^{-1}(\langle U, q\rangle)$, hence g_V is continuous. ◀

▶ **Theorem 13.** *$\mathbf{V}(\mathbf{I}(\prec))$ and $\mathbf{I}(\prec_V)$ are homeomorphic (via f_V and g_V).*

Proof. It only remains to show that f_V and g_V are inverses of each other.

To show that $g_V \circ f_V$ is the identity function, it suffices to show that $g_V(f_V(\nu)) \in \langle U, q\rangle$ if and only if $\nu \in \langle U, q\rangle$ for each $\nu \in \mathbf{V}(\mathbf{I}(\prec))$ and each subbasic open $\langle U, q\rangle$. If $g_V(f_V(\nu)) \in \langle U, q\rangle$, then there must be $r \in f_V(\nu)$ with $q < \sum_{b\in dom(r)} r(b)$ and $\bigcup_{b\in dom(r)} [b]_\prec \subseteq U$. This implies that $dom(r) \neq \emptyset$, and using the definition of f_V we obtain $q < \sum_{b\in dom(r)} r(b) < \nu(\bigcup_{b\in dom(r)} [b]_\prec) \leq \nu(U)$, hence $\nu \in \langle U, q\rangle$. Conversely, if $\nu \in \langle U, q\rangle$ then since ν is continuous there exist $b_0, \ldots, b_n \in \mathbb{N}$ such that $\bigcup_{i\leq n} [b_i]_\prec \subseteq U$ and $q < \nu(\bigcup_{i\leq n} [b_i]_\prec)$. If $\nu([b_i]_\prec) = \infty$ for some $i \leq n$, then the partial function r defined as $dom(r) = \{b_i\}$ and $r(b_i) = q + 1$ is in $f_V(\nu)$, which implies $g_V(f_V(\nu)) \in \langle U, q\rangle$. Otherwise $\nu([b_i]_\prec) < \infty$ for each $i \leq n$, so define

$$m_i = \nu([b_i]_\prec) - \nu\big([b_i]_\prec \cap \bigcup_{j<i} [b_j]_\prec\big).$$

Note that the modularity of ν implies $m_i = \nu(\bigcup_{j\leq i} [b_j]_\prec) - \nu(\bigcup_{j<i} [b_j]_\prec)$, hence a simple inductive argument yields $\sum_{i\leq n} m_i = \nu(\bigcup_{i\leq n} [b_i]_\prec)$, which is strictly larger than q. Let $G = \{i \mid m_i > 0\}$. Then there exists $r \in \mathcal{B}$ with $dom(r) = \{b_i \mid i \in G\}$ and $(\forall i \in G)\, r(b_i) < m_i$ and $q < \sum_{b\in dom(r)} r(b)$. If $F \subseteq G$ is non-empty, then

$$\begin{aligned}\sum_{i\in F} r(b_i) < \sum_{i\in F} m_i &= \sum_{i\in F} \left(\nu([b_i]_\prec) - \nu\big([b_i]_\prec \cap \bigcup_{j<i} [b_j]_\prec\big)\right) \\ &\leq \sum_{i\in F} \left(\nu([b_i]_\prec) - \nu\big([b_i]_\prec \cap \bigcup_{\substack{j<i\\ j\in F}} [b_j]_\prec\big)\right) = \nu\left(\bigcup_{i\in F} [b_i]_\prec\right).\end{aligned}$$

Thus, $r \in f_V(\nu)$ and $q < \sum_{b\in dom(r)} r(b)$, hence $g_V(f_V(\nu)) \in \langle U, q\rangle$.

Next we show that $f_V(g_V(I)) = I$ for each $I \in \mathbf{I}(\prec_V)$. By unwinding the definitions of f_V and g_V, we have $r \in f_V(g_V(I))$ if and only if for every non-empty $F \subseteq dom(r)$ there is $s \in I$ such that $\bigcup_{c\in dom(s)} [c]_\prec \subseteq \bigcup_{b\in F} [b]_\prec$ and $\sum_{b\in F} r(b) < \sum_{c\in dom(s)} s(c)$. Thus, given any $r \in I$, by Lemma 8 there is $s \in I$ with $r \prec_V s$ and $dom(r) \prec_U dom(s)$, hence $\bigcup_{c\in dom(s)} [c]_\prec \subseteq \bigcup_{b\in F} [b]_\prec$ and $\sum_{b\in F} r(b) < \sum_{c\in dom(s)} s(c)$, which implies $r \in f_V(g_V(I))$. Therefore, $I \subseteq f_V(g_V(I))$.

To prove $f_V(g_V(I)) \subseteq I$, fix any $r \in f_V(g_V(I))$. Then for every non-empty $F \subseteq dom(r)$ there is $s_F \in I$ such that $\bigcup_{c\in dom(s_F)} [c]_\prec \subseteq \bigcup_{b\in F} [b]_\prec$ and $\sum_{b\in F} r(b) < \sum_{c\in dom(s_F)} s_F(c)$. Using Lemma 9, we can assume that $F \prec_U dom(s_F)$. Let $s \in I$ be a $\prec_V$-upper bound of all of the s_F. Then for any non-empty $F \subseteq dom(r)$, we have

$$\begin{aligned}\sum_{b\in F} r(b) &< \sum_{c\in\uparrow F\cap dom(s_F)} s_F(c) \qquad \text{(by choice of } s_F\text{)} \\ &< \sum_{c\in\uparrow F\cap dom(s)} s(c) \qquad \text{(because } s_F \prec_V s \text{ and } \prec \text{ is transitive)}.\end{aligned}$$

Therefore $r \prec_V s$, hence $r \in I$ because I is a lower-set. It follows that $f_V(g_V(I)) \subseteq I$, which completes the proof that $f_V(g_V(I)) = I$. ◀

We remark that the homeomorphisms f_V and g_V are computable in the sense of TTE [21] when $\prec$ is computably enumerable, and therefore our approach is consistent with previous work on computable measures in [19, 13, 18]. The computability of f_V is obvious. For g_V, note that for any $U \in \mathbf{O}(\mathbf{I}(\prec))$ and any $A \subseteq \mathbb{N}$ satisfying $U = \bigcup_{a\in A}[a]_\prec$, Lemma 9 implies

$$g_V(I)(U) = \bigvee \left\{ \sum_{c\in dom(s)} s(c) \;\middle|\; s \in I \,\&\, (\forall c \in dom(s))(\exists a \in A)\, a \prec c \right\},$$

which shows that g_V is computable.

References

1 R. Chen. Borel functors, interpretations, and strong conceptual completeness for $\mathcal{L}_{\omega_1\omega}$. *Transactions of the American Mathematical Society*, 372:8955–8983, 2019.

2 M. de Brecht. Quasi-Polish spaces. *Annals of Pure and Applied Logic*, 164:356–381, 2013.

3 M. de Brecht. Extending continuous valuations on quasi-Polish spaces to Borel measures. Twelfth International Conference on Computability and Complexity in Analysis, 2015.

4 M. de Brecht. Some notes on spaces of ideals and computable topology. In *Proceedings of the 16th Conference on Computability in Europe, CiE 2020*, volume 12098 of *Lecture Notes in Computer Science*, pages 26–37, 2020.

5 M. de Brecht, J. Goubault-Larrecq, X. Jia, and Z. Lyu. Domain-complete and LCS-complete spaces. *Electronic Notes in Theoretical Computer Science*, 345:3–35, 2019.

6 M. de Brecht and T. Kawai. On the commutativity of the powerspace constructions. *Logical Methods in Computer Science*, 15:1–25, 2019.

7 M. de Brecht, A. Pauly, and M. Schröder. Overt choice. *Computability*, 9:169–191, 2020.

8 G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. W. Mislove, and D. S. Scott. *Continuous Lattices and Domains*. Cambridge University Press, 2003.

9 J. Goubault-Larrecq. Complete quasi-metrics for hyperspaces, continuous valuations, and previsions. arXiv, 2017. `arXiv:1707.03784`.

10 J. Goubault-Larrecq and K. Ng. A few notes on formal balls. *Logical Methods in Computer Science*, 13(4):1–34, 2017.

11 R. Heckmann. Spaces of valuations. *Annals of the New York Academy of Sciences*, 806(1):174–200, 1996.

12 R. Heckmann. Spatiality of countably presentable locales (proved with the Baire category theorem). *Math. Struct. in Comp. Science*, 25:1607–1625, 2015.

13 M. Hoyrup and C. Rojas. Computability of probability measures and Martin-Löf randomness over metric spaces. *Information and Computation*, 207:830–847, 2009.

14 C. Jones. *Probabilistic Non-determinism*. PhD thesis, University of Edinburgh, 1989.

15 C. Mummert. *On the Reverse Mathematics of General Topology*. PhD thesis, Pennsylvania State University, 2005.

16 C. Mummert. Reverse Mathematics of MF Spaces. *Journal of Mathematical Logic*, 06(02):203–232, 2006.

17 C. Mummert and F. Stephan. Topological aspects of Poset spaces. *Michigan Mathematical Journal*, 59(1):3–24, 2010.

18 A. Pauly, D. Seon, and M. Ziegler. Computing Haar Measures. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020*, volume 152 of *LIPIcs*, pages 34:1–34:17, 2020.

19 M. Schröder. Admissible representations of probability measures. *Electr. Notes Theor. Comput. Sci.*, 167:61–78, 2007.

20 S. Vickers. A localic theory of lower and upper integrals. *Mathematical Logic Quarterly*, 54:109–123, 2008.

21 K. Weihrauch. *Computable Analysis*. Springer, 2000.

On the Complexity of SPEs in Parity Games

Léonard Brice
Université Gustave Eiffel, Marne-la-Vallée, France
Université Libre de Bruxelles, Belgium

Jean-François Raskin
Université Libre de Bruxelles, Belgium

Marie van den Bogaard
LIGM, Univ. Gustave Eiffel, CNRS, F-77454 Marne-la-Vallée, France

Abstract

We study the complexity of problems related to subgame-perfect equilibria (SPEs) in infinite duration non zero-sum multiplayer games played on finite graphs with parity objectives. We present new complexity results that close gaps in the literature. Our techniques are based on a recent characterization of SPEs in prefix-independent games that is grounded on the notions of requirements and negotiation, and according to which the plays supported by SPEs are exactly the plays consistent with the requirement that is the least fixed point of the negotiation function. The new results are as follows. First, checking that a given requirement is a fixed point of the negotiation function is an **NP**-complete problem. Second, we show that the SPE constrained existence problem is **NP**-complete, this problem was previously known to be **ExpTime**-easy and **NP**-hard. Third, the SPE constrained existence problem is fixed-parameter tractable when the number of players and of colors are parameters. Fourth, deciding whether some requirement is the least fixed point of the negotiation function is complete for the second level of the Boolean hierarchy. Finally, the SPE-verification problem – that is, the problem of deciding whether there exists a play supported by a SPE that satisfies some LTL formula – is **PSpace**-complete, this problem was known to be **ExpTime**-easy and **PSpace**-hard.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Logic and verification; Theory of computation → Solution concepts in game theory

Keywords and phrases Games on graphs, subgame-perfect equilibria, parity objectives

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.10

Related Version *Full Version*: `https://arxiv.org/abs/2107.07458` [3]

Funding This work is partially supported by the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS & FWO), the COST Action 16228 GAMENET (European Cooperation in Science and Technology), and by the PDR project *Subgame perfection in graph games* (F.R.S.- FNRS).

Acknowledgements We wish to thank anonymous reviewers for their useful observations.

1 Introduction

Nash equilibrium (NE) is one of the central concepts from game theory to formalize the notion of rationality. It describes profiles of strategies in which no player has an incentive to change their strategy unilaterally. However, in sequential games, like games played on graphs, NEs are known to be plagued by *non-credible threats*: players can threaten other players in subgames with *non-rational actions* in order to force an equilibrium that avoids these subgames. To avoid non-credible threats, subgame-perfect equilibria are used instead. *Subgame-perfect equilibria* (SPEs) are NEs that are NEs in all subgames of the original game: the players must act rationally in all subgames even after a deviation by another player.

In this paper, we study the complexity of decision problems related to SPEs in sequential games played on graphs with parity objectives. In such a game, each vertex of the game graph has one color per player, and each player wants the least color they see infinitely often along a play, which is an infinite path in the graph, to be even. Parity conditions, in games as well as in automata, are canonical ways to represent ω-regular constraints. It is known that SPEs always exist in parity games, as shown in [14]. Unfortunately, the precise complexity of the SPE constrained existence problem, i.e. the problem of deciding whether there exists an SPE that generates payoffs between two given thresholds, is left open in the literature: it is known to be **ExpTime**-easy and **NP**-hard. We prove here that it is in fact **NP**-complete, and we provide several other new complexity results on related problems of interest.

While previous attempts to solve this decision problem were based on alternating tree automata ([9]), we obtain the new tight complexity results starting from concepts that we have introduced recently in [2] to capture SPEs in mean-payoff games: the notions of *requirements* and *negotiation*. A requirement is a function λ that maps each vertex v of the game graph to a real value, that represents the lowest payoff that the player controlling v should accept when facing other rational players. A play $\rho = \rho_0\rho_1\dots$ is λ-*consistent* if for each vertex ρ_k, the player controlling ρ_k gets at least the payoff $\lambda(\rho_k)$ in ρ. In Boolean games, such as parity games, we naturally consider requirements whose values are either 0 or 1 (1 meaning that the player must achieve their objective, 0 that they may not).

The negotiation function maps a requirement λ to a requirement $\mathsf{nego}(\lambda)$, which captures from any vertex v the maximal payoff that the corresponding player can ensure, against λ-*rational* players, that is, players who play in such a way that they obtain at least the payoff specified by λ. Clearly, if λ_0 maps each vertex to 0, then every play is λ_0-consistent. Then, the requirement $\mathsf{nego}(\lambda_0)$ maps each vertex v to its *antagonistic value*, i.e. the best payoff that the player controlling v can ensure against an adversarial coalition of the other players (as any behavior of the other players is λ_0-rational). It is the case that the $\mathsf{nego}(\lambda_0)$-consistent plays are exactly the plays supported by NEs.

But then, the following natural question is: given v and λ, can the player who controls v improve his worst-case value, if only plays that are consistent with λ are proposed by the other players? Or equivalently, can this player enforce a better value when playing against players that are not willing to give away their own worst-case value? which is clearly a minimal goal for any rational adversary. So $\mathsf{nego}(\lambda)(v)$ returns this value; and this reasoning can be iterated. In [2], it is shown that the least fixed point λ^* of the negotiation function is exactly characterizing the set of plays supported by SPEs, for all prefix independent payoff functions with steady negotiation (which is the case for parity objectives) .

Using that characterization of SPEs, we prove that SPEs always exist in parity games (Theorem 25). That result had already been proved by Ummels in [14]: we use the concepts of requirements and negotiation to rephrase his proof in a more succinct way.

Main contributions

In order to get tight complexity results, we establish the links between the negotiation function and a class of zero-sum two-player games, the *abstract negotiation games* (Theorem 28). Those games are played on an infinite arena, but we show that the players can play simple strategies that have polynomial size representation, while still playing optimally (Lemma 36). We show that a non-deterministic polynomial algorithm can decide which player has a winning strategy in that game, i.e. can decide whether $\mathsf{nego}(\lambda)(v) = 0$ or 1, for a given requirement λ on a given vertex in a given game: as a consequence, deciding whether the

requirement λ is a fixed point of the negotiation function is **NP**-easy (Lemma 38). We also show that the computation of λ^*, and consequently the SPE constrained existence problem, are fixed-parameter tractable if we fix the number of players and colors in the game (Theorem 43).

Those algorithms can be exploited to obtain upper bounds for several problems related to SPEs. Most classical of them, the problem of deciding whether there exists an SPE generating a payoff vector between two given thresholds (SPE constrained existence problem), is **NP**-complete (Theorem 48). That problem can be solved without computing the least fixed point of the negotiation function: such a problem is $\mathbf{BH}_2$-complete, as well as its decisional version – i.e. deciding whether a given requirement λ is equal to λ^* (Theorem 49). Finally, deciding whether there exists an SPE generating a play that satisfies some LTL formula (SPE-verification problem) is **PSpace**-complete (Theorem 50).

Related works

In [9], Ummels and Grädel solve the SPE constrained existence problem in parity games, and prove that such games always contain SPEs. Their algorithm is based on the construction of alternating tree automata, on which one can solve the emptiness problem in exponential time. The SPE constrained existence problem is therefore **ExpTime**-easy, which was, to our knowledge, the best upper bound existing in the literature. The authors also prove the **NP**-hardness of that problem. In what follows, we prove that it is actually **NP**-complete.

In [8], Flesch and Predtetchinski present a general non-effective procedure to characterize the plays supported by an SPE in games with finitely many possible payoff vectors, as parity games. That characterization uses the abstract negotiation game, but does not use the notions of requirements and negotiation, and as a consequence does not yield an effective algorithm – their procedure requires to solve infinitely many games that have an uncountable state space.

In [2], we solve the SPE constrained existence problem on mean-payoff games. To that end, we define the notions of requirements and negotiation, and highlight the links between negotiation and the abstract negotiation game. Part of our results can be applied to every prefix-independent game with steady negotiation, which includes parity games. But, in order to get tight complexity results, we need here to introduce new notions, such as reduced strategies and deviation graphs.

In [4], Brihaye et al. prove that the SPE constrained existence problem on quantitative reachability games is **PSpace**-complete. Their algorithm updates continuously a function that heralds the notion of requirement, until it reaches a fixed point, that we can interpret as the least fixed point of the negotiation function.

In [5], Brihaye et al. give a characterization of NEs in cost-prefix linear games, based on the worst-case value. Parity games are cost-prefix linear, and the worst-case value is captured by our notion of requirement. The authors do not study the notion of SPE in their paper.

In [11], Meunier proposes a method to decide the existence of SPEs generating a given payoff, proving that it is equivalent to decide which player has a winning strategy in a Prover-Challenger game. That method could be used with parity games, but it would not lead to a better complexity than [14], and so it would not yield our **NP**-completeness result.

The applications of non-zero sum infinite duration games targeting reactive synthesis problems have gathered significant attention during the recent years, hence a rich literature on that topic. The interested reader may refer to the surveys [1, 6] and their references.

Structure of the paper

In Section 2, we introduce the necessary background. In Section 3, we present the concepts of requirements, negotiation and abstract negotiation game, and show how they can be applied to characterize SPEs in parity games. In Section 4, we turn that characterization into algorithms that solve the aforementioned problems, and deduce upper bounds for their complexities. In Section 5, we match them with lower bounds, and conclude on the precise complexities of those problems. The detailed proofs of our results can be found in appendices of [3], the full version of this paper.

2 Background

In the sequel, we use the word *game* for Boolean turn-based games played on finite graphs.

▶ **Definition 1** (Game). A *game* is a tuple $G = (\Pi, V, (V_i)_{i\in\Pi}, E, \mu)$, where:

- Π is a finite set of *players*;
- (V, E) is a finite directed graph, whose vertices and edges are also called *states* and *transitions*, and in which every state has at least one outgoing transition;
- $(V_i)_{i\in\Pi}$ is a partition of V, where each V_i is the set of the states *controlled* by player i;
- $\mu : V^\omega \to \{0,1\}^\Pi$ is a *payoff function*, which maps each sequence of states ρ to the tuple $\mu(\rho) = (\mu_i(\rho))_i$ of the players' payoffs: player i *wins* ρ if $\mu_i(\rho) = 1$, and *loses* otherwise.

A play in such a game can be seen as an infinite sequence of moves of a token on the graph (V, E): when the token is on a given vertex, the player controlling that vertex chooses the edge along which it will move. And then, the player controlling the next vertex chooses where it moves next, and so on.

▶ **Definition 2** (Play, history). A *play* (resp. *history*) in the game G is an infinite (resp. finite) path in the graph (V, E). We write $\mathsf{Plays}G$ (resp. $\mathsf{Hist}G$) for the set of plays (resp. histories) in G. We write Hist_iG for the set of histories in G of the form hv, where $v \in V_i$. We write $\mathsf{Occ}(\rho)$ (resp. $\mathsf{Occ}(h)$) for the set of vertices that occur at least once in the play ρ (resp. the history h), and $\mathsf{Inf}(\rho)$ for the set of vertices that occur infinitely often in ρ. We write $\mathsf{first}(h)$ (resp. $\mathsf{first}(\rho)$) the first vertex of h (resp. ρ), and $\mathsf{last}(h)$ its last vertex.

Often, we need to specify an initial state for a game.

▶ **Definition 3** (Initialized game). An *initialized game* is a pair (G, v_0), often written $G_{\restriction v_0}$, where G is a game and $v_0 \in V$ is the *initial vertex*. A play (resp. history) of G is a play (resp. history) of $G_{\restriction v_0}$ iff its first state is v_0. We write $\mathsf{Plays}G_{\restriction v_0}$ (resp. $\mathsf{Hist}G_{\restriction v_0}$, $\mathsf{Hist}_iG_{\restriction v_0}$) for the set of plays (resp. histories, histories ending in V_i) in $G_{\restriction v_0}$.

When the context is clear, we call *game* both non-initialized and initialized games.

▶ **Definition 4** (Strategy, strategy profile). A *strategy* for player i in $G_{\restriction v_0}$ is a function $\sigma_i : \mathsf{Hist}_iG_{\restriction v_0} \to V$ such that for each history $hv \in \mathsf{Hist}_iG_{\restriction v_0}$, we have $v\sigma_i(hv) \in E$.

A *strategy profile* for $P \subseteq \Pi$ is a tuple $\bar{\sigma}_P = (\sigma_i)_{i\in P}$ where each σ_i is a strategy for player i. When $P = \Pi$, the strategy profile is *complete*, and we usually write it $\bar{\sigma}$. For each $i \in \Pi$, we write $-i$ for the set $\Pi \setminus \{i\}$. When $\bar{\tau}_P, \bar{\tau}'_Q$ are two strategy profiles with $P \cap Q = \emptyset$, we write $(\bar{\tau}_P, \bar{\tau}'_Q)$ the strategy profile $\bar{\sigma}_{P\cup Q}$ defined by $\sigma_i = \tau_i$ if $i \in P$, and $\sigma_i = \tau'_i$ if $i \in Q$. We write $\Sigma_iG_{\restriction v_0}$ (resp. $\Sigma_PG_{\restriction v_0}$) the set of all strategies (resp. strategy profiles) for player i (resp. the set P) in $G_{\restriction v_0}$.

A history or a play is *compatible with* (or *supported by*) a strategy σ_i if for each of its prefixes hv with $h \in \mathsf{Hist}_i G$, we have $v = \sigma_i(h)$. It is compatible with a strategy profile $\bar{\sigma}_P$ if it is compatible with σ_i for each $i \in P$. When a strategy profile $\bar{\sigma}$ is complete, there is one unique play in $G_{\restriction v_0}$ that is compatible with it, written $\langle \bar{\sigma} \rangle_{v_0}$ and called the *outcome* of $\bar{\sigma}$.

A strategy σ_i is *memoryless* when for each state v and every two histories h and h', we have $\sigma_i(hv) = \sigma_i(h'v)$. In that case, we liberally consider that σ_i is defined from every state, and write $\sigma_i(v)$ for every $\sigma_i(hv)$.

Before defining the notion of SPEs, we need to define a weaker, but more classical, solution concept: Nash equilibria. A Nash equilibrium is a strategy profile such that no player can improve their payoff by deviating unilaterally from their strategy.

▶ **Definition 5** (Nash equilibrium)**.** A complete strategy profile $\bar{\sigma}$ in $G_{\restriction v_0}$ is a *Nash equilibrium* – or *NE* for short – iff for each player i and for every strategy σ'_i, we have $\mu_i(\langle \bar{\sigma}_{-i}, \sigma'_i \rangle_{v_0}) \leq \mu_i(\langle \bar{\sigma} \rangle_{v_0})$.

An SPE is an NE in all the subgames, in the following formal sense.

▶ **Definition 6** (Subgame, substrategy)**.** Let hv be a history in $G_{\restriction v_0}$. The *subgame* of G after hv is the initialized game $G_{\restriction hv} = (\Pi, V, (V_i)_i, E, \mu_{\restriction hv})_{\restriction v}$, where $\mu_{\restriction hv}$ maps each play to its payoff in G, assuming that the history hv has already been played: formally, for every $\rho \in \mathsf{Plays} G_{\restriction hv}$, we have $\mu_{\restriction hv}(\rho) = \mu(h\rho)$. If σ_i is a strategy in $G_{\restriction v_0}$, its *substrategy* after hv is the strategy $\sigma_{i \restriction hv}$ in $G_{\restriction hv}$, defined by $\sigma_{i \restriction hv}(h') = \sigma_i(hh')$ for every $h' \in \mathsf{Hist}_i G_{\restriction hv}$.

▶ **Definition 7** (Subgame-perfect equilibrium)**.** A complete strategy profile $\bar{\sigma}$ in $G_{\restriction v_0}$ is a *subgame-perfect equilibrium* – or *SPE* for short – iff for every history hv in $G_{\restriction v_0}$, the substrategy profile $\bar{\sigma}_{\restriction hv}$ is a Nash equilibrium.

Throughout this paper, we mostly study *parity* games.

▶ **Definition 8** (Parity game)**.** The game G is a *parity game* if there exists a tuple of *color functions* $(\kappa_i : V \to \mathbb{N})_{i \in \Pi}$, such that each play ρ is won by a given player i – i.e. $\mu_i(\rho) = 1$ – iff the least color seen infinitely often by player i, i.e. the integer $\min \kappa_i(\mathsf{Inf}(\rho))$, is even.

A *Büchi* game is a parity game where all colors are either 0 or 1 – or equivalently, a game in which the objective of each player is to visit infinitely often a given set of vertices. A *coBüchi* game is a parity game where all colors are either 1 or 2 – or equivalently, a game in which the objective of each player is to eventually avoid a given set of vertices.

▶ Example 9. Consider the (coBüchi) game represented by Figure 1: both players win the play ace^ω, and lose any other. A first NE in that game is the strategy profile in which both players always go to the right: its outcome is ace^ω, which is won by both players, hence none can strictly improve their payoff by deviating. A second NE is the strategy profile in which both players always go down: its outcome is ab^ω, which is lost by both players. However, player □ cannot improve his strategy, because he never plays; and player ○ cannot neither, because if she goes right, then □ plans to go down, and she still loses. Only the first one is an SPE: for player □, planning to go down from the state c is a non-credible threat.

An important property of parity games is that they are *prefix-independent*.

▶ **Definition 10** (Prefix-independent game)**.** The game G is *prefix-independent* iff for every history h, we have $\mu_{\restriction h} = \mu$ – or, equivalently, $G_{\restriction h} = G_{\restriction \mathsf{last}(h)}$.

In such games, we search algorithms that solve the following problems. Let us specify that in all the sequel, tuples, as well as mappings, are ordered by the componentwise order.

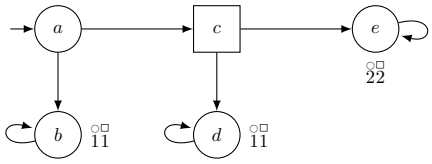


Figure 1 A coBüchi game with two NEs and one SPE.

▶ **Problem 11** (SPE constrained existence problem). Given a parity game $G_{\restriction v_0}$ and two thresholds $\bar{x}, \bar{y} \in \{0,1\}^{\Pi}$, is there an SPE $\bar{\sigma}$ in $G_{\restriction v_0}$ such that $\bar{x} \leq \mu(\langle \bar{\sigma} \rangle_{v_0}) \leq \bar{y}$?

The next problem requires a definition of the linear temporal logic, LTL.

▶ **Definition 12** (LTL formulas). The *linear temporal logic* – or *LTL* for short – over the set of atomic propositions $\mathbb{A}$ is defined as follows: syntactically, each $a \in \mathbb{A}$ is an LTL formula, and if φ and ψ are LTL formulas, then $\neg\varphi$, $\varphi \vee \psi$, $\mathbf{X}\varphi$, and $\varphi \mathbf{U} \psi$ are LTL formulas.

Semantically, if $\nu = \nu_0 \nu_1 \dots$ is an infinite sequence of valuations of $\mathbb{A}$, then:

- $\nu \models a$ iff $\nu_0(a) = 1$;
- $\nu \models \neg\varphi$ iff $\nu \not\models \varphi$;
- $\nu \models \varphi \vee \psi$ iff $\nu \models \varphi$ or $\nu \models \psi$;
- $\nu \models \mathbf{X}\varphi$ iff $\nu_1 \nu_2 \dots \models \varphi$;
- $\nu \models \varphi \mathbf{U} \psi$ iff there exists $k \in \mathbb{N}$ such that $\nu_k \nu_{k+1} \dots \models \psi$, and for each $\ell < k$, we have $\nu_\ell \nu_{\ell+1} \dots \models \varphi$.

We will also make use of the classical notations $\wedge$, $\top$, $\bot$, $\Rightarrow$, $\mathbf{F}$ or $\mathbf{G}$ defined as abbreviations using the symbols chosen here as primitives. In particular, we write $\top$ for $a \vee \neg a$, $\mathbf{F}\varphi$ ("finally φ") for $\top \mathbf{U} \varphi$, and $\mathbf{G}\varphi$ ("globally φ") for $\neg \mathbf{F} \neg \varphi$. When we use LTL to describe plays in a game, w.l.o.g. and for simplicity, the atom set is $\mathbb{A} = V$, and each play ρ is assimilated to the sequence of valuations ν defined by $\nu_k(v) = 1$ iff $\rho_k = v$. For example, when u and v are two vertices, the play $(uv)^\omega$ is the only play satisfying the formula $u \wedge \mathbf{G}\left((u \Rightarrow \mathbf{X}v) \wedge (v \Rightarrow \mathbf{X}u)\right)$.

▶ **Problem 13** (SPE-verification problem). Given a parity game $G_{\restriction v_0}$ and an LTL formula φ, is there an SPE $\bar{\sigma}$ in $G_{\restriction v_0}$ such that $\langle \bar{\sigma} \rangle_{v_0} \models \varphi$?

▶ Remark. The natural problems of deciding whether *all* SPEs generate a payoff vector between two thresholds, or a play that satisfies some LTL formula, are the duals of the aforementioned problems, and their complexities are obtained as direct corollaries. For example, in a given parity game, all the outcomes of SPEs satisfy the formula φ if and only if there does not exist an SPE whose outcome satisfies $\neg\varphi$. Since we will show that the SPE-verification problem is **PSpace**-complete, its dual will also be **PSpace**-complete. Similarly, the SPE constrained universality problem is **coNP**-complete as we will show that its dual, the SPE constrained existence problem, is **NP**-complete.

While we do not recall the definition of classical complexity classes here (such as **NP**, **coNP**, or **PSpace**, see [12]), we recall the definition of the class $\mathbf{BH}_2$: the second level of the Boolean hierarchy. For more details about the Boolean hierarchy itself, see [15].

▶ **Definition 14** (Class $\mathbf{BH}_2$). The complexity class $\mathbf{BH}_2$ is the class of problems of the form $P \cap Q$, where P is **NP**-easy and Q is **coNP**-easy, and both have the same set of instances. In other words, a $\mathbf{BH}_2$-easy problem is a problem that can be decided with one call to an **NP** algorithm, and one to a **coNP** algorithm.

▶ Remark. The class $\mathbf{BH}_2$ must not be mistaken with the class $\mathbf{NP} \cap \mathbf{coNP}$, that gathers the problems that can be solved by an **NP** algorithm *as well as* by a **coNP** one: the latter is included in **NP** and in **coNP**, while the former contains them.

To attach intuition to this definition, let us present a useful $\mathbf{BH}_2$-complete problem.

▶ **Problem 15** (SAT × coSAT)**.** Given a pair (φ_1, φ_2) of propositional logic formulas, is it true that φ_1 is satisfiable and that φ_2 is not?

▶ **Lemma 16.** *The problem* SAT × coSAT *is* $\mathbf{BH}_2$*-complete.*

3 Negotiation in parity games

3.1 Requirements, negotiation, and link with SPEs

In the algorithms we provide, we make use of the characterization of SPEs in prefix-independent games that has been presented in [2]. We recall here the notions needed, slightly adapted to Boolean games.

▶ **Definition 17** (Requirement)**.** A *requirement* on a game G is a mapping $\lambda : V \to \{0, 1, +\infty\}$. The set of the requirements on G is denoted by $\mathsf{Req}G$, and is ordered by the componentwise order $\leq$: we write $\lambda \leq \lambda'$ when we have $\lambda(v) \leq \lambda'(v)$ for all v.

▶ **Definition 18** (λ-consistency)**.** Let λ be a requirement on the game G. A play ρ in G is λ-*consistent* iff for each player i and every index k such that $\rho_k \in V_i$, we have $\mu_i(\rho_k\rho_{k+1}\dots) \geq \lambda(\rho_k)$. The set of λ-consistent plays in $G_{\restriction v_0}$ is denoted by $\lambda\mathsf{Cons}(v_0)$.

In this paper, we will consider specifically requirements that are *satisfiable*.

▶ **Definition 19** (Satisfiability)**.** The requirement λ, on the game G, is *satisfiable* iff for each state $v \in V$, there exists at least one λ-consistent play from v.

▶ **Lemma 20.** *Given a parity game G and a requirement λ, deciding whether λ is satisfiable is* **NP***-easy.*[1]

Proof. An **NP** algorithm for that problem guesses, first, a family $(h_v, W_v)_{v\in V}$, where for each v, h_v is a history without cycle starting from v and W_v is a subset of V with $\mathsf{last}(h_v) \in W_v$. That family is an object of polynomial size, and certifies that λ is satisfiable if for each v: (1) $\lambda(v) \neq +\infty$; (2) the subgraph $(W_v, E \cap W_v^2)$ is strongly connected; (3) for each vertex $u \in \mathsf{Occ}(h_v) \cup W_v$ such that $\lambda(u) = 1$, if i is the player who controls u, then the color min $\kappa_i(W_v)$ is even.

Indeed, if those three points are satisfied, then for each v, the play $h_v c_v^\omega$, where c_v is a cycle (not necessarily simple, but which can be chosen of size at most $(\mathsf{card}W_v)^2$) that visits all the vertices of W_v at least once and none other, is λ-consistent. Conversely, if from each v, there exists a λ-consistent play ρ, then the pair (h_v, W_v), where $W_v = \mathsf{Inf}(\rho)$ and h_v is a prefix of ρ ending in W_v in which the cycles have been removed, satisfies those three properties – those can be checked in polynomial time. ◀

Each requirement induces a notion of rationality for a coalition of players.

[1] It is actually **NP**-complete, as we can prove by slightly adapting the proof of Theorem 47.

▶ **Definition 21** (λ-rationality)**.** Let λ be a requirement on the game G. A strategy profile $\bar{\sigma}_{-i}$ is λ-*rational assuming* the strategy σ_i iff for every history hv compatible with $\bar{\sigma}_{-i}$, the play $\langle \bar{\sigma}_{\restriction hv} \rangle_v$ is λ-consistent. It is λ-*rational* if it is λ-rational assuming some strategy. The set of λ-rational strategy profiles in $G_{\restriction v_0}$ is denoted by $\lambda\mathsf{Rat}(v_0)$.

The notion of λ-rationality qualifies the *environment* against player i, i.e. the coalition of all the players except i: they play λ-rationally if their strategy profile can be completed by a strategy of player i, such that in every subgame, each player gets their requirement satisfied.

But then, λ-rationality restrains the behaviours of the players against player i: that one may be able to win against a λ-rational environment while it is not the case against a fully hostile one. This is what the *negotiation function* captures.

▶ **Definition 22** (Negotiation)**.** The *negotiation function* is a function that transforms every requirement λ into a requirement $\mathsf{nego}(\lambda)$, defined by, for each $i \in \Pi$ and $v \in V_i$, and with the convention $\inf \emptyset = +\infty$:

$$\mathsf{nego}(\lambda)(v) = \inf_{\bar{\sigma}_{-i} \in \lambda\mathsf{Rat}(v)} \sup_{\sigma_i \in \Sigma_i(G_{\restriction v})} \mu_i(\langle \bar{\sigma}_{-i}, \sigma_i \rangle_v).$$

▶ Remark. If player i follows the strategy σ_i assuming which $\bar{\sigma}_{-i}$ is λ-rational, then they get at least the payoff $\lambda(v)$, hence $\mathsf{nego}(\lambda)(v) \geq \lambda(v)$ and the negotiation function is non-decreasing. Moreover, if $\lambda \leq \lambda'$, then all the λ'-rational strategy profiles are also λ-rational, hence $\mathsf{nego}(\lambda) \leq \mathsf{nego}(\lambda')$ and the negotiation function is monotonic.

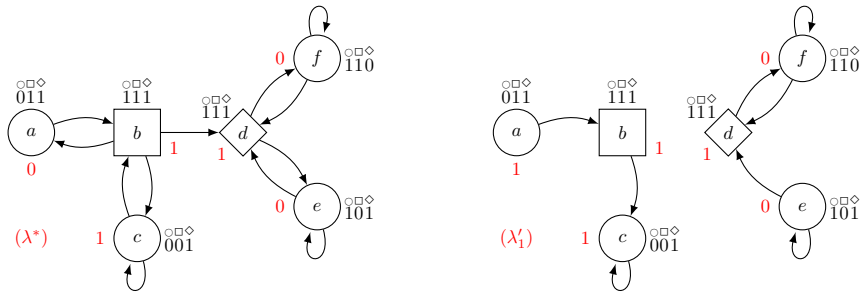
The fixed points of the negotiation function characterize the SPEs of a game: indeed, when some play ρ is λ-consistent for some fixed point λ, it means that it is won by every player who could ensure their victory from a state visited by ρ, while playing against a rational environment. Better: all the SPEs are characterized by the least fixed point of the negotiation function, which exists by Tarski's fixed point theorem, and which we will write λ^* in the rest of this paper. An equivalent result exists for NEs, that are characterized by the requirement $\mathsf{nego}(\lambda_0)$, where $\lambda_0 : v \mapsto 0$ is the *vacuous requirement*.

▶ **Theorem 23.** *In a prefix-independent Boolean game $G_{\restriction v_0}$:*
- *the set of NE outcomes is exactly the set of $\mathsf{nego}(\lambda_0)$-consistent plays;*
- *the set of SPE outcomes is exactly the set of λ^*-consistent plays.*

Proof. By [2], this result is true for any prefix-independent game *with steady negotiation*, i.e. such that for every requirement λ, for every player i and for every vertex v, if there exists a λ-rational strategy profile $\bar{\sigma}_{-i}$ from v, there exists one that minimizes the quantity $\sup_{\sigma_i} \mu_i(\langle \bar{\sigma}_{-i}, \sigma_i \rangle_v)$. In the case of Boolean games, the function μ_i can only take the values 0 and 1, hence this supremum is always realized. ◀

▶ Example 24. Let us consider again the game of Figure 1. Every play in that game – like in every game – is λ_0-rational. The requirement $\lambda_1 = \mathsf{nego}(\lambda_0)$ is equal to 1 on the states c and e (the states from which the player controlling those states can enforce the victory), and to 0 in each other one. Then, the λ_1-consistent plays are exactly the plays supported by a Nash equilibrium: the play ace^ω, and the play ab^ω.

Now, from the state a, the only strategy profile that can make player ○ lose if she chooses to go to c is $\sigma_\square : ac \mapsto d$, which was λ_0-rational but is not λ_1-rational: the play cd^ω is not λ_1-consistent. Therefore, against a λ_1-rational environment, player ○ can enforce the victory by going to the state c, hence $\lambda_2(a) = 1$, where $\lambda_2 = \mathsf{nego}(\lambda_1)$. Then, the requirement λ_2 is a fixed point of the negotiation function, and consequently the least one, hence the only play supported by an SPE from the state a is the only play that is λ_2-consistent, namely ace^ω.



(a) A Büchi game. **(b)** Fixed point as computed in [14].

Figure 2 An illustration of Ummels' algorithm.

3.2 The existence of SPEs in parity games

In parity games, the existence of SPEs is guaranteed.

▶ **Theorem 25** ([14]). *There exists an SPE in every parity game.*

Proof sketch. This theorem is a result due to Ummels. In [3], we rephrase his proof in terms of requirements and negotiation. Let us give here the main intuitions. We define a decreasing sequence $(E_n)_n$ of subsets of E, and an associated sequence $(\lambda'_n)_n$ of requirements, keeping the hypothesis that E_n always contains at least one outgoing edge from each vertex. First, $E_0 = E$ and λ'_0 is the vacuous requirement. Then, for every n, for each player i and each $v \in V_i$, we define $\lambda'_{n+1}(v)$ as equal to 1 if and only if in the game obtained from G by removing the edges that are not in E_n, player i can enforce the victory from v, against a fully hostile environment. Then, from each such state, we choose a memoryless winning strategy (which always exists, see [10]), that is, we choose one edge to always follow to ensure the victory, and we remove the other outgoing edges from E_n to obtain E_{n+1}. We prove that for each n, we have $\lambda'_{n+1} \geq \mathsf{nego}(\lambda'_n)$, hence the sequence $(\lambda'_n)_n$ converges to a satisfiable fixed point of the negotiation function – which is not necessarily the least one. ◀

As a consequence, in every parity game G, we have $\lambda^*(v) \in \{0, 1\}$: this is why in what follows, we only consider requirements with values in $\{0, 1\}$.

▶ Example 26. Let us consider the (Büchi) game of Figure 2a: recall that the objective of each player is to see infinitely often the color 0. If we follow the algorithm from [14], as presented above, we remove the edge df – because always going to e is a winning strategy for player $\diamond$ from d – and the edges ba and bd – because always going to c is a winning strategy for player $\square$ from b. Then, the algorithm reaches a fixed point, see Figure 2b. Our proof states that every play that uses only the remaining edges is a play supported by an SPE. Indeed, those plays are λ'_1-consistent, where λ'_1 is the requirement given in red on Figure 2b: it is a fixed point of the negotiation function. However, it is not the least one: for example, the play $ab(dedf)^\omega$ is not λ'_1-consistent, but it is also a play that is supported by an SPE. The least fixed point is given in red on Figure 2a.

The interested reader will find in [3] an additional example of parity game, on which we computed the iterations of the negotiation function.

3.3 Abstract negotiation game

Now, let us study how we can compute the negotiation function. The *abstract negotiation game* is a tool which already appeared in [8], and which has been linked to the negotiation function in [2]. It is a game on an infinite graph that opposes two players, *Prover* and

Challenger: Prover constructs a λ-rational strategy profile by proposing plays, and Challenger constructs player i's response by accepting those plays or deviating from them. We slightly simplify the definition here, by considering only satisfiable requirements, which guarantees that Prover has always a play to propose[2].

▶ **Definition 27** (Abstract negotiation game)**.** Let G be a parity game, let λ be a satisfiable requirement, let $i \in \Pi$ and let $v_0 \in V_i$. The associated *abstract negotiation game* is the two-player zero-sum game $\mathsf{Abs}_{\lambda i}(G)_{\restriction [v_0]} = (\{\mathbb{P}, \mathbb{C}\}, S, (S_{\mathbb{P}}, S_{\mathbb{C}}), \Delta, \nu)_{\restriction [v_0]}$ where:

- the players $\mathbb{P}$ and $\mathbb{C}$ are called respectively *Prover* and *Challenger*;
- Challenger's states are of the form $[\rho]$, where ρ is a λ-consistent play of G;
- Prover's states are of the form $[hv]$, where $h \in \mathsf{Hist}_i(G) \cup \{\varepsilon\}$ and $\mathsf{last}(h)v \in E$, plus one additional sink state $\top$;
- the set Δ contains the transitions of the forms:
  - $[v][\rho]$, where $\mathsf{first}(\rho) = v$: Prover proposes the play ρ;
  - $[\rho][\rho_0 \dots \rho_k v]$, where $k \in \mathbb{N}$, $v \neq \rho_{k+1}$ and $\rho_k v \in E$: Challenger refuses and deviates;
  - $[hv][v]$ with $h \neq \varepsilon$: then, Prover has to propose a new play from the vertex v;
  - $[\rho]\top$: Challenger accepts the proposed play;
  - $\top\top$: the game is over;
- When π is a play in the abstract negotiation game, we will use the notation $\dot{\pi}$ to denote the play in the original game constructed by Prover's proposals and Challenger's deviations. Thus, the play π is won by Challenger iff one of the following conditions is satisfied:
  - the play π has the form $[v_0][\rho^0][h^0 v_1][v_1][\rho^1] \dots [h^{n-1} v_n][v_n][\rho^n]\top^\omega$, i.e. Challenger accepts a play proposed by Prover, and the play $\dot{\pi} = h^0 \dots h^{n-1}\rho^n$ is won by player i;
  - or the play π has the form $[v_0][\rho^0][h^0 v_1][v_1][\rho^1][h^1 v_2] \dots$, i.e. Challenger always deviates from the play proposed by Prover, and the play $\dot{\pi} = h^0 h^1 \dots$ is won by player i.

▶ **Theorem 28** ([2], Appendix E)**.** *Let G be a prefix-independant Boolean game, let λ be a satisfiable requirement, let $i \in \Pi$ and let $v_0 \in V_i$. Then, we have* $\mathsf{nego}(\lambda)(v_0) = 0$ *if and only if Prover has a winning strategy in the associated abstract negotiation game.*[3]

▶ Example 29. Let G be the game from Figure 1. In this particular case, since there are finitely many possible plays, the abstract negotiation games $\mathsf{Abs}_{\lambda_0 \bigcirc} G$ and $\mathsf{Abs}_{\lambda_1 \bigcirc}$ have a finite state space. They are represented in Figure 3: the blue states are Prover's states, and the orange ones are Challenger's. The dashed states belong to $\mathsf{Abs}_{\lambda_0 \bigcirc} G$ but not to $\mathsf{Abs}_{\lambda_1 \bigcirc}$. Observe that Prover has a winning strategy in $\mathsf{Abs}_{\lambda_0 \bigcirc} G$ (in red), but not in $\mathsf{Abs}_{\lambda_1 \bigcirc} G$.

4 Algorithms

Let us now study how we can use the abstract negotiation game to solve the problems presented in the introduction. We first define an equivalence relation between histories and between plays; then, we show that in the abstract negotiation game, Prover can propose only plays that are simple representatives of their equivalence class, and propose always the same play from each vertex.

[2] In [2], a second sink state $\bot$ is added to enable Prover to give up when she has no play to propose. Another purely technical difference is the existence here of a mandatory transition from each state $[hv]$ to the state $[v]$, instead of letting Prover propose a play directly from the state $[hv]$: thus, there are few states from which Prover has a choice to make, which will be useful in what follows.

[3] The non-existence of the sink state $\bot$, in which Prover is supposed to get the payoff $-\infty$, does not change this result: since λ is assumed to be satisfiable, Prover has always a strategy to get at least the payoff 0, hence no optimal strategy of Prover plans to follow a transition to $\bot$.

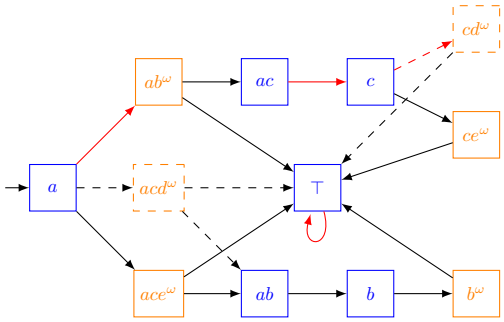


Figure 3 An abstract negotiation game.

4.1 Reduced plays and reduced strategy

The equivalence relation that we use is based on the order in which vertices appear.

▶ **Definition 30** (Occurrence-equivalence (histories))**.** Two histories h and h' are *occurrence-equivalent*, written $h \approx h'$, iff $\mathsf{first}(h) = \mathsf{first}(h')$, $\mathsf{last}(h) = \mathsf{last}(h')$ and $\mathsf{Occ}(h) = \mathsf{Occ}(h')$.

▶ **Definition 31** (Occurrence-equivalence (plays))**.** Two plays ρ and ρ' are *occurrence-equivalent*, written $\rho \approx \rho'$, iff the three following conditions are satisfied:

- $\mathsf{Inf}(\rho) = \mathsf{Inf}(\rho')$;
- for each history prefix of ρ, there exists a occurrence-equivalent history prefix of ρ';
- for each history prefix of ρ', there exists a occurrence-equivalent history prefix of ρ.

▶ Example 32. Let us consider the game of Figure 2a. In that game, the play $ab(dedf)^\omega$ is occurrence-equivalent to the play $abde(dedf)^\omega$, but not to the play $ab(dfde)^\omega$. Indeed, the latter has the history $abdf$ as a prefix, which is not occurrence-equivalent to any prefix of $ab(dedf)^\omega$, in which the state f occurs only when the state e has already occurred.

▶ Remark. The operators Occ, Inf and μ are stable by occurrence-equivalence.

The interest of that equivalence relation lies in the finite number of its equivalence classes, and by the existence of simple representatives of each of them.

▶ **Lemma 33.** *Let ρ be a play of G. There exists a lasso $hc^\omega \approx \rho$ with $|h| \leq n^3 + n^2$ and $|c| \leq n^2$, where $n = \mathsf{card}V$.*

We call such lassos *reduced plays*. For each ρ, we write $\tilde{\rho}$ for an arbitrary occurrence-equivalent reduced play. Then, operations such as computing $\mu(\tilde{\rho})$, $\mathsf{Occ}(\tilde{\rho})$, $\mathsf{Inf}(\tilde{\rho})$, or checking whether $\tilde{\rho}$ is λ-consistent, can be done in time $O(n^3)$.

▶ **Definition 34** (Reduced strategy)**.** A strategy $\tau_{\mathbb{P}}$ for Prover in $\mathsf{Abs}_{\lambda i}(G)$ is *reduced* iff it is memoryless, and for each state v, the play ρ with $[\rho] = \tau_{\mathbb{P}}([v])$ is a reduced play.

▶ Example 35. In Figure 3, Prover's winning strategy, defined by the red arrows, is reduced.

If $\rho \approx \rho'$, and if Challenger can deviate from ρ after the history hv, then he can also deviate in ρ' after some history $h'v$ that traverses the same states. Thus, Prover can play optimally while proposing only reduced plays, and by proposing always the same play from each vertex; that is, by following a reduced strategy.

▶ **Lemma 36.** *Prover has a winning strategy in the abstract negotiation game if and only if she has a reduced one.*

4.2 Checking that a reduced strategy is winning: the deviation graph

We have established that Prover is winning the abstract negotiation game if and only if she has a reduced winning strategy. Such a strategy has polynomial size and can thus be guessed in nondeterministic polynomial time. It remains us to show that we can verify in deterministic polynomial time that a guessed strategy is winning. For that purpose, we construct its *deviation graph*.

▶ **Definition 37** (Deviation graph)**.** Let $\tau_{\mathbb{P}}$ be a reduced strategy of Prover, and let $i \in \Pi$. The *deviation graph* associated to $\tau_{\mathbb{P}}$ and v is the colored graph $\mathsf{Dev}_i(\tau_{\mathbb{P}})$:

- the vertices are the plays $\tau_{\mathbb{P}}([w])$, for every vertex w of the original game;
- there is an edge from $[\tilde{\rho}]$ to $\tau_{\mathbb{P}}([w])$ with color c iff there exists $k \in \mathbb{N}$ such that $\tilde{\rho}_k \in V_i$, $\tilde{\rho}_k w \in E$, $w \neq \tilde{\rho}_{k+1}$ and $\min \kappa_i(\mathsf{Occ}(\tilde{\rho}_0 \dots \tilde{\rho}_k)) = c$.

Constructing the deviation graph associated to a memoryless strategy $\tau_{\mathbb{P}}$ enables to decide whether $\tau_{\mathbb{P}}$ is a winning strategy or not.

▶ **Lemma 38.** *The reduced strategy $\tau_{\mathbb{P}}$ is winning in the abstract negotiation game if and only if in the corresponding deviation graph, there neither exists, from the vertex $\tau_{\mathbb{P}}([v_0])$:*

- *a finite path to a vertex $[\tilde{\rho}]$ such that the play $\tilde{\rho}$ is winning for player i;*
- *nor an infinite path along which the minimal color seen infinitely often is even.*

As a consequence, given a parity game G and a requirement λ, deciding whether λ is a fixed point of the negotiation function is **NP***-easy.*

Proof. Given G and λ, let n be the number of states in G, and m be the number of colors. The deviation graph can be seen as the abstract negotiation game itself, where one removed the transitions that were not compatible with $\tau_{\mathbb{P}}$; removed the states that were not accessible from $[v_0]$; and merged the paths $[\tilde{\rho}][hv][v][\tilde{\rho}']$ into one edge $[\tilde{\rho}][\tilde{\rho}']$ with color $\min \kappa_i(h)$.

Therefore, a path from the vertex $\tau_{\mathbb{P}}([v_0])$ can be seen as a history (if it is finite) or a play (if it is infinite), compatible with the strategy $\tau_{\mathbb{P}}$, in the abstract negotiation game. In particular, the finite paths to a vertex $[\tilde{\rho}]$ with $\mu_i(\tilde{\rho}) = 1$ correspond to the histories that lead Prover to propose a play that is winning for player i, that Challenger can accept to win. Similarly, the infinite paths along which the minimal color seen infinitely often is even correspond to the plays π where Challenger deviates infinitely often, and constructs the play $\dot{\pi}$ that is winning for player i. Such paths will be called *winning paths*.

Now, the deviation graph has n vertices, and at most mn^2 edges. Constructing it requires time $O(n^4)$. Deciding the existence of a finite winning path is a simple accessibility problem, and can be done in time $O(mn^2)$. Deciding the existence of an infinite winning path is similar to deciding the emptiness of a parity automaton, and requires a time $O(mn^3)$. As a consequence, deciding whether a reduced strategy is winning can be done in polynomial time – and it is an object of polynomial size.

Thus, an **NP** algorithm that decides whether the requirement λ is a fixed point of the negotiation function is the following: we guess, at the same time, a certificate that proves that λ is satisfiable (Lemma 20), and a reduced strategy $\tau_{\mathbb{P}}^v$ for Prover in $\mathsf{Abs}_{\lambda i}(G)_{\restriction [v]}$, for each $i \in \Pi$ and $v \in V_i$ such that $\lambda(v) = 0$: by Lemma 36, there exists such a winning strategy if and only if $\mathsf{nego}(\lambda)(v) = 0$. Those objects form a certificate of polynomial size, that can be checked in polynomial time. ◀

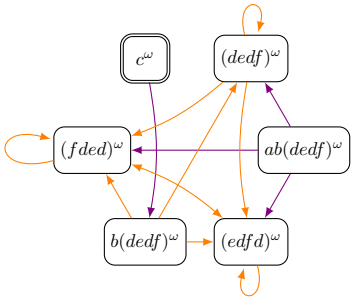


Figure 4 A deviation graph.

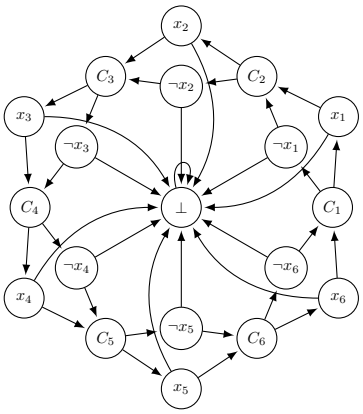


Figure 5 The game G_φ.

▶ Example 39. Let us consider the game of Figure 2a, and the requirement λ^*, presented on the same figure. Let $\tau_{\mathbb{P}}$ be the memoryless strategy in the game $\mathsf{Abs}_{\lambda^*\bigcirc}(G)_{\restriction [a]}$ defined by:

$$\tau_{\mathbb{P}}([a]) = ab(dedf)^\omega \qquad \tau_{\mathbb{P}}([c]) = c^\omega \qquad \tau_{\mathbb{P}}([e]) = (edfd)^\omega$$
$$\tau_{\mathbb{P}}([b]) = b(dedf)^\omega \qquad \tau_{\mathbb{P}}([d]) = (dedf)^\omega \qquad \tau_{\mathbb{P}}([f]) = (fded)^\omega.$$

The corresponding deviation graph is given by Figure 4. The purple edges have color 0, and the orange ones have color 1. Observe that there is no winning path from the vertex $\tau_{\mathbb{P}}([a]) = ab(dedf)^\omega$: each purple edge can be used at most once, and even though the play c^ω is winning for player $\bigcirc$, the corresponding vertex is not accessible. Therefore, the strategy $\tau_{\mathbb{P}}$ is winning in $\mathsf{Abs}_{\lambda^*\bigcirc}(G)_{\restriction [a]}$, which proves the equality $\mathsf{nego}(\lambda^*)(a) = \lambda^*(a) = 0$.

4.3 Upper bounds

Let us now give the main problems for which the concepts given above yield a solution.

A first application is an algorithm for the SPE constrained existence problem.

▶ **Lemma 40.** *The SPE constrained existence problem for parity games is* **NP***-easy.*

Proof. Given a parity game $G_{\restriction v_0}$ and two thresholds $\bar{x}$ and $\bar{y}$, we can guess a reduced play $\tilde{\eta}$ from v_0, a requirement λ, and the certificates required to decide whether λ is a fixed point of nego, according to Lemma 38. All those objects have polynomial size. Then, to check that $\tilde{\eta}$ is an SPE outcome, using Theorem 23, we check that λ is a fixed point of nego, that $\tilde{\eta}$ is λ-consistent, and that $\bar{x} \leq \mu(\tilde{\eta}) \leq \bar{y}$ in polynomial time. ◀

This algorithm does not need an effective characterization of all the SPEs in a game, which is given by the least fixed point of the negotiation function λ^*. Computing such a characterization can be done with a call to a **NP** oracle and to a **coNP** oracle, i.e. it belongs to the class $\mathbf{BH}_2$.

▶ **Lemma 41.** *Given a parity game G and a requirement λ, deciding whether $\lambda = \lambda^*$ is $\mathbf{BH}_2$-easy.*

Proof. First, deciding whether λ is a fixed point of the negotiation function is an **NP**-easy problem by Lemma 38. Deciding whether it is the least one is **coNP**-easy, because a negative instance can be recognized as follows. We guess a requirement $\lambda' < \lambda$, and the certificates of the algorithm given by Lemma 38; then, we check that λ' is a fixed point of nego. ◀

Finally, SPE-verification requires polynomial space.

▶ **Lemma 42.** *The SPE-verification problem is* **PSpace***-easy.*

Proof. Given a parity game $G_{\restriction v_0}$ and an LTL formula φ, by Lemma 41, the requirement λ^* can be computed by a deterministic algorithm using polynomial space – indeed, we have the inclusions $\mathbf{NP} \subseteq \mathbf{PSpace}$ and $\mathbf{coNP} \subseteq \mathbf{PSpace}$, hence the guess of λ^*, followed by one call to an **NP** algorithm and one call to a **coNP** algorithm, can be transformed into a **PSpace** algorithm. Then, we can construct in polynomial time the LTL formula ψ_{λ^*}, that is satisfied exactly by the λ^*-consistent plays:

$$\psi_{\lambda^*} = \bigwedge_i \bigwedge_{v \in V_i, \lambda^*(v)=1} \left(\mathbf{F} v \Rightarrow \bigvee_{2k \leq m} \left(\bigvee_{\kappa_i(w)=2k} \mathbf{GF} w \wedge \bigvee_{\kappa_i(w)<2k} \mathbf{FG} \neg w \right) \right),$$

where m is the largest color appearing in G.

Then, deciding whether there exists an SPE outcome in $G_{\restriction v_0}$ that satisfies the formula φ is equivalent to decide whether there exists a play in $G_{\restriction v_0}$ that satisfies the formula $\varphi \wedge \psi_{\lambda^*}$. As for any LTL formula, that can be done using polynomial space: see for example [13]. ◀

4.4 Fixed-parameter tractability

We end this section by mentioning an additional complexity result on the SPE constrained existence problem: it is fixed-parameter tractable.

▶ **Theorem 43.** *The SPE constrained existence problem on parity games is fixed-parameter tractable when the number of players and the number of colors are parameters. More precisely, there exists a deterministic algorithm that solves that problem in time $O(2^{2^{pm}} n^{12})$, where n is the number of vertices, p is the number of players and m is the number of colors.*

Proof sketch. This result is obtained by constructing and solving a generalized parity game on a finite arena, that is exponential in the number of players and polynomial in the size of the original game. This game, called the *concrete* negotiation game, is equivalent to the abstract negotiation game. Its construction is inspired by a construction that we have introduced in [2], for games with mean-payoff objectives. The main idea of this construction is to decompose the plays proposed by Prover, by passing them to Challenger edges by edges, and by encoding the λ-consistence condition into a generalized parity condition. Then, we can apply a FPT algorithm to solve generalized parity games that was first proposed in [7]. While this deterministic algorithm does not improve on the worst-case complexity of the deterministic **ExpTime** algorithm of [14], it allows for a finer parametric analysis. ◀

5 Matching lower bounds

In this section, we provide matching complexity lower bounds for the problems we adressed in the previous section. For that purpose, we first need the following construction, inspired from a game designed by Ummels in [9] to prove the **NP**-hardness of the SPE constrained existence problem. Our definition is slightly different: while Ummels defined one player per variable, we need one player per literal. However, the core intuitions are the same.

▶ **Definition 44** (G_φ)**.** Let $\varphi = \bigwedge_{j\in\mathbb{Z}/m\mathbb{Z}} C_j$ be a formula of the propositional logic, constructed on the finite set of variables $\{x_1, \dots, x_n\}$. We define the parity game G_φ as follows.

- The players are the variables $x_1, \dots, x_n$, their negations, and *Solver*, denoted by $\mathbb{S}$.
- The states controlled by Solver are all the clauses C_j, and the sink state $\bot$.
- The states controlled by player $L = \pm x_i$ are the pairs (C_j, L), where L is a literal of C_j.
- There are edges from each clause state C_j to all the states (C_j, L); from each pair state (C_j, L) to the state C_{j+1}, and to the sink state $\bot$; and from the sink state $\bot$ to itself.
- For Solver, every state has the color $\kappa_{\mathbb{S}}(v) = 2$, except the state $\bot$, which has color 1.
- For each literal player L, every state has the color $\kappa_L(v) = 2$, except the states of the form $(C, \overline{L})$, that have the color 1.

▶ Remark. The game G_φ is a coBüchi game: Solver has to avoid the sink state $\bot$, and player L the states of the form $(C, \overline{L})$. Therefore, all the following theorems can also be applied to the more restrictive class of coBüchi games.

▶ Example 45. The game G_φ, when φ is the tautology $\bigwedge_{j=1}^{6}(x_j \vee \neg x_j)$, is given by Figure 5.

The game G_φ is strongly linked with the satisfiability of φ, in the following formal sense.

▶ **Lemma 46.** *The game G_φ has the following properties.*

- *The least fixed point of the negotiation function is equal to 0 on the states controlled by Solver, and to 1 on the other ones.*
- *For every SPE outcome ρ in G_φ that does not reach $\bot$, the formula φ is satisfied by:*

$$\nu_\rho : x \mapsto \begin{cases} 1 & \text{if } \exists C, (C, x) \in \mathsf{Inf}(\rho) \\ 0 & \text{otherwise.} \end{cases}$$

- *Conversely, for every valuation ν satisfying φ, the play $\rho_\nu = (C_1(C_1, L_1)\dots C_m(C_m, L_m))^\omega$, where for each j, the literal L_j is satisfied by ν, is an SPE outcome.*

A first consequence is the lower bound on the complexity of deciding whether some requirement is, or not, a fixed point of the negotiation function.

▶ **Theorem 47.** *Given a parity game G and a requirement λ, deciding whether λ is a fixed point of the negotiation function is* **NP***-complete.*

Proof. Easiness is given by Lemma 38. For hardness, we proceed by reduction from the **NP**-complete problem Sat. Given a formula $\varphi = \bigwedge_{j\in\mathbb{Z}/m\mathbb{Z}} C_j$ of the propositional logic, we can construct the game G_φ in polynomial time. Then, let us define on G_φ the requirement λ that is constantly equal to 1, except in $\bot$, where it is equal to 0. Since there is no winning play for Solver from $\bot$, the requirement λ is a fixed point of the negotiation function if and only if there exists a λ-consistent play from each vertex. If it is the case, then let ρ be a

λ-consistent play from C_1. Since $\lambda \geq \lambda^*$, the play ρ is also λ^*-consistent, and is therefore an SPE outcome. It is also a play won by Solver, since $\lambda(C_1) = 1$: therefore, it does not reach the state $\bot$. Then, by Lemma 46, we can define the valuation ν_ρ, which satisfies φ.

Conversely, if ν is a valuation satisfying φ, then the play ρ_ν is an SPE outcome from C_1, and it does not end in $\bot$, hence it is won by Solver, and is therefore λ-consistent. If we consider the suffixes of ρ_ν, we find a λ^*-consistent play from each state C_j; and from the states of the form (C_j, L), the play $(C_j, L)\bot^\omega$ is λ^*-consistent, hence there is a λ^*-consistent play from each vertex: deciding whether λ is a fixed point of the negotiation function is **NP**-hard. ◀

A similar proof ensures the same lower bound on the SPE constrained existence problem.

▶ **Theorem 48.** *The SPE constrained existence problem in parity games is* **NP***-complete.*

Proof. Easiness is given by Lemma 40. For hardness, we proceed by reduction from the problem Sat. Given a formula $\varphi = \bigwedge_{j \in \mathbb{Z}/m\mathbb{Z}} C_j$ of the propositional logic, we can construct the game G_φ in polynomial time. By Lemma 46, there exists an SPE outcome from C_1 and won by Solver, i.e. an SPE outcome from C_1 that does not reach the sink state $\bot$, if and only if there exists a valuation satisfying φ: the SPE constrained existence problem is **NP**-hard. ◀

Now, we show that the algorithm we presented to compute λ^* is optimal as well.

▶ **Theorem 49.** *Given a parity game G and a requirement λ, deciding whether $\lambda = \lambda^*$ is* $\mathbf{BH}_2$*-complete. Given a parity game G, computing λ^* can be done by a non-deterministic algorithm in polynomial time iff* $\mathbf{NP} = \mathbf{BH}_2$.

Finally, the LTL model-checking problem easily reduces to the SPE-verification problem, which gives us the matching lower bound.

▶ **Theorem 50.** *The SPE-verification problem is* **PSpace***-complete.*

References

1. Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications – 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016.
2. Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Subgame-perfect equilibria in mean-payoff games. *CoRR*, abs/2101.10685, 2021. `arXiv:2101.10685`.
3. Léonard Brice, Marie van den Bogaard, and Jean-François Raskin. On the complexity of spes in parity games. *CoRR*, abs/2107.07458, 2021. `arXiv:2107.07458`.
4. 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 *CONCUR*, volume 140 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
5. Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer cost games with simple nash equilibria. In *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.

6 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In *Developments in Language Theory – 21st International Conference, DLT 2017, Liège, Belgium, August 7–11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017.

7 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. Parameterized complexity of games with monotonically ordered omega-regular objectives. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4–7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.29.

8 János Flesch and Arkadi Predtetchinski. A characterization of subgame-perfect equilibrium plays in borel games of perfect information. *Math. Oper. Res.*, 42(4):1162–1179, 2017. doi:10.1287/moor.2016.0843.

9 Erich Grädel and Michael Ummels. Solution Concepts and Algorithms for Infinite Multiplayer Games. In Krzysztof Apt and Robert van Rooij, editors, *New Perspectives on Games and Interaction*, volume 4 of *Texts in Logic and Games*, pages 151–178. Amsterdam University Press, 2008. URL: http://www.logic.rwth-aachen.de/~ummels/knaw07.pdf.

10 Ralf Küsters. Memoryless determinacy of parity games. In 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*, pages 95–106. Springer, 2001. doi:10.1007/3-540-36387-4_6.

11 Noémie Meunier. *Multi-Player Quantitative Games: Equilibria and Algorithms*. PhD thesis, Université de Mons, 2016.

12 Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.

13 Philippe Schnoebelen. The complexity of temporal logic model checking. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logic 4, papers from the fourth conference on "Advances in Modal logic," held in Toulouse, France, 30 September – 2 October 2002*, pages 393–436. King's College Publications, 2002.

14 Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13–15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2006. doi:10.1007/11944836_21.

15 Gerd Wechsung. On the boolean closure of NP. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9–13, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 485–493. Springer, 1985. doi:10.1007/BFb0028832.

Synthetic Integral Cohomology in Cubical Agda

Guillaume Brunerie
Independent researcher, Stockholm, Sweden

Axel Ljungström
Department of Mathematics, Stockholm University, Sweden

Anders Mörtberg
Department of Mathematics, Stockholm University, Sweden

Abstract

This paper discusses the formalization of synthetic cohomology theory in a cubical extension of `Agda` which natively supports univalence and higher inductive types. This enables significant simplifications of many proofs from Homotopy Type Theory and Univalent Foundations as steps that used to require long calculations now hold simply by computation. To this end, we give a new definition of the group structure for cohomology with $\mathbb{Z}$-coefficients, optimized for efficient computations. We also invent an optimized definition of the cup product which allows us to give the first complete formalization of the axioms needed to turn the integral cohomology groups into a graded commutative ring. Using this, we characterize the cohomology groups of the spheres, torus, Klein bottle and real/complex projective planes. As all proofs are constructive we can then use `Cubical Agda` to distinguish between spaces by computation.

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Theory of computation → Type theory

Keywords and phrases Synthetic Homotopy Theory, Cohomology Theory, Cubical Agda

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.11

Supplementary Material A complete formalization of all results in the paper can be found at: *Software (Source Code)*: `https://github.com/agda/cubical/blob/master/Cubical/Papers/ZCohomology.agda`

Funding The material in this paper is based upon research supported by the Swedish Research Council (SRC, Vetenskapsrådet) under Grant No. 2019-04545.

Acknowledgements The authors are grateful to the anonymous reviewers for their insightful comments and feedback. The authors would also like to thank Evan Cavallo for many discussions and for his contributions of various useful results to the `Cubical Agda` library.

1 Introduction

Homotopy Type Theory and Univalent Foundations (HoTT/UF) [38] extends Martin-Löf type theory [30] with Voevodsky's univalence axiom [41] and higher inductive types (HITs). This is based on a close correspondence between types and topological spaces represented as Kan simplicial sets [24]. With this interpretation, points in spaces correspond to elements of types, while paths and homotopies correspond to identity types between these elements [3]. This enables homotopy theory to be developed *synthetically* using type theory. Many classical results from homotopy theory have been formalized in HoTT/UF this way: the definition of the Hopf fibration [38], the Blakers-Massey theorem [22], the Seifert-van Kampen theorem [23] and the Serre spectral sequence [39], among others. Using these results, many homotopy groups of spaces – represented as types – have been characterized. However, just like in classical algebraic topology, these groups tend to be complicated to work with. Because of this, other topological invariants like cohomology have been invented.

Informally, the integral cohomology groups $H^n(X)$ of a space X describe its n-dimensional holes. For instance, the n-dimensional hole in the n-sphere $\mathbb{S}^n$ corresponds to $H^n(\mathbb{S}^n) \simeq \mathbb{Z}$. These holes constitute a topological invariant, making cohomology a powerful technique for establishing which spaces cannot be homotopy equivalent. The usual formulation of singular cohomology using cochain complexes relies on taking the underlying set of topological spaces when defining the singular cochains [19]. This operation is *not* invariant under homotopy equivalence, which makes it impossible to use when formalizing cohomology synthetically. Luckily, there is an alternative definition of cohomology using Eilenberg-MacLane spaces which is homotopy invariant [26]. This was initially studied at the IAS special year on HoTT/UF in 2012–2013 [33] and has since been used to develop the Eilenberg-Steenrod axioms [11] and cellular cohomology [8]. This paper builds on this prior work, but uses `Cubical Agda`– a recent *cubical* extension of the dependently typed programming language `Agda` [35].

The `Cubical Agda` system is based on a variation of cubical type theory formulated by Coquand et al. [14]. These type theories can be seen as refinements of Book HoTT [38] where the homotopical intuitions are taken very literally and made part of the theory. Instead of relying on the inductively defined identity type [29] to define paths and homotopies, a primitive interval type `I` is added. Paths and homotopies are then represented as functions out of `I`, just like in traditional topology. This has some benefits compared to Book HoTT. First, many proofs become simpler. For instance, function extensionality becomes trivial to prove, as opposed to in Book HoTT where it either has to be postulated or derived from the univalence axiom [42]. Second, it gives computational meaning to HoTT/UF, which makes it possible to use the system to do computations using univalence and HITs. Finally, it makes it possible to formulate a general schema for HITs where the eliminators compute definitionally for higher constructors [12, 15]. This is still an open problem for Book HoTT, and HITs have to be added axiomatically, which leads to bureaucratic transports that complicate proofs.

Mörtberg and Pujet explored practical implications of formalizing synthetic homotopy theory in `Cubical Agda` in [31]. This work provided empirical evidence that formalizing synthetic homotopy theory in cubical type theory can lead to significant simplifications of the corresponding formal Book HoTT proofs. For instance, the proof of the 3×3 lemma for pushouts was shortened from 3000 lines of code (LOC) in `HoTT-Agda` [7] to only 200 in `Cubical Agda`. Another proof that becomes substantially shorter is the proof that the torus is equivalent to the product of two circles. This elementary result in topology turned out to have a surprisingly non-trivial proof in Book HoTT because of the lack of definitional computation rules for higher constructors [25, 34]. With the additional computation rules of `Cubical Agda`, this proof is now trivial [40, Sect. 2.4.1].

The present paper is a natural continuation of this prior work and the two main goals are to *characterize* $\mathbb{Z}$-cohomology groups of types and to *compute* using these groups. In classical algebraic topology, *characterize* and *compute* are often used interchangeably when discussing cohomology. We are careful to distinguish these two notions. When *characterizing* a cohomology group of some type, we prove that it is isomorphic to another group. As all of our proofs are constructive, we can then use `Cubical Agda` to actually *compute* with this isomorphism. Having the possibility of doing proofs simply by computation is one of the most appealing aspects of developing synthetic homotopy theory cubically. As this is not possible with pen and paper proofs, or even with many formalized proofs in Book HoTT, one often has to resort to doing long calculations by hand. If proofs instead can be carried out using a computer, many of these long calculations become obsolete. This is a reason why many proofs from synthetic homotopy theory are substantially shorter in `Cubical Agda`. However, not everything has successfully been possible to reduce to computations. A famous

example is the *Brunerie number*. This is a synthetic definition of a number $n : \mathbb{Z}$ such that $\pi_4(\mathbb{S}^3) = \mathbb{Z}/n\mathbb{Z}$. Brunerie proved in his PhD thesis [5] that $n = \pm 2$, but even though this is a constructive definition, it has thus far proved infeasible to compute using `Cubical Agda`, despite considerable efforts. In this paper, we construct a similar number, also inspired by [5], using the multiplicative structure on $H^n(\mathbb{C}P^2)$. This number was proved to be ± 1 using sophisticated techniques in [5, Chapter 6], but we have thus far been unable to verify this purely by computation. However, as this number is substantially simpler than the Brunerie number, it provides a new computational challenge which should be more feasible.

Contributions. The main novel result of the paper is the first computer formalization of the graded commutative ring axioms for $\mathbb{Z}$-cohomology in HoTT/UF (Section 4). To this end, we first develop $\mathbb{Z}$-cohomology groups (Section 3). The definitions are inspired by [5], but the additive structure is new and optimized for efficient computations. The definition of the cup product is also new and provides significant simplifications compared to related proofs in `HoTT-Agda` [4]. We also characterize the cohomology groups of various types (Section 5); for instance, we give the first synthetic characterizations of the cohomology groups of the Klein bottle and real projective plane. In order to characterize $H^n(\mathbb{C}P^2)$, we have verified that our definition of cohomology satisfies the Eilenberg-Steenrod axioms for cohomology theories and constructed the Mayer-Vietoris sequence. We finally reap the fruits of our constructive definitions in Section 6 where we prove that $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$ and the torus are not equivalent by computing with `Cubical Agda`.

All results in the paper have been formalized in `Cubical Agda`. Much of the code in the paper is literal `Cubical Agda` code, but we have taken some liberties when typesetting, to closer resemble standard mathematical notations. In order to clarify the connections between the paper and formalization, we provide a summary file: `https://github.com/agda/cubical/blob/master/Cubical/Papers/ZCohomology.agda`. This file typechecks with the `--safe` flag, which ensures that there are no postulates or unfinished goals.

2 Homotopy Type Theory in Cubical Agda

The `Agda` system [35] is a dependently typed programming language in which both programs and proofs can be written using the same syntax. Dependent function types (Π-types) are written $(x : A) \to B$ while non-dependent function types are written $A \to B$. Implicit arguments to functions are written using curly braces $\{x : A\} \to B$ and function application is written using juxtaposition, so $f\,x$ instead of $f(x)$. Universes are written `Type` ℓ, where ℓ is a *universe level*. In order to ease notation, we omit universe levels in this paper. Readers familiar with `Agda` will also notice that we rename `Set` to `Type`. `Agda` supports many features of modern proof assistants and has recently been extended with an experimental *cubical* mode. The goal of this section is to introduce notions from HoTT/UF (including their formalizations in `Cubical Agda`) which the rest of the paper relies on. Due to space constraints, we omit many technical details and refer curious readers to the paper of Vezzosi et al. [40] for a comprehensive technical treatment of the features of `Cubical Agda`.

2.1 Important notions in Cubical Agda

The first addition to make `Agda` *cubical* is an interval type `I` with endpoints `i0` and `i1`. This corresponds to the real interval $[0, 1] \subset \mathbb{R}$. However, in `Cubical Agda`, this is a purely formal object. A variable $i : \mathsf{I}$ represents a point varying continuously between the endpoints. There are three primitive operations on `I`: *minimum*/*maximum* (`_∧_`, `_∨_` : `I` → `I` → `I`) and *reversal*

($\sim\_ : \mathbb{I} \to \mathbb{I}$). A function $\mathbb{I} \to \mathsf{Type}$ represents a line between two types. By iterating this, we obtain squares, cubes and hypercubes of types making `Agda` inherently *cubical*. In order to specify the endpoints of a line, we use *path types*:

```
PathP : (A : I → Type) → A i0 → A i1 → Type
```

As paths are functions, they are introduced as $\lambda\, i \to t : \mathsf{PathP}\ A\ t[\mathsf{i0}\,/\,i]\ t[\mathsf{i1}\,/\,i]$. Given $p : \mathsf{PathP}\ A\ a_0\ a_1$, we can apply it to $r : \mathbb{I}$ and obtain $p\ r : A\ r$. Also, we always have that $p\ \mathsf{i0}$ reduces to a_0 and $p\ \mathsf{i1}$ reduces to a_1. The PathP types should be thought of as representing heterogeneous equalities since the two endpoints are in different types; this is similar to dependent paths in Book HoTT [38, Section 6.2]. Given $A : \mathsf{Type}$, we define the type of non-dependent paths in A using PathP as follows:

```
_≡_ : A → A → Type
_≡_ x y = PathP (λ _ → A) x y
```

Representing equalities as paths allows us to directly reason about equality. For instance, the constant path $\lambda\ i \to x$ represents a proof of reflexivity $\mathsf{refl} : \{x : A\} \to x \equiv x$. We can also directly apply a function to a path in order to prove that dependent functions respect path-equality, as shown in the definition of cong below:

```
cong : {B : A → Type} {x y : A} (f : (x : A) → B x) (p : x ≡ y) → PathP (λ i → B (p i)) (f x) (f y)
cong f p = λ i → f (p i)
```

We write cong_2 for the binary version of cong; its proof is equally direct. These functions satisfy the standard property that refl gets mapped to refl. They are also definitionally functorial. The latter is an important difference to the corresponding operations defined using path induction which only satisfy the functoriality equations up to a path. Path types also let us prove new things that are not provable in standard `Agda`, e.g. function extensionality:

```
funExt : {B : A → Type} {f g : (x : A) → B x} → ((x : A) → f x ≡ g x) → f ≡ g
funExt p i x = p x i
```

One of the key operations of type theoretic equality is *transport*: given a path between types, we get a function between these types. In `Cubical Agda`, this is defined using another primitive called transp. However, for this paper, the cubical transport function suffices:

```
transport : {A B : Type} → A ≡ B → A → B
transport p a = transp (λ i → p i) i0 a
```

The substitution principle, called "transport" in [38], is an instance of cubical transport:

```
subst : (B : A → Type) {x y : A} → x ≡ y → B x → B y
subst B p b = transport (λ i → B (p i)) b
```

This function invokes transport with a proof that the family B respects the equality p. By combining transport and $\_\wedge\_$, we can define the induction principle for paths. However, an important difference between path types in `Cubical Agda` and Book HoTT is that $\_\equiv\_$ does not behave like an inductive type. In particular, the cubical path induction principle does not definitionally satisfy the computation rule when applied to refl. Nevertheless, we can still prove that this rule holds up to a path. This is a subtle, but important, difference between cubical type theory and Book HoTT. Readers familiar with Book HoTT might be worried

that the failure of this equality to hold definitionally complicates many proofs. However, in our experience, this is rarely the case, as many proofs that require path induction in Book HoTT can be proved more directly using cubical primitives.

`Cubical Agda` also has a primitive operation hcomp for composing paths and, more generally, for composing higher dimensional cubes. An important special case is binary composition of paths $\_\cdot\_ : x \equiv y \to y \equiv z \to x \equiv z$. By composing paths and higher cubes using hcomp, we can reason about paths in a very direct way, avoiding path induction.

2.2 Important concepts from HoTT/UF in Cubical Agda

Pointed types and functions will play an important role in this paper. Formally, a pointed type is a pair $(A, *_A)$ where A is a type with $*_A : A$. We write Type_* for the universe of pointed types. Given $A, B : \mathsf{Type}_*$, a pointed function is a pair $(f, p) : A \to_* B$, where $f : A \to B$ and $p : f\ *_A \equiv *_B$. We often leave $*_A$ and p implicit and write simply $A : \mathsf{Type}_*$ and $f : A \to_* B$. We also sometimes just write A for the underlying type of $A : \mathsf{Type}_*$.

Most HITs in [38] can be defined directly using the general schema of `Cubical Agda`. For example, the circle and suspension HITs can be written as:

```
data 𝕊¹ : Type where
  base : 𝕊¹
  loop : base ≡ base
```

```
data Susp (A : Type) : Type where
  north : Susp A
  south : Susp A
  merid : (a : A) → north ≡ south
```

Functions out of HITs are written using pattern-matching equations, just like regular `Agda` functions. When typechecking the cases for path constructors, `Cubical Agda` checks that the endpoints of what the user writes match up. We could directly define specific higher spheres as HITs with a base point and a constructor for iterated paths. However, the following definition is often easier to work with, as one can reason inductively about it:

▶ **Definition 1** ($\mathbb{S}^n$). *The n-spheres are pointed types defined by recursion:*

$$\mathbb{S}^n = \begin{cases} (\mathsf{Bool}, \mathsf{true}) & \text{if } n = 0 \\ (\mathbb{S}^1, \mathsf{base}) & \text{if } n = 1 \\ (\mathsf{Susp}\ \mathbb{S}^{n-1}, \mathsf{north}) & \text{if } n \geq 2 \end{cases}$$

We could equivalently have defined $\mathbb{S}^1 = (\mathsf{Susp}\ \mathsf{Bool}, \mathsf{north})$, but in our experience, the base/loop-construction is often easier to work with and gives faster computations.

Consistent with the intuition that types correspond to topological spaces (up to homotopy equivalence), we may consider loop spaces of pointed types.

▶ **Definition 2** (Loop spaces). *Given a pointed type $A : \mathsf{Type}_*$, we define its loop space as the pointed type $\Omega\, A = (*_A \equiv *_A, \mathsf{refl})$. For $n : \mathbb{N}$, we let $\Omega^{n+1}\, A = \Omega\, (\Omega^n\, A)$.*

As an example of a non-trivial result which is proved using path induction in Book HoTT, but which can be proved very concisely in `Cubical Agda`, consider the Eckmann-Hilton argument. It says that path composition in higher loop spaces is commutative and can be proved using a single transport with the unit laws for $\_\cdot\_$ and some interval operations.

```
EH : {n : ℕ} (p q : Ω^(2 + n) A) → p ∙ q ≡ q ∙ p
EH p q = transport (λ i → (λ j → rUnit (p j) i) ∙ (λ j → lUnit (q j) i)
                        ≡ (λ j → lUnit (q j) i) ∙ (λ j → rUnit (p j) i))
                   (λ i → (λ j → p (j ∧ ~ i) ∙ q (j ∧ i)) ∙ (λ j → p (~ i ∨ j) ∙ q (i ∨ j)))
```

A type A is not uniquely determined by its points – also (higher) paths in A have to be taken into account. However, for some types, these paths become trivial at some point. We define what this means formally as follows.

▶ **Definition 3** (n-types). *Given $n \geq -2$, a type A is:*

- *a (-2)-type if A is contractible (i.e. A is pointed by a unique point).*
- *an $(n+1)$-type if for all $x, y : A$, $x \equiv y$ is an n-type.*

We write n-Type for the universe of n-types (at some level ℓ).

Equivalently, we could have said that, for $n \geq -1$, A is an n-type if $\Omega^{n+1}A$ is contractible for any choice of base point $a : A$. We follow HoTT/UF terminology and refer to (-1)-types as *propositions* and 0-types as *sets*. A type is a proposition iff all of its elements are path-equal.

Sometimes we are only interested in the structure of a type A and its paths up to a certain level n. That is, we want to turn A into an n-type while preserving the structure of A for levels less than or equal to n. This can be achieved using the n-truncation HITs $\| A \|_n$. Just like for $\mathbb{S}^n$, these are easily defined in `Cubical Agda` for fixed n, but for general $n \geq -2$ we rely on the "hub and spoke" construction [38, Section 7.3].[1] This construction introduces a point constructor $|\_| : A \to \| A \|_n$ and constructors hub and spoke ensuring that any map $\mathbb{S}^{n+1} \to \| A \|_n$ is constant (thus contracting $\Omega^{n+1} \| A \|_n$). Using pattern-matching, we can define the usual elimination principle which says: given $B : \| A \|_n \to n\text{-Type}$, in order to construct an element of type $B\ x$, we may assume that x is of the form $|a|$ for some $a : A$. This extends to paths $p : |x| \equiv |y|$ in $\| A \|_{n+1}$. Suppose we have $B : |x| \equiv |y| \to n\text{-Type}$ and want to construct $B\ p$. The elimination principle tells us that it suffices to do so when $p = \mathsf{cong}\ |\_|\ q$ for $q : x \equiv y$ in A. This is motivated by [38, Theorem 7.3.12].

Truncations allow us to talk about how connected a type is.

▶ **Definition 4** (Connectedness). *A type A is n-connected if $\| A \|_n$ is contractible.*

Connectedness expresses in particular that $|x| \equiv |y|$ holds in $\| A \|_n$ for all $x, y : A$ of an n-connected type A. This enables applications of the induction principle for truncated path spaces discussed above. Most types in this paper are 0-connected. For such types, we can assume that $x \equiv y$ holds for $x, y : A$ whenever we are proving a family of propositions.

Another important class of HITs are pushouts. These correspond to homotopy pushouts in topology. Given $f : A \to B$ and $g : A \to C$, the pushout of the span $B \xleftarrow{f} A \xrightarrow{g} C$ is:

```
data Pushout (f : A → B) (g : A → C) : Type where
  inl : B → Pushout f g
  inr : C → Pushout f g
  push : (a : A) → inl (f a) ≡ inr (g a)
```

Many types that we have seen so far can be defined as pushouts. For instance, Susp A is equivalent to the pushout of the span $\mathbb{1} \leftarrow A \to \mathbb{1}$. Another example is wedge sums:

▶ **Definition 5** (Wedge sums). *Given pointed types A and B, the* wedge sum $A \vee B$ *is the pushout of the span $A \xleftarrow{\lambda\, x \to *_A} \mathbb{1} \xrightarrow{\lambda\, x \to *_B} B$. This is pointed by* inl $*_A$.

[1] For $n = -2$ this construction fails. In this case, simply let $\| A \|_{-2} = \mathbb{1}$ where $\mathbb{1}$ is the unit type.

2.3 Univalence

One of the most important notions in HoTT/UF is Voevodsky's univalence axiom [41]. Informally, this postulates that for all types A and B, there is a term

$$\mathsf{univalence} : (A \simeq B) \simeq (A \equiv B)$$

Here, $A \simeq B$ is the type of functions $e : A \to B$ equipped with a proof that the fiber/preimage of e is contractible at every $x : B$ [38, Chapter 4.4]. This axiom is a provable theorem in `Cubical Agda` using the Glue types of [14, Section 6]. This gives a function $\mathsf{ua} : A \simeq B \to A \equiv B$ which converts equivalences to paths. Transporting along a path constructed using ua applies the function e of the underlying equivalence.

Equivalences $A \simeq B$ are often constructed by exhibiting functions $f : A \to B$ and $g : B \to A$ together with proofs that they cancel. Such a quadruple is referred to as a *quasi-equivalence* in [38]. It is a corollary of [38, Theorem 4.4.5] that all quasi-equivalences can be promoted to equivalences. This fact is used throughout the formalization and paper.

An important consequence of univalence is that it also applies to *structured* types. A structure on types is simply a function $S : \mathsf{Type} \to \mathsf{Type}$. By taking the dependent sum of this, one obtains types with S-structures as pairs $(A\,, s) : \Sigma_{A:\mathsf{Type}}\,(S\ A)$. One example is the type of groups. This is defined as $(G\,, \mathsf{isGroup}\ G)$, where $\mathsf{isGroup}\ G$ is a structure which consists of proofs that G is a set, is pointed by some $0_G : G$, admits a binary operation $+_G$, and satisfies the usual group laws. In [2], a notion of *univalent* structure and structure preserving isomorphisms $\cong$, for which it is direct to prove that ua induces a function $\mathsf{sip} : A \cong B \to A \equiv B$, are introduced in `Cubical Agda`. This is one way to formalize the informal *Structure Identity Principle* (SIP) [38, Section 9.8]. One can show that isGroup (with group isomorphism) is a univalent structure and that equivalences $e : G \simeq H$ sending $+_G$ to $+_H$ preserve this structure. In other words: sip implies that isomorphic groups are path-equal.

3 Integral cohomology in Cubical Agda

In classical mathematics, the nth cohomology group with coefficients in an abelian group G of a CW-complex X may be characterized as the group of homotopy classes of functions $X \to K(G, n)$. Here, $K(G, n)$ denotes the nth *Eilenberg-MacLane space* of G. That is, $K(G, n)$ is the unique space with a single non-trivial homotopy group isomorphic to G, i.e. $\pi_n(K(G, n)) \cong G$ and $\pi_m(K(G, n)) \cong \mathbb{1}$ for $m \neq n$. While this is a theorem in classical mathematics, we take it as our definition of the nth cohomology group of a type A:

$$H^n(A; G) = \| A \to K(G, n) \|_0$$

This type will inherit the group structure[2] from $K(G, n)$ and the goal of this section is to define this explicitly when $G = \mathbb{Z}$. The group structure which we will define here differs (intensionally) from previous variations in that it is optimized for efficient computations.

3.1 Eilenberg-MacLane spaces

The family of spaces $K(G, n)$ was constructed as a HIT and proved to be an n-truncated and $(n-1)$-connected pointed type by Licata and Finster [26]. In this paper, we focus on the case $G = \mathbb{Z}$ and define this special case following Brunerie [5, Def. 5.1.1]:

[2] Technically, $K(G, n)$ is a *higher* group – it is not a set, but satisfies all other group axioms.

▶ **Definition 6.** *The nth Eilenberg-MacLane space of* $\mathbb{Z}$*, written* K_n*, is a pointed type:*

$$\mathsf{K}_n = \begin{cases} (\mathbb{Z}, 0) & \text{if } n = 0 \\ (\| \mathbb{S}^n \|_n, | \ast_{\mathbb{S}^n} |) & \text{if } n \geq 1 \end{cases}$$

We write $H^n(A)$ for $H^n(A; \mathbb{Z})$ with K_n for $K(\mathbb{Z}, n)$. The type K_n is clearly n-truncated and the fact that it is $(n-1)$-connected follows from the following proposition.

▶ **Proposition 7.** $\mathbb{S}^n$ *is* $(n-1)$*-connected for* $n : \mathbb{N}$*.*

Proof. By the definition of $(n-1)$-truncation, the map $|\_| : \mathbb{S}^n \to \| \mathbb{S}^n \|_{n-1}$ is constant. Hence, $\| \mathbb{S}^n \|_{n-1}$ has a trivial constructor and must be contractible. ◀

Note that, in particular, K_n is 0-connected for $n > 0$; it is an easy lemma that any m-connected type is also k-connected for $k < m$. Alternatively, one may prove 0-connectedness of K_n directly by truncation elimination and sphere elimination.

The above proof is much more direct than the one in [5, Prop. 2.4.2] which relies on general results about connectedness of pushouts. The reason we prefer this more direct, but less general proof, is that it computes much faster. The problem seems to be that the general theory of connectedness heavily uses univalence. In particular, it relies on repeated use of [38, Thm. 7.3.12] which says that the type of paths $|x| \equiv |y|$ over $\| A \|_{n+1}$ is equivalent to the type of truncated paths $\| x \equiv y \|_n$.

A more substantial deviation from [5] is in the definition of the group structure on K_n. This is defined in [5, Prop. 5.1.4] using $\mathsf{K}_n \simeq \Omega \mathsf{K}_{n+1}$ which itself is proved using the Hopf fibration [38, Section 8.5] when $n = 1$ and the Freudenthal suspension theorem [38, Section 8.6] when $n \geq 2$. This gives rather indirect definitions of addition and negation on K_n by going through $\Omega \mathsf{K}_{n+1}$. It turns out that these indirect definitions lead to slow computations [28]. To circumvent this, we give a direct definition of the group structure on K_n which in turn gives a direct proof that $\mathsf{K}_n \simeq \Omega \mathsf{K}_{n+1}$ inspired by the proof that $\Omega \mathbb{S}^1 \simeq \mathbb{Z}$ of Licata and Shulman [27]. The strategy of first defining the group structure on K_n to then prove that $\Omega \mathsf{K}_{n+1} \simeq \mathsf{K}_n$ is similar to the one for proving the corresponding statements for general $K(G, n)$ in [26]. However, we deviate in that we avoid the Freudenthal suspension theorem and theory about connectedness.

The neutral element of K_n is $\ast_{\mathsf{K}_n}$ and we denote it by 0_k. In order to prove that K_n is a (higher) group, we first define addition $+_k : \mathsf{K}_n \to \mathsf{K}_n \to \mathsf{K}_n$. The following lemma is the key for doing this. It is a special case of [38, Lemma 8.6.2], but the proof does not rely on general theory about connected types.

▶ **Lemma 8.** *Let* $n, m \geq 1$ *and suppose we have a fibration* $P : \mathbb{S}^n \times \mathbb{S}^m \to (n+m-2)$-Type *together with functions*

$$\mathsf{f}_l : (x : \mathbb{S}^n) \to P\,(x, \ast_{\mathbb{S}^m}) \qquad \mathsf{f}_r : (y : \mathbb{S}^m) \to P\,(\ast_{\mathbb{S}^n}, y)$$

and a path $p : \mathsf{f}_l\ \ast_{\mathbb{S}^n} \equiv \mathsf{f}_r\ \ast_{\mathbb{S}^m}$*. There is a function* $f : (z : \mathbb{S}^n \times \mathbb{S}^m) \to P\,z$ *with paths*

$$\mathsf{left} : (x : \mathbb{S}^n) \to \mathsf{f}_l\ x \equiv \mathsf{f}\,(x, \ast_{\mathbb{S}^m}) \qquad \mathsf{right} : (y : \mathbb{S}^m) \to \mathsf{f}_r\ y \equiv \mathsf{f}\,(\ast_{\mathbb{S}^n}, y)$$

such that $p \equiv \mathsf{left}\ \ast_{\mathbb{S}^n} \cdot (\mathsf{right}\ \ast_{\mathbb{S}^m})^{-1}$*. Furthermore, either* left *or* right *holds definitionally (modulo pattern matching on* n *and* m*).*

Proof. By sphere induction on both $\mathbb{S}^n$ and $\mathbb{S}^m$. For details, see the formalization. ◀

The general version of Lemma 8 is used for $K(G,n)$ in [26]. The advantage of the above form is the definitional reductions which follow from use of sphere induction in its proof. Consequently, we may define $+_k$ so that e.g. $0_k +_k |x| \equiv |x|$ holds definitionally. This allows for statements and proofs which would otherwise not be well-typed.

We define $+_k : \mathsf{K}_n \to \mathsf{K}_n \to \mathsf{K}_n$ and $-_k : \mathsf{K}_n \to \mathsf{K}_n$ by cases on n. When $n = 0$, these are integer addition and negation. Otherwise, we consider the following cases:

- When $n = 1$, we define $+_k$ and $-_k$ by cases:

$$
\begin{aligned}
&|\mathsf{base}| +_k |x| \;\; = |x| && -_k |\mathsf{base}| = |\mathsf{base}| \\
&|\mathsf{loop}\, i| +_k |\mathsf{base}| = |\mathsf{loop}\, i| && -_k |\mathsf{loop}\, i| = |\mathsf{loop}\, (\sim i)| \\
&|\mathsf{loop}\, i| +_k |\mathsf{loop}\, j| = |\mathsf{Q}\, i\, j|
\end{aligned}
$$

  where Q is a suitable filler of a square with loop on all sides. The filler Q is easily defined by an hcomp so that $\mathsf{cong}_2\ (\lambda\, x\, y \to |x| +_k |y|)\ \mathsf{loop}\ \mathsf{loop} \equiv \mathsf{cong}\ |\_|\ (\mathsf{loop} \cdot \mathsf{loop})$ holds definitionally. We will, from now on, with some abuse of notation, simply write loop for the canonical loop in K_1, i.e. $\mathsf{cong}\ |\_|\ \mathsf{loop}$.
- When $n \geq 2$, we need to construct a map $\mathbb{S}^n \times \mathbb{S}^n \to \mathsf{K}_n$ to define addition. Because K_n is n-truncated, it is also an $(n+n-2)$-Type. By Lemma 8, we are done if we can provide two maps $\mathbb{S}^n \to \mathsf{K}_n$ and prove that they agree on $*_{\mathbb{S}^n}$. In both cases we choose the truncation map $\lambda\, x \to |x|$. We then just need to prove that $|*_{\mathbb{S}^n}| \equiv |*_{\mathbb{S}^n}|$, which we do by refl.

  To construct $-_k$, we send $|\mathsf{north}|$ and $|\mathsf{south}|$ to 0_k and $|\mathsf{merid}\, a\, i|$ to $\sigma_n\ a\ (\sim i)$. Here, σ_n is the map from the Freudenthal equivalence [38, Section 8.6] defined by:

$$
\begin{aligned}
&\sigma_n : \mathsf{K}_n \to \Omega\, \mathsf{K}_{n+1} \\
&\sigma_n\ |x| = \mathsf{cong}\ |\_|\ (\mathsf{merid}\ x \cdot (\mathsf{merid}\ *_{\mathbb{S}^n})^{-1})
\end{aligned}
$$

The fact that $+_k$ and $-_k$ satisfy the group laws follows from Lemma 8. In fact, all group laws either hold by refl or have proofs that are at least path-equal to refl at 0_k. This in turn simplifies many later proofs and improves the efficiency of computations. We write lUnit_k/rUnit_k for the left/right unit laws and $\mathsf{lCancel}_k$/$\mathsf{rCancel}_k$ for the left/right inverse laws.

The definition of $+_k$ for $n \geq 2$ may seem naive. However, it provably agrees with the definition given in [5, Prop. 5.1.4]. In fact, a simple corollary of Lemma 8 is that there is at most one binary operation on K_n with lUnit_k and rUnit_k such that $\mathsf{lUnit}_k\ 0_k \equiv \mathsf{rUnit}_k\ 0_k$ (i.e. there is at most one *h-structure* [26, Def. 4.1] on K_n). The fact that this is satisfied by $+_k$ holds by refl. The same result was proved for the addition of [5, Prop. 5.1.4] in [28].

The group structure on K_n allows us to extend the usual encode-decode proof that $\mathbb{Z} \simeq \Omega\, \mathbb{S}^1$ (or, equivalently, $\mathsf{K}_0 \simeq \Omega\, \mathsf{K}_1$) to K_n with $n \geq 1$. We should note that a similar proof was used in [26] in order to prove that $G \simeq \pi_1(\mathsf{K}(G,1))$.

▶ **Theorem 9.** $\mathsf{K}_n \simeq \Omega\, \mathsf{K}_{n+1}$

Proof. The proof is a direct encode-decode proof involving $+_k$ and σ_n. As usual, this proof technique uses univalence. The details can be found in the formalization. ◀

In addition to this, the direct definition of $+_k$ gives a short proof that $\Omega\, \mathsf{K}_n$ is commutative.

▶ **Lemma 10.** *For $n \geq 1$ and $p, q : \Omega\, \mathsf{K}_n$, we have $p \cdot q \equiv \mathsf{cong}_2\ +_k\ p\ q$.*

Proof. First, we remark that the statement is well-typed due to the definitional equality $0_k +_k 0_k \equiv 0_k$. Recall, $p, q : 0_k \equiv 0_k$ and $\mathsf{cong}_2\ +_k\ p\ q$ is of type $0_k +_k 0_k \equiv 0_k +_k 0_k$.

Using this definitional equality, we may apply rUnit_k and lUnit_k pointwise to p and q:

$$p \equiv \mathsf{cong}\ (\lambda x\ \to\ x +_k 0_k)\ p \qquad q \equiv \mathsf{cong}\ (\lambda y\ \to\ 0_k +_k y)\ q$$

By functoriality of cong_2, we get

$$p \cdot q \equiv \mathsf{cong}\ (\lambda x \to x +_k 0_k)\ p \cdot \mathsf{cong}\ (\lambda y \to 0_k +_k y)\ q \equiv \mathsf{cong}_2 +_k p\ q$$ ◀

▶ **Lemma 11.** *For $n \geq 1$ and $p, q : \Omega\,\mathsf{K}_n$, we have $\mathsf{cong}_2 +_k p\ q \equiv \mathsf{cong}_2 +_k q\ p$.*

Proof. By a very similar argument as in Lemma 10, but using commutativity of $+_k$. ◀

▶ **Theorem 12.** *$\Omega\,\mathsf{K}_n$ is commutative with respect to path composition.*

Proof. As $\mathbb{Z}$ is a set, this is trivial for $n = 0$. For $n \geq 1$ it follows from Lemmas 10 and 11. ◀

An alternative proof of Theorem 12 can be found in [5, Prop. 5.1.4]. In that proof, one first translates $\Omega\,\mathsf{K}_n$ into $\Omega^2\,\mathsf{K}_{n-1}$, applies the Eckmann-Hilton argument and then translates back. This translation back-and-forth is problematic from a computational point of view, and the proof of Theorem 12 is more computationally efficient.

3.2 Group structure on $H^n(A)$

We now return to $H^n(A)$ and define $0_h = |\,\lambda x\ \to\ 0_k\,|$ together with the group operations:

$$|\,f\,| +_h |\,g\,| = |\,\lambda x\ \to\ f\ x +_k g\ x\,| \qquad -_h |\,f\,| = |\,\lambda x\ \to\ -_k\ f\ x\,|$$

The fact that $(H^n(A), 0_h, +_h, -_h)$ forms an abelian group follows immediately from the group laws for K_n and funExt. We have also defined a *reduced* version of our cohomology theory and proved that it satisfies the Eilenberg-Steenrod axioms [16]. We refer the interested reader to the formalization for the statement and verification of these axioms. This allows us to use abstract machinery to characterize cohomology groups of many spaces. However, in order to obtain definitions with good computational properties, we often prefer giving direct characterizations not relying on abstract results.

4 The cup product and cohomology ring

We will now equip the cohomology groups studied in the previous section with a multiplicative structure $\smile : H^n(A) \to H^m(A) \to H^{n+m}(A)$. This operation is called the *cup product* and it turns the $H^n(A)$ into a graded commutative ring $H^*(A)$ called the *cohomology ring* of A.

4.1 Defining the cup product in Cubical Agda

The cup product $\smile$ for $\mathbb{Z}$-cohomology in HoTT/UF was introduced by Brunerie [5, Section 5.1]. The definition is induced from a pointed map $\mathsf{K}_n \wedge \mathsf{K}_m \to_* \mathsf{K}_{n+m}$, where $\wedge$ is the *smash product* HIT. This HIT has proved to be surprisingly complex to reason about formally [6] and we therefore consider an alternative definition of $\smile$. The key observation in this reformulation is the pointed equivalence of $A \wedge B \to_* C$ and $A \to_* B \to_* C$ proved in HoTT/UF by van Doorn [39, Thm 4.3.8]. We hence construct $\smile$ by first defining a pointed map $x \smile_k y : \mathsf{K}_n \to \mathsf{K}_m \to_* \mathsf{K}_{n+m}$ by induction on n, thereby avoiding the smash product. When $n = 0$, this map just adds y to itself x times and similarly when $m = 0$. When $n, m \geq 1$, the key lemma is:

▶ **Lemma 13.** *The type* $\mathsf{K}_m \to_* \mathsf{K}_{n+m}$ *is an* n*-type.*

Proof. This is a special case of [9, Corollary 4.3]. We have formalized a direct proof of this special case which does not rely on any explicit connectedness arguments. ◀

Truncation elimination can hence be applied and we only need to define $|\,a\,| \smile_k y$ for $a : \mathbb{S}^n$.

$\mathbf{n = 1}$:	$\mathbf{n \geq 2}$:
$\lvert\,\mathsf{base}\,\rvert \smile_k y = 0_k$	$\lvert\,\mathsf{north}\,\rvert \smile_k y = 0_k$
$\lvert\,\mathsf{loop}\ i\,\rvert \smile_k y = \sigma_m\ y\ i$	$\lvert\,\mathsf{south}\,\rvert \smile_k y = 0_k$
	$\lvert\,\mathsf{merid}\ a\ i\,\rvert \smile_k y = \sigma_{(n-1)+m}\ (\lvert\,a\,\rvert \smile_k y)\ i$

The fact that $\lambda\, y \to x \smile_k y$ is pointed for $x : \mathsf{K}_n$ follows easily. In addition, we get pointedness in x immediately by construction. With this simple definition, we can now define the cup product $\smile : H^n(A) \to H^m(A) \to H^{n+m}(A)$ analogously to $+_h$ by:

$$|\,f\,| \smile |\,g\,| = |\,\lambda\, x \to f\ x \smile_k g\ x\,|$$

4.2 The cohomology ring

We will now prove that $\smile$ turns $H^n(A)$ into a graded ring. First of all, as $\smile_k$ is pointed in both arguments, we get that $x \smile 0_h \equiv 0_h \equiv 0_h \smile y$. Furthermore, it is easy to see that $1_h = |\,\lambda\, x \to 1\,|$ in $H^0(A)$ is a unit for $\smile$. The key lemma for proving properties of $\smile_k$ is:

▶ **Lemma 14.** *Given a pointed type* A *and two pointed functions* $(f, p), (g, q) : A \to_* \mathsf{K}_n$*, we have that if* $f \equiv g$ *then* $(f, p) \equiv (g, q)$*.*

Proof. This is proved using a notion of *homogeneous* types, see the formalization. ◀

In order to increase readability, we omit transports in Propositions 15, 17, and 18. We first verify that $\smile_k$ distributes over $+_k$.

▶ **Proposition 15.** *For* $z : \mathsf{K}_n$ and $x, y : \mathsf{K}_m$, we have $z \smile_k (x +_k y) \equiv z \smile_k x +_k z \smile_k y$ and $(x +_k y) \smile_k z \equiv x \smile_k z +_k y \smile_k z$.

Proof. We sketch the proof for left distributivity and focus on the case when $n, m \geq 1$. We want to show that $\lambda\, z \to z \smile_k (x +_k y)$ and $\lambda\, z \to z \smile_k x +_k z \smile_k y$ are equal *as pointed functions*. This allows for truncation elimination on x and y by Lemma 13. Thus we want to show that $z \smile_k (|\,a\,| +_k |\,b\,|) \equiv z \smile_k |\,a\,| +_k z \smile_k |\,b\,|$ for $a, b : \mathbb{S}^m$. We are proving an $(m-1)$-type and Lemma 8 applies. Hence we need to construct

$$f_l : (a : \mathbb{S}^n) \to z \smile_k (|\,a\,| +_k 0_k) \equiv z \smile_k |\,a\,| +_k z \smile_k 0_k$$

$$f_r : (b : \mathbb{S}^m) \to z \smile_k (0_k +_k |\,b\,|) \equiv z \smile_k 0_k +_k z \smile_k |\,b\,|$$

such that $f_l(*_{\mathbb{S}^n}) \equiv f_r(*_{\mathbb{S}^m})$. By Lemma 14, we only need to construct f_l and f_r for the underlying functions. We get f_l and f_r by applications of lUnit_k/rUnit_k and the law of right multiplication by 0_k. Due to definitional equalities at 0_k, $f_l(*_{\mathbb{S}^n}) \equiv f_r(*_{\mathbb{S}^m})$ holds by refl. ◀

In order to prove that $\smile_k$ is associative, we need the following lemma:

▶ **Lemma 16.** *Let* $n, m \geq 1$*. For* $x : \mathsf{K}_n$ *and* $y : \mathsf{K}_m$*,* $\sigma_{n+m}(x \smile_k y) \equiv \mathsf{cong}\,(\smile_k y)\,(\sigma_n\ x)$.

Lemma 16 occurs in [5, Prop. 6.1.1], albeit for a different definition of $\smile$. Interestingly, Brunerie does not use it to prove associativity of $\smile_k$, but to construct the *Gysin sequence*.

▶ **Proposition 17.** *For* $x : \mathsf{K}_n$, $y : \mathsf{K}_m$ and $z : \mathsf{K}_\ell$, we have $x \smile_k (y \smile_k z) \equiv (x \smile_k y) \smile_k z$.

Proof. The proof is easy when one of n, m or ℓ is 0. When $n, m, \ell \geq 1$, we want to show that $\lambda\, z\, y \to x \smile_k (y \smile_k z)$ and $\lambda\, z\, y \to (x \smile_k y) \smile_k z$ are equal as doubly pointed functions, i.e. as terms of $\mathsf{K}_m \to_* \mathsf{K}_\ell \to_* \mathsf{K}_{n+m+\ell}$. This is an n-type by repeated use of Lemma 13 and we may let $x = |\, a \,|$ for $a : \mathsf{K}_n$. We again only need to prove the underlying functions equal. We do this by induction on n. For $n = 1$, the case $a = \mathsf{base}$ holds by refl. In the case $a = \mathsf{loop}\, i$, we need to prove that $\sigma_{m+\ell}(y \smile_k z) \equiv \mathsf{cong}\, (\smile_k z)\, (\sigma_m\, y)$ which is Lemma 16. The $n \geq 2$ case follows by an analogous argument using the inductive hypothesis. ◀

Finally, we can verify that $\smile_k$ is graded commutative.

▶ **Proposition 18.** *For* $x : \mathsf{K}_n$ and $y : \mathsf{K}_m$, we have $x \smile_k y \equiv \text{-}_k{}^{m \cdot n}\, (y \smile_k x)$.

Proof. The proof is by induction on n and m. Due to space constraints, we omit the base cases and focus on the inductive step where $n, m \geq 2$ (the case $n = 1$ and $m \geq 1$ is close to identical). We may assume as our inductive hypothesis that the statement holds for all $n', m' : \mathbb{N}$ such that $n' + m' < n + m$. The proof boils down to showing that

$$\lambda\, i\, j \to |\, \mathsf{merid}\, a\, i \,| \smile_k |\, \mathsf{merid}\, b\, j \,| \equiv \lambda\, i\, j \to \text{-}_k{}^{m \cdot n}\ (|\, \mathsf{merid}\, b\, j \,| \smile_k |\, \mathsf{merid}\, a\, i \,|)$$

ignoring coherence paths and transports. Here, $a : \mathbb{S}^{n-1}$ and $b : \mathbb{S}^{m-1}$. We fix i and j and give a rough outline of the argument. We have:

$$\begin{aligned}
|\, \mathsf{merid}\, a\, i \,| \smile_k |\, \mathsf{merid}\, b\, j \,| &\equiv \sigma_{n+m-1}(|\, a \,| \smile_k |\, \mathsf{merid}\, b\, j \,|)\, i \\
&\equiv \text{-}_k{}^{m \cdot (n-1)}\, (\sigma_{n+m-1}(|\, \mathsf{merid}\, b\, j \,| \smile_k |\, a \,|)\, i) \\
&\equiv \text{-}_k{}^{m \cdot (n-1)}\, (\sigma_{n+m-1}(\sigma_{n+m-2}(|\, b \,| \smile_k |\, a \,|)\, j)\, i) \\
&\equiv \text{-}_k{}^{m \cdot (n-1)} \text{-}_k{}^{(n-1) \cdot (m-1)}\, (\sigma_{n+m-1}(\sigma_{n+m-2}(|\, a \,| \smile_k |\, b \,|)\, j)\, i) \\
&\equiv \text{-}_k{}^{n+1}\, (\sigma_{n+m-1}(\sigma_{n+m-2}(|\, a \,| \smile_k |\, b \,|)\, j)\, i) \\
&\equiv \text{-}_k{}^{n+1}\, (\sigma_{n+m-1}(|\, \mathsf{merid}\, a\, j \,| \smile_k |\, b \,|)\, i) \\
&\equiv \text{-}_k{}^{n+1} \text{-}_k{}^{(m-1) \cdot n}\, (\sigma_{n+m-1}(|\, b \,| \smile_k |\, \mathsf{merid}\, a\, j \,|)\, i) \\
&\equiv \text{-}_k{}^{n+1} \text{-}_k{}^{(m-1) \cdot n}\, (|\, \mathsf{merid}\, b\, i \,| \smile_k |\, \mathsf{merid}\, a\, j \,|) \\
&\equiv \text{-}_k{}^{m \cdot n+1}\, (|\, \mathsf{merid}\, b\, i \,| \smile_k |\, \mathsf{merid}\, a\, j \,|) \\
&\equiv \text{-}_k{}^{m \cdot n}\, (|\, \mathsf{merid}\, b\, j \,| \smile_k |\, \mathsf{merid}\, a\, i \,|)
\end{aligned}$$

The above chain of equalities repeatedly uses that for a path $p : \Omega^2 A$, we have $(\lambda\, i\, j \to p\, j\, i) \equiv p^{-1}$, and for a path $q : \Omega\, \mathsf{K}_n$, we have $\mathsf{cong}\ \text{-}_k\ q \equiv q^{-1}$. The remaining steps are just unfoldings of the definition of $\smile_k$ and applications of the inductive hypothesis and $\sigma_n(\text{-}_k\, x) \equiv \mathsf{cong}\ \text{-}_k\ (\sigma_n\, x)$. ◀

Although this informal argument is fairly direct, the formal version is much more technical as we also have to verify that the proof sketched above respects the boundary constraints (i.e. our choices of paths for the point constructors). As we also need to express many of these equalities using PathP or $\mathsf{transport}$ (over paths in $\mathbb{N}$), things become even more complicated.

The cup product $\smile$ inherits the properties of $\smile_k$ and we can hence organize $H^n(A)$ into a graded commutative ring $H^*(A)$.

5 Characterizing integral cohomology groups

We will now characterize $H^n(A)$ for A being the spheres, torus, Klein bottle, and real/complex projective planes. The cases when $H^n(A) \simeq \mathbb{1}$ for $n \geq 1$ are easy using connectedness arguments (see the formalization). It is also an easy lemma that $H^0(X) \simeq \mathbb{Z}$ if X is 0-connected, which is the case for all types considered here. The main focus in this section will hence be on the non-trivial $H^n(A)$ with $n \geq 1$. Furthermore, we only focus on the equivalence parts of the characterizations, but we emphasize that all cases, including homomorphism proofs, have been formalized.

5.1 Spheres

The key to characterizing the cohomology groups of $\mathbb{S}^n$ is the **Suspension** axiom for cohomology. This axiom says that $H^n(A) \simeq H^{n+1}(\mathsf{Susp}\ A)$ and a proof can be found in the formalization. Recall that $\mathbb{S}^{m+1} = \mathsf{Susp}\,\mathbb{S}^m$ for $m \geq 1$ and thus we have that $H^{n+1}(\mathbb{S}^{m+1}) \simeq H^n(\mathbb{S}^m)$.

▶ **Proposition 19.** *$H^n(\mathbb{S}^n) \simeq \mathbb{Z}$ for $n \geq 1$.*

Proof. By **Suspension** and induction, it suffices to consider the $n = 1$ case. We inspect the underlying function space of $H^1(\mathbb{S}^1)$, i.e. $\mathbb{S}^1 \to \mathsf{K}_1$. A map $f : \mathbb{S}^1 \to \mathsf{K}_1$ is uniquely determined by $f\ \mathsf{base} : \mathsf{K}_1$ and $\mathsf{cong}\ f\ \mathsf{loop} : f\ \mathsf{base} \equiv f\ \mathsf{base}$. Thus, we have $H^1(\mathbb{S}^1) \simeq \| \sum_{x:\mathsf{K}_1} x \equiv x \|_0$. By a base change we get $(x \equiv x) \simeq (0_k \equiv 0_k)$ for any $x : \mathsf{K}_1$. Hence

$$H^1(\mathbb{S}^1) \simeq \| \mathsf{K}_1 \times \Omega\,\mathsf{K}_1 \|_0 \simeq \| \mathsf{K}_1 \|_0 \times \| \Omega\,\mathsf{K}_1 \|_0 \simeq \| \Omega\,\mathsf{K}_1 \|_0 \simeq \| \Omega\,\mathbb{S}^1 \|_0 \simeq \mathbb{Z} \qquad \blacktriangleleft$$

5.2 The torus

The torus HIT, $\mathbb{T}^2$, is defined as follows:

```
data 𝕋² : Type where
  pt : 𝕋²
  ℓ₁ ℓ₂ : pt ≡ pt
  □ : PathP (λ i → ℓ₂ i ≡ ℓ₂ i) ℓ₁ ℓ₁
```

The constructor $\square$ corresponds to the usual gluing diagram for constructing the torus in classical topology as it identifies ℓ_1 with itself over an identification of ℓ_2 with itself. As discussed in the introduction, proving $\mathbb{T}^2 \simeq \mathbb{S}^1 \times \mathbb{S}^1$ is easy in `Cubical Agda`. This allows us to curry $\mathbb{T}^2 \to \mathsf{K}_n$, which is the key step to prove Propositions 20 and 21.

▶ **Proposition 20.** $H^1(\mathbb{T}^2) \simeq \mathbb{Z} \times \mathbb{Z}$

Proof. We inspect the underlying function space $\mathbb{T}^2 \to \mathsf{K}_1$, which is equivalent to $\mathbb{S}^1 \to (\mathbb{S}^1 \to \mathsf{K}_1)$. From Proposition 19, we know that $(\mathbb{S}^1 \to \mathsf{K}_1) \simeq \mathsf{K}_1 \times \Omega\,\mathsf{K}_1 \simeq \mathsf{K}_1 \times \mathbb{Z}$. Hence

$$H^1(\mathbb{T}^2) \simeq \| \mathbb{S}^1 \to \mathsf{K}_1 \times \mathbb{Z} \|_0 \simeq \| \mathbb{S}^1 \to \mathsf{K}_1 \|_0 \times \| \mathbb{S}^1 \to \mathbb{Z} \|_0 \overset{\text{def}}{\equiv} H^1(\mathbb{S}^1) \times H^0(\mathbb{S}^1) \simeq \mathbb{Z} \times \mathbb{Z} \qquad \blacktriangleleft$$

▶ **Proposition 21.** $H^2(\mathbb{T}^2) \simeq \mathbb{Z}$

Proof. The underlying function space, post currying, is $\mathbb{S}^1 \to (\mathbb{S}^1 \to \mathsf{K}_2)$. Like above, this is $(\mathbb{S}^1 \to \mathsf{K}_2 \times \Omega\,\mathsf{K}_2) \simeq (\mathbb{S}^1 \to \mathsf{K}_2 \times \mathsf{K}_1) \simeq (\mathbb{S}^1 \to \mathsf{K}_2) \times (\mathbb{S}^1 \to \mathsf{K}_1)$. Hence

$$H^2(\mathbb{T}^2) \simeq \| (\mathbb{S}^1 \to \mathsf{K}_2) \times (\mathbb{S}^1 \to \mathsf{K}_1) \|_0 \simeq H^2(\mathbb{S}^1) \times H^1(\mathbb{S}^1) \simeq \mathbb{Z} \qquad \blacktriangleleft$$

5.3 The Klein bottle and real projective plane

The Klein bottle and real projective plane are also HITs, but with twists in □ just like in the classical gluing diagrams:

```
data 𝕂² : Type where
  pt : 𝕂²
  ℓ₁ ℓ₂ : pt ≡ pt
  □ : PathP (λ i → ℓ₂ (~ i) ≡ ℓ₂ i) ℓ₁ ℓ₁
```

```
data ℝP² : Type where
  pt : ℝP²
  ℓ : pt ≡ pt
  □ : ℓ ≡ ℓ ⁻¹
```

Note that □ for $\mathbb{K}^2$ equivalently may be interpreted as the path $\ell_2 \cdot \ell_1 \cdot \ell_2 \equiv \ell_1$. To characterize the cohomology groups of $\mathbb{K}^2$, we need to understand their underlying function spaces. It is easy to see that

$$(\mathbb{K}^2 \to \mathsf{K}_n) \simeq \sum_{x:\mathsf{K}_n} \sum_{p,q:x \equiv x} (p \cdot q \cdot p \equiv q)$$

By Theorem 12, $\_\cdot\_$ in $\Omega\,\mathsf{K}_n$ is commutative, so $(p \cdot q \cdot p \equiv q) \simeq (p \cdot p \equiv \mathsf{refl})$. Hence

$$(\mathbb{K}^2 \to \mathsf{K}_n) \simeq \sum_{x:\mathsf{K}_n} \left((x \equiv x) \times \sum_{p:x \equiv x} (p \cdot p \equiv \mathsf{refl}) \right) \tag{1}$$

▶ **Proposition 22.** $H^1(\mathbb{K}^2) \simeq \mathbb{Z}$

Proof. Note that for $x : \mathsf{K}_1$, we have that $\sum_{p:x \equiv x}(p \cdot p \equiv \mathsf{refl}) \simeq \sum_{a:\mathbb{Z}}(a + a \equiv 0) \simeq \mathbb{1}$. This allows us to simplify (1) and get

$$H^1(\mathbb{K}^2) \simeq \| \mathbb{K}^2 \to \mathsf{K}_1 \|_0 \simeq \| \sum_{x:\mathsf{K}_1} (x \equiv x) \|_0 \simeq H^1(\mathbb{S}^1) \simeq \mathbb{Z}$$ ◀

▶ **Lemma 23.** *For $n : \mathbb{Z}$, define $p : \| \sum_{x:\mathsf{K}_1}(x +_k x \equiv 0_k) \|_0$ by $p = |\,(0_k\,, \mathsf{loop}^n)\,|$. We have $p \equiv |\,(0_k\,, \mathsf{refl})\,|$ if n is even and $p \equiv |\,(0_k\,, \mathsf{loop})\,|$ if n is odd.*

▶ **Proposition 24.** $H^2(\mathbb{K}^2) \simeq \mathbb{Z}/2\mathbb{Z}$

Proof. Using 0-connectedness of K_2 and $(x \equiv x)$ for $x : \mathsf{K}_2$, it is easy to see that, by truncating both sides of (1), we get

$$H^2(\mathbb{K}^2) \simeq \| \sum_{p:\Omega\,\mathsf{K}_2} (p \cdot p \equiv \mathsf{refl}) \|_0$$

Using the equivalence $\Omega\,\mathsf{K}_2 \simeq \mathsf{K}_1$ and the fact that it takes path composition to addition, this can be further simplified to $\| \sum_{x:\mathsf{K}_1}(x +_k x \equiv 0_k) \|_0$. It is easy to see that for any $p : \| \sum_{x:\mathsf{K}_1}(x +_k x \equiv 0_k) \|_0$, we have that $p \equiv |\,(0_k\,, \mathsf{loop}^n)\,|$ for some $n : \mathbb{N}$. We map p into $\mathbb{Z}/2\mathbb{Z}$ by sending it to 0 if n is even and 1 if n is odd. As an immediate consequence of Lemma 23, this map must be an equivalence, and thus we are done. ◀

The attentive reader will have noticed that something reminiscent of the real projective plane, $\mathbb{R}P^2$, appears in both proofs in this section. We characterize $H^n(\mathbb{R}P^2)$ for $n \geq 1$ by

$$\| \mathbb{R}P^2 \to \mathsf{K}_n \|_0 \simeq \| \sum_{x:\mathsf{K}_n} \sum_{p:x \equiv x} (p \equiv p^{-1}) \|_0 \simeq \| \sum_{x:\mathsf{K}_n} \sum_{p:x \equiv x} (p \cdot p \equiv \mathsf{refl}) \|_0 \simeq \| \sum_{p:\Omega\,\mathsf{K}_n} (p \cdot p \equiv \mathsf{refl}) \|_0$$

When n is 1 or 2, this is precisely one of the types appearing in the proofs of Propositions 22 and 24 respectively, so $H^1(\mathbb{R}P^2) \simeq \mathbb{1}$ and $H^2(\mathbb{R}P^2) \simeq \mathbb{Z}/2\mathbb{Z}$.

5.4 The complex projective plane

We define the complex projective plane, $\mathbb{C}P^2$, as the pushout of the span $\mathbb{S}^2 \xleftarrow{h} \mathbb{S}^3 \to \mathbb{1}$ where h is part of the Hopf fibration [38, Section 8.5]. The function space $\mathbb{C}P^2 \to \mathsf{K}_n$ is quite hard to work with directly, so we settle for an indirect characterization of $H^n(\mathbb{C}P^2)$ via the Mayer-Vietoris sequence (see the formalization). For $n \geq 2$, this gives us an exact sequence:

$$H^{n-1}(\mathbb{S}^2) \to H^{n-1}(\mathbb{S}^3) \to H^n(\mathbb{C}P^2) \to H^n(\mathbb{S}^2) \to H^n(\mathbb{S}^3)$$

For $n \in \{3, 5, 6, \dots\}$, we have that $H^n(\mathbb{C}P^2) \simeq \mathbb{1}$, as other groups in the sequence become trivial. When $n = 2$, all groups but $H^2(\mathbb{S}^2)$ are trivial, and hence $H^2(\mathbb{C}P^2) \simeq H^2(\mathbb{S}^2) \simeq \mathbb{Z}$. When $n = 4$, the only non trivial group is $H^3(\mathbb{S}^3)$, and hence we get $H^4(\mathbb{C}P^2) \simeq H^3(\mathbb{S}^3) \simeq \mathbb{Z}$. A simple connectedness argument finally gives us that $H^1(\mathbb{C}P^2) \simeq \mathbb{1}$.

6 Proving by computations in Cubical Agda

One of the appealing aspects of developing cohomology theory in `Cubical Agda` is that we can prove properties purely by computation. This can discharge proof goals involving complex path algebra as soon as the types are fully instantiated. For example, in Proposition 18 when $m = n = 1$, the main subgoal involves compositions paths in $\Omega^2 \mathsf{K}_2$ which can be reduced to a computation purely involving $\mathbb{Z}$, using the equivalence $\Omega^2 \mathsf{K}_2 \simeq \mathbb{Z}$. As we have been careful about proving things as directly as possible with efficient computations in mind, this works quite well, but there are some cases which are surprisingly slow in `Cubical Agda`, and we have collected some benchmarks at `https://github.com/agda/cubical/blob/master/Cubical/Experiments/ZCohomology/Benchmarks.agda`.

Furthermore, we can use the fact that the isomorphisms compute to establish that some types cannot be equivalent. This is the case for all spaces in the previous section, as they have different cohomology groups. However, there are some spaces where it is not enough to only look at the cohomology groups. We have proved that our cohomology theory satisfies the **Binary Additivity** axiom which says that $H^n(A \vee B) \simeq H^n(A) \times H^n(B)$. So we can easily prove that $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$ has the same cohomology groups as $\mathbb{T}^2$. However, these two types are not equivalent and the standard way to prove this is to use the cup product. We can do this traditional proof computationally in `Cubical Agda` by defining a predicate $P : \mathsf{Type} \to \mathsf{Type}$ by $P(A) = (x\ y : H^1(A)) \to x \smile y \equiv 0_h$ and show that $P(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$ holds while $P(\mathbb{T}^2)$ does not. In `Cubical Agda`, we have defined isomorphisms:

$$f_1 : H^1(\mathbb{T}^2) \cong \mathbb{Z} \times \mathbb{Z} \qquad g_1 : H^1(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \cong \mathbb{Z} \times \mathbb{Z}$$
$$f_2 : H^2(\mathbb{T}^2) \cong \mathbb{Z} \qquad g_2 : H^2(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \cong \mathbb{Z}$$

To disprove $P(\mathbb{T}^2)$ we need $x, y : H^1(\mathbb{T}^2)$ such that $x \smile y \not\equiv 0_h$. Let $x = f_1^{-1}(0, 1)$ and $y = f_1^{-1}(1, 0)$. In `Cubical Agda`, $f_2(x \smile y) \equiv 1$ holds by `refl` and thus $x \smile y \not\equiv 0_h$. It remains to prove $P(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$. Let $x, y : H^1(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$. In `Cubical Agda`, we have that $g_2(g_1^{-1}(g_1\ x) \smile g_1^{-1}(g_1\ y)) \equiv 0$, again by `refl`, and thus $g_1^{-1}(g_1\ x) \smile g_1^{-1}(g_1\ y) \equiv x \smile y \equiv 0_h$.

For a more ambitious example, consider [5, Chapter 6]. This is devoted to proving, using sophisticated techniques like the Gysin sequence, that the generator $e : H^2(\mathbb{C}P^2)$ when multiplied with itself yields a generator of $H^4(\mathbb{C}P^2)$. Let $g : \mathbb{Z} \to \mathbb{Z}$ be the map described by

$$\mathbb{Z} \xrightarrow{\cong} H^2(\mathbb{C}P^2) \xrightarrow{\lambda\, x \to x \smile x} H^4(\mathbb{C}P^2) \xrightarrow{\cong} \mathbb{Z}$$

The number $g(1)$ should reduce to ± 1 for $e \smile e$ to generate $H^4(\mathbb{C}P^2)$ and by evaluating it in `Cubical Agda` we should be able to reduce the whole chapter to a single computation. However, `Cubical Agda` is currently stuck on computing $g(1)$. This number can hence be

seen as another "Brunerie number" – a mathematically interesting number which is currently infeasible to compute using an implementation of cubical type theory. This computation should be more feasible than the original Brunerie number. As our definition of $\smile$ produces very simple terms, most of the work has to occur in the two isomorphisms, and we are optimistic that future optimizations will allow us to perform this computation.

7 Conclusions

We have developed multiple classical results from cohomology theory synthetically in `Cubical Agda`. This has led to new and more direct constructive proofs than what already existed in the HoTT/UF literature. Furthermore, Section 4 contains the first fully formalized verification of the graded commutative ring axioms for $\mathbb{Z}$-cohomology. The key to this is the new definition of $\smile$ which avoids the smash product. The synthetic characterizations of the cohomology groups of $\mathbb{K}^2$ and $\mathbb{R}P^2$ are also novel. The proofs have been constructed with computational efficiency in mind, allowing us to make explicit computations involving several non-trivial cohomology groups. In particular, the number $g(1)$ is another "Brunerie number" which should be more feasible to compute, and its computation would allow us to reduce the complex proofs of [5, Chapter 6] to a single computation. This is hence a new challenge for future improvements of `Cubical Agda` and related systems like `cooltt` [37].

Related and future work

In addition to the related work already mentioned in the paper, there is some related prior work in `Cubical Agda`. Qian [32] formalized $K(G,1)$ as a HIT, following [26], and proved that it satisfies $\pi_1(K(G,1)) \equiv G$. Alfieri [1] and Harington [18] formalized $K(G,1)$ as the classifying space BG using G-torsors. Using this, $H^1(\mathbb{S}^1;\mathbb{Z}) \equiv \mathbb{Z}$ was proved – however, computing using the maps in this definition proved to be infeasible. It is not clear where the bottlenecks are, but we emphasize that with the definitions in this paper, there are no problems computing with this cohomology group.

Certified computations of homology groups using proof assistants have been considered prior to HoTT/UF. For instance, the `Coq` system [36] has been used to compute homology [21] and persistent homology [20] with coefficients in a field. This was later extended to homology with $\mathbb{Z}$-coefficients in [10]. The approach in these papers was entirely algebraic and spaces were represented as simplicial complexes. However, a synthetic approach to homology in HoTT/UF was developed informally by Graham [17] using stable homotopy groups. This was later extended with a proof of Hurewicz theorem by Christensen and Scoccola [13]. It would be interesting to see if this could be made formal in `Cubical Agda` so that we can also characterize and compute with homology groups.

The definition of $H^*(A)$ in HoTT/UF is due to Brunerie [5, Chapter 5.1]. Here, however, $\smile$ relies on the smash product which has proved very complex to reason about formally [6]. Despite this, Baumann generalized this to $H^n(X;G)$ and managed to formalize graded commutativity in `HoTT-Agda` [4]. Baumann's formal proof of this property is ~ 5000 LOC while our formalization is just ~ 900 LOC. This indicates that it would be infeasible to formalize other algebraic properties of $H^*(A)$ with this definition. Associativity seems particularly infeasible, but with our definition the formal proof is only ~ 200 LOC. However, this comparison should be taken with a grain of salt as Baumann proves the result for $H^n(X;G)$. Nevertheless, we conjecture that our constructions should be relatively easy to generalize to cohomology with coefficients in an arbitrary group.

References

1 Victor Alfieri. Formalisation de notions de théorie des groupes en théorie cubique des types, 2019. Internship report, supervised by Thierry Coquand.

2 Carlo Angiuli, Evan Cavallo, Anders Mörtberg, and Max Zeuner. Internalizing representation independence with univalence. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434293.

3 Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, January 2009. doi:10.1017/S0305004108001783.

4 Tim Baumann. The cup product on cohomology groups in homotopy type theory. Master's thesis, University of Augsburg, 2018.

5 Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, Université Nice Sophia Antipolis, 2016. arXiv:1606.05916.

6 Guillaume Brunerie. Computer-generated proofs for the monoidal structure of the smash product. *Homotopy Type Theory Electronic Seminar Talks*, November 2018. URL: https://www.uwo.ca/math/faculty/kapulkin/seminars/hottest.html.

7 Guillaume Brunerie, Kuen-Bang Hou (Favonia), Evan Cavallo, Tim Baumann, Eric Finster, Jesper Cockx, Christian Sattler, Chris Jeris, Michael Shulman, et al. Homotopy Type Theory in Agda, 2018. URL: https://github.com/HoTT/HoTT-Agda.

8 Ulrik Buchholtz and Kuen-Bang Hou Favonia. Cellular Cohomology in Homotopy Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 521–529, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209188.

9 Ulrik Buchholtz, Floris van Doorn, and Egbert Rijke. Higher Groups in Homotopy Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 205–214, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209150.

10 Guillaume Cano, Cyril Cohen, Maxime Dénès, Anders Mörtberg, and Vincent Siles. Formalized Linear Algebra over Elementary Divisor Rings in Coq. *Logical Methods in Computer Science*, 12(2), 2016. doi:10.2168/LMCS-12(2:7)2016.

11 Evan Cavallo. Synthetic Cohomology in Homotopy Type Theory. Master's thesis, Carnegie Mellon University, 2015.

12 Evan Cavallo and Robert Harper. Higher Inductive Types in Cubical Computational Type Theory. *Proceedings of the ACM on Programming Languages*, 3(POPL):1:1–1:27, January 2019. doi:10.1145/3290314.

13 J. Daniel Christensen and Luis Scoccola. The Hurewicz theorem in Homotopy Type Theory, 2020. Preprint. arXiv:2007.05833.

14 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TYPES.2015.5.

15 Thierry Coquand, Simon Huber, and Anders Mörtberg. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS 2018, pages 255–264, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209197.

16 Samuel Eilenberg and Norman Steenrod. *Foundations of Algebraic Topology*. Foundations of Algebraic Topology. Princeton University Press, 1952.

17 Robert Graham. Synthetic Homology in Homotopy Type Theory, 2018. Preprint. arXiv:1706.01540.

18 Elies Harington. Groupes de cohomologie en théorie des types univalente, 2020. Internship report, supervised by Thierry Coquand.

19 Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002. URL: `https://pi.math.cornell.edu/~hatcher/AT/AT.pdf`.

20 Jónathan Heras, Thierry Coquand, Anders Mörtberg, and Vincent Siles. Computing Persistent Homology Within `Coq`/SSReflect. *ACM Transactions on Computational Logic*, 14(4):1–26, 2013. `doi:10.1145/2528929`.

21 Jónathan Heras, Maxime Dénès, Gadea Mata, Anders Mörtberg, María Poza, and Vincent Siles. Towards a Certified Computation of Homology Groups for Digital Images. In *Proceedings of the 4th International Conference on Computational Topology in Image Context*, CTIC'12, pages 49–57, Berlin, Heidelberg, 2012. Springer-Verlag. `doi:10.1007/978-3-642-30238-1_6`.

22 Kuen-Bang Hou (Favonia), Eric Finster, Daniel R. Licata, and Peter LeFanu Lumsdaine. A Mechanization of the Blakers-Massey Connectivity Theorem in Homotopy Type Theory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 565–574, New York, NY, USA, 2016. ACM. `doi:10.1145/2933575.2934545`.

23 Kuen-Bang Hou (Favonia) and Michael Shulman. The Seifert-van Kampen Theorem in Homotopy Type Theory. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CSL.2016.22`.

24 Krzysztof Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky). *Journal of the European Mathematical Society*, 23:2071–2126, March 2021. `doi:10.4171/JEMS/1050`.

25 Daniel R. Licata and Guillaume Brunerie. A Cubical Approach to Synthetic Homotopy Theory. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '15, pages 92–103, Washington, DC, USA, 2015. IEEE Computer Society. `doi:10.1109/LICS.2015.19`.

26 Daniel R. Licata and Eric Finster. Eilenberg-MacLane Spaces in Homotopy Type Theory. In *Proceedings of the 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, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2603088.2603153`.

27 Daniel R. Licata and Michael Shulman. Calculating the Fundamental Group of the Circle in Homotopy Type Theory. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '13, pages 223–232, Washington, DC, USA, 2013. IEEE Computer Society. `doi:10.1109/LICS.2013.28`.

28 Axel Ljungström. Computing Cohomology in `Cubical Agda`. Master's thesis, Stockholm University, 2020.

29 Per Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975. `doi:10.1016/S0049-237X(08)71945-1`.

30 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

31 Anders Mörtberg and Loïc Pujet. Cubical Synthetic Homotopy Theory. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 158–171, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3372885.3373825`.

32 Zesen Qian. Towards Eilenberg-MacLane Spaces in Cubical Type Theory. Master's thesis, Carnegie Mellon University, 2019.

33 Michael Shulman. Cohomology, 2013. Post on the Homotopy Type Theory blog: `http://homotopytypetheory.org/2013/07/24/`.

34 Kristina Sojakova. The Equivalence of the Torus and the Product of Two Circles in Homotopy Type Theory. *ACM Transactions on Computational Logic*, 17(4):29:1–29:19, November 2016. doi:10.1145/2992783.

35 The Agda Development Team. The Agda Programming Language, 2021. URL: http://wiki.portal.chalmers.se/agda/pmwiki.php.

36 The Coq Development Team. The Coq Proof Assistant, 2021. URL: https://www.coq.inria.fr.

37 The RedPRL Development Team. The cooltt proof assistant, 2021. URL: https://github.com/RedPRL/cooltt/.

38 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Self-published, Institute for Advanced Study, 2013. URL: https://homotopytypetheory.org/book/.

39 Floris van Doorn. *On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory*. PhD thesis, Carnegie Mellon University, May 2018. arXiv:1808.10690.

40 Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types. *Proceedings of the ACM on Programming Languages*, 3(ICFP):87:1–87:29, August 2019. doi:10.1145/3341691.

41 Vladimir Voevodsky. The equivalence axiom and univalent models of type theory, February 2010. Notes from a talk at Carnegie Mellon University. URL: http://www.math.ias.edu/vladimir/files/CMU_talk.pdf.

42 Vladimir Voevodsky. An experimental library of formalized mathematics based on the univalent foundations. *Mathematical Structures in Computer Science*, 25(5):1278–1294, 2015. doi:10.1017/S0960129514000577.

On the Minimisation of Transition-Based Rabin Automata and the Chromatic Memory Requirements of Muller Conditions

Antonio Casares
LaBRI, Université de Bordeaux, France

Abstract

In this paper, we relate the problem of determining the chromatic memory requirements of Muller conditions with the minimisation of transition-based Rabin automata. Our first contribution is a proof of the NP-completeness of the minimisation of transition-based Rabin automata. Our second contribution concerns the memory requirements of games over graphs using Muller conditions. A memory structure is a finite state machine that implements a strategy and is updated after reading the edges of the game; the special case of chromatic memories being those structures whose update function only consider the colours of the edges. We prove that the minimal amount of chromatic memory required in games using a given Muller condition is exactly the size of a minimal Rabin automaton recognising this condition. Combining these two results, we deduce that finding the chromatic memory requirements of a Muller condition is NP-complete. This characterisation also allows us to prove that chromatic memories cannot be optimal in general, disproving a conjecture by Kopczyński.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Automata on Infinite Words, Games on Graphs, Arena-Independent Memory, Complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.12

Related Version *Full Version*: https://arxiv.org/abs/2105.12009 [5]

Acknowledgements I would like to thank Alexandre Blanché for pointing me to the chromatic number problem. I also want to thank Bader Abu Radi, Thomas Colcombet, Nathanaël Fijalkow, Orna Kupferman, Karoliina Lehtinen and Nir Piterman for interesting discussions on the minimisation of transition-based automata, a problem introduced to us by Orna Kupferman. Finally, I warmly thank Thomas Colcombet, Nathanaël Fijalkow and Igor Walukiewicz for their help in the preparation of this paper. This work was done while the author was participating in the program *Theoretical Foundations of Computer Systems* at the Simons Institute for the Theory of Computing.

1 Introduction

Games and memory. Automata on infinite words and infinite duration games over graphs are well established areas of study in Computer Science, being central tools used to solve problems such as the synthesis of reactive systems (see for example the Handbook [8]). Games over graphs are used to model the interaction between a system and the environment, and winning strategies can be used to synthesize controllers ensuring that the system satisfies some given specification. The games we will consider are played between two players (Eve and Adam), that alternatively move a pebble through the edges of a graph forming an infinite path. In order to define which paths are winning for the first player, Eve, we suppose that each transition in the game produces a colour in a set Γ, and a winning condition is defined by a subset $\mathbb{W} \subseteq \Gamma^\omega$. A fundamental parameter of the different winning conditions is the amount of memory that the players may require in order to define a winning strategy

in games where they can force a victory. This parameter will influence the complexity of algorithms solving games that use a given winning condition, as well as the resources needed in a practical implementation of such a strategy as a controller for a reactive system.

A memory structure for Eve for a given game is a finite state machine that implements a strategy: for every position of the game, each state of the memory determines what move to perform next. After a transition of the game takes place, the memory state is updated according to an update function. We consider 3 types of memory structures:

- General memories.
- Chromatic memories: if the update function only takes as input the colour produced by the transition of the game.
- Arena-independent memories for a condition $\mathbb{W}$: if the memory structure can be used to implement winning strategies in any game using the condition $\mathbb{W}$.

In this work, we study these three notions of memories for Muller conditions, an important class of winning conditions that can be used to represent any ω-regular language via some deterministic automaton. Muller conditions appear naturally, for example, in the synthesis of reactive systems specified in Linear Temporal Logic [23, 22].

In the seminal paper [12], the authors establish the exact general memory requirements of Muller conditions, giving matching upper and lower bounds for every Muller condition in terms of its Zielonka tree. However, the memory structures giving the upper bounds are not chromatic. In his PhD thesis [18, 19], Kopczyński raised the questions of whether minimal memory structures for games can always be chosen to be chromatic, and whether arena-independent memories can be optimal, that is, if for each condition $\mathbb{W}$ there is a game won by Eve where the optimal amount of memory she can use is the size of a minimal arena-independent memory for $\mathbb{W}$. Another question appearing in [18, 19] concerns the influence in the memory requirements of allowing or not ε-transitions in games (that is, transitions that do not produce any colour). In particular, Kopczyński asks whether all conditions that are half-positionally determined over transition-coloured games without ε-transitions are also half-positionally determined when allowing ε-transitions (it was already shown in [29] that it is not the case in state-coloured games).

In this work, we characterise the minimal amount of chromatic memory required by Eve in games using a Muller condition as the size of a minimal deterministic transition-based Rabin automaton recognising the Muller condition, that can also be used as an arena-independent memory (Theorem 27); further motivating the study of the minimisation of transition-based Rabin automata. We prove that, in general, this quantity is strictly greater than the general memory requirements of the Muller condition, answering negatively the question by Kopczyński (Proposition 30). Moreover, we show that the general memory requirements of a Muller condition are different over ε-free games and over games with ε-transitions (Proposition 24), but that this is no longer the case when considering the chromatic memory requirements (Theorem 27). In particular, in order to obtain the lower bounds of [12] we need to use games with ε-transitions. However, the question stated in [18, 19] of whether allowing ε-transitions could have an impact on the half-positionality of conditions remains open, since it cannot be the case for Muller conditions (Lemma 23).

Minimisation of transition-based automata. Minimisation is a well studied problem for many classes of automata. Automata over finite words can be minimised in polynomial time [15], and for every regular language there is a canonical minimal automaton recognising it. For automata over infinite words, the status of the minimisation problem for different models of ω-automata is less well understood. Traditionally, the acceptance conditions of ω-automata

have been defined over the set of states; however, the use of transition-based automata is becoming common in both practical and theoretical applications (see for instance [13]), and there is evidence that decision problems relating to transition-based models might be easier than the corresponding problems for state-based ones. The minimisation of state-based Büchi automata has been proven to be NP-complete by Schewe (therefore implying the NP-hardness of the minimisation of state-based parity, Rabin and Streett automata), both for deterministic [26] and Good-For-Games (GFG) automata [27]. However, these reductions strongly use the fact that the acceptance condition is defined over the states and not over the transitions. Abu Radi and Kupferman have proven that the minimisation of GFG-transition-based co-Büchi automata can be done in polynomial time and that a canonical minimal GFG-transition-based automaton can be defined for co-Büchi languages [1, 2]. This suggests that transition-based automata might be a more adequate model for ω-automata, raising many questions about the minimisation of different kinds of transition-based automata (Büchi, parity, Rabin, GFG-parity, etc). Moreover, Rabin automata are of great interest, since the determinization of Büchi automata via Safra's construction naturally provides deterministic transition-based Rabin automata [24, 25], and, as proven in Theorem 27, these automata provide minimal arena-independent memories for Muller games.

In Section 2.2, we prove that the minimisation of transition-based Rabin automata is NP-complete (Theorem 14). The proof consists in a reduction from the chromatic number problem of graphs. This reduction uses a particularly simple family of ω-regular languages: languages $L \subseteq \Sigma^\omega$ that correspond to Muller conditions, that is, whether a word $w \in \Sigma^\omega$ belongs to L or not only depends in the set of letters appearing infinitely often in w (we call these *Muller languages*). A natural question is whether we can extend this reduction to prove the NP-hardness of the minimisation of other kinds of transition-based automata, like parity or generalised Büchi ones. However, we prove in Section 2.3 that the minimisation of parity and generalised Büchi automata recognising Muller languages can be done in polynomial time. This is based in the fact that the minimal parity automaton recognising a Muller language is given by the Zielonka tree of the associated condition [6, 21].

These results allow us to conclude that determining the chromatic memory requirements of a Muller condition is NP-complete even if the condition is represented by its Zielonka tree (Theorem 29). This is a surprising result, since the Zielonka tree of a Muller condition allows to compute in linear time the non-chromatic memory requirements of it [12].

Related work. As already mentioned, the works [12, 18, 29] extensively study the memory requirements of Muller conditions. In the paper [10], the authors characterise parity conditions as the only prefix-independent conditions that admit positional strategies over transition-coloured infinite graphs. This characterisation does not apply to state-coloured games, which supports the idea that transition-based systems might present more canonical properties. Conditions that admit arena-independent memories are characterised in [3], extending the work of [14] characterising conditions that accept positional strategies over finite games. The memory requirements of generalised safety conditions have been established in [9]. The use of Rabin automata as memories for games with ω-regular conditions have been fruitfully used in [11] in order to obtain theoretical lower bounds on the size of deterministic Rabin automata obtained by the determinisation of Büchi automata.

Concerning the minimisation of automata over infinite words, beside the aforementioned results of [26, 27, 1], it is also known that weak automata can be minimised in $\mathcal{O}(n \log n)$ [20]. The algorithm minimising a parity automaton recognising a Muller language used in the

proof of Proposition 16 can be seen as a generalisation of the algorithm appearing in [4] computing the Rabin index of a parity automaton. Both of them have their roots in the work of Wagner [28].

Organisation of this paper. In Section 2 we discuss the minimisation of transition-based Rabin and parity automata. We give the necessary definitions in Section 2.1, in Section 2.2 we show the NP-completeness of the minimisation of Rabin automata and in Section 2.3 we prove that we can minimise transition-based parity and generalised Büchi automata recognising Muller languages in polynomial time.

In Section 3 we introduce the definitions of games and memory structures, and we discuss the impact on the memory requirements of allowing or not ε-transitions in the games.

In Section 4, the main contributions concerning the chromatic memory requirements of Muller conditions are presented.

2 Minimising transition-based automata

In this section, we present our main contributions concerning the minimisation of automata. We start in Section 2.1 by giving some basic definitions and results related to automata used throughout the paper. In Section 2.2 we show a reduction from the problem of determining the chromatic number of a graph to the minimisation of Rabin automata, proving the NP-completeness of the latter. Moreover, the languages used in this proof are Muller languages. In Section 2.3 we prove that, on the contrary, we can minimise parity and generalised Büchi automata recognising Muller conditions in polynomial time.

2.1 Automata over infinite words

General notations

The greek letter ω stands for the set $\{0, 1, 2, \dots\}$. We write $[1, k]$ to denote the set $\{1, 2, \dots, k\}$. Given a set A, we write $\mathcal{P}(A)$ to denote its power set and $|A|$ to denote its cardinality. A word over an alphabet Σ is a sequence of letters from Σ. We let Σ^* and Σ^ω be the set of finite and infinite words over Σ, respectively. For an infinite word $w \in \Sigma^\omega$, we write $Inf(w)$ to denote the set of letters that appear infinitely often in w. We will extend functions $\gamma : A \to \Gamma$ to A^*, A^ω and $\mathcal{P}(A)$ in the natural way, without explicitly stating it.

A (directed) *graph* $G = (V, E)$ is given by a set of vertices V and a set of edges $E \subseteq V \times V$. A graph $G = (V, E)$ is *undirected* if every pair of vertices (v, u) verifies $(v, u) \in E \Leftrightarrow (u, v) \in E$. A graph $G = (V, E)$ is *simple* if $(v, v) \notin E$ for any $v \in V$. A *coloured graph* $G = (V, E)$ is given by a set of vertices V and a set of edges $E \subseteq V \times C_1 \times \cdots \times C_k \times V$, where $C_1, \dots, C_k$ are sets of colours.

Automata

An *automaton* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Gamma, Acc)$, where Q is a finite set of states, Σ is a finite input alphabet, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q \times \Gamma$ is a transition function, Γ is an output alphabet and Acc is an accepting condition defining a subset $\mathbb{W} \subseteq \Gamma^\omega$ (the conditions will be defined more precisely in the next paragraph). In this paper, all automata will be deterministic, complete (δ is a function) and transition-based (the output letter that is produced depends on the transition, and not only on the arrival state). The *size* of an automaton is the cardinality of its set of states, $|Q|$.

Given an input word $w = w_0w_1w_2\cdots \in \Sigma^\omega$, the *run over* w in $\mathcal{A}$ is the only sequence of pairs $(q_0, c_0), (q_1, c_1), \cdots \in Q \times \Gamma$ verifying that q_0 is the initial state and $\delta(q_i, w_i) = (q_{i+1}, c_i)$. The *output* produced by w is the word $c_0c_1c_2\cdots \in \Gamma^\omega$. A word $w \in \Sigma^\omega$ is *accepted by* the automaton $\mathcal{A}$ if its output belongs to the set $\mathbb{W} \subseteq \Gamma^\omega$ defined by the accepting condition. The *language accepted by* an automaton $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. Given two automata $\mathcal{A}$ and $\mathcal{B}$ over the same input alphabet Σ, we say that they are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

Given an automaton $\mathcal{A}$, the *graph associated to* $\mathcal{A}$, denoted $G(\mathcal{A})$, is the coloured graph $G(\mathcal{A}) = (Q, E_\mathcal{A})$, whose set of vertices is Q, and the set of edges $E_\mathcal{A} \subseteq Q \times \Sigma \times \Gamma \times Q$ is given by $(q, a, c, q') \in E_\mathcal{A} \Leftrightarrow \delta(q, a) = (q', c)$. We denote $\iota\colon E_\mathcal{A} \to \Sigma$ the projection over the second component and $\gamma\colon E_\mathcal{A} \to \Gamma$ the projection over the third one.

A *cycle* of an automaton $\mathcal{A}$ is a subset of edges $\ell \subseteq E_\mathcal{A}$ such that there is a state $q \in Q$ and a path in $G(\mathcal{A})$ starting and ending in q passing through exactly the edges in ℓ. We write $\gamma(\ell) = \bigcup_{e\in\ell} \gamma(e)$ to denote the set of colours appearing in the cycle ℓ. A state $q \in Q$ is *contained in* a cycle $\ell \subseteq E_\mathcal{A}$ if there is some edge in ℓ whose first component is q. We write $\mathit{States}(\ell)$ to denote the set of states contained in ℓ.

Acceptance conditions

Let Γ be a set of colours. We define next some of the acceptance conditions used to define subsets $\mathbb{W} \subseteq \Gamma^\omega$. All the subsequent conditions verify that the acceptance of a word $w \in \Gamma^\omega$ only depends on the set $\mathit{Inf}(w)$.

Muller. A *Muller condition* is given by a family of subsets $\mathcal{F} = \{S_1, \dots, S_k\}$, $S_i \subseteq \Gamma$. A word $w \in \Gamma^\omega$ is accepting if $\mathit{Inf}(w) \in \mathcal{F}$.

Rabin. A *Rabin condition* is represented by a family of *Rabin pairs*, $R = \{(E_1, F_1), \dots, (E_r, F_r)\}$, where $E_i, F_i \subseteq \Gamma$. A word $w \in \Gamma^\omega$ is accepting if $\mathit{Inf}(w) \cap E_i \neq \emptyset$ and $\mathit{Inf}(w) \cap F_i = \emptyset$ for some index $i \in \{1, \dots, r\}$.

Streett. A *Streett condition* is represented by a family of pairs $S = \{(E_1, F_1), \dots, (E_r, F_r)\}$, $E_i, F_i \subseteq \Gamma$. A word $w \in \Gamma^\omega$ is accepting if $\mathit{Inf}(w) \cap E_i \neq \emptyset \rightarrow \mathit{Inf}(w) \cap F_i \neq \emptyset$ for every $i \in \{1, \dots, r\}$.

Parity. To define a *parity condition* we suppose that Γ is a finite subset of $\mathbb{N}$. A word $w \in \Gamma^\omega$ is accepting if $\max \mathit{Inf}(w)$ is even. The elements of Γ are called *priorities* in this case.

Generalised Büchi. A *generalised Büchi condition* is represented by a family of subsets $\{B_1, \dots, B_r\}$, $B_i \subseteq \Gamma$. A word $w \in \Gamma^\omega$ is accepted if $\mathit{Inf}(w) \cap B_i \neq \emptyset$ for all $i \in \{1, \dots, r\}$.

Generalised co-Büchi. A *generalised co-Büchi condition* is represented by a family of subsets $\{B_1, \dots, B_r\}$, $B_i \subseteq \Gamma$. A word $w \in \Gamma^\omega$ is accepted if $\mathit{Inf}(w) \cap B_i = \emptyset$ for some $i \in \{1, \dots, r\}$.

An automaton $\mathcal{A}$ using a condition of type X will be called an X-automaton.

We remark that all the previous conditions define a family of subsets $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ and can therefore be represented as Muller conditions (in particular, all automata referred to in this paper can be regarded as Muller automata). Also, parity conditions can be represented as Rabin or Streett ones. We say that a language $L \subseteq \Gamma^\omega$ is a *Muller language* if $w_1 \in L$ and $w_2 \notin L$ implies that $\mathit{Inf}(w_1) \neq \mathit{Inf}(w_2)$. We associate to each Muller condition $\mathcal{F}$ the language $L_\mathcal{F} = \{w \in \Gamma^\omega \ : \ \mathit{Inf}(w) \in \mathcal{F}\}$.

The *parity index* (also called *Rabin index*) of an ω-regular language $L \subseteq \Sigma^\omega$ is the minimal $p \in \mathbb{N}$ such that there exists a parity automaton recognising L using p priorities in its condition.

Given an ω-regular language $L \subseteq \Sigma^\omega$, we write $\mathfrak{rabin}(L)$ to denote the size of a minimal Rabin automaton recognising L.

▶ Remark 1. Let $\mathcal{A}$ be a Rabin-automaton recognising a language $L \subseteq \Sigma^\omega$. If we consider the Streett automaton obtained by regarding the Rabin pairs of $\mathcal{A}$ as defining a Streett condition, we obtain an automaton $\mathcal{A}'$ recognising the language $\Sigma^\omega \setminus L$ (and vice-versa). Therefore, the size of a minimal Rabin automaton recognising L coincides with that of a minimal Streett automaton recognising $\Sigma^\omega \setminus L$, and the minimisation problem for both classes of automata is equivalent. Similarly for generalised Büchi and generalised co-Büchi automata.

Let $\mathcal{A}$ be an automaton using some of the acceptance conditions above defining a family $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$. We say that a cycle ℓ of $\mathcal{A}$ is *accepting* if $\gamma(\ell) \in \mathcal{F}$ and that it is *rejecting* otherwise.

We are going to be interested in simplifying the acceptance conditions of automata, while preserving their structure. We say that we can *define a condition of type X on top of a Muller automaton $\mathcal{A}$* if we can recolour the transitions of $\mathcal{A}$ with colours in a set Γ' and define a condition of type X over Γ' such that the resulting automaton is equivalent to $\mathcal{A}$. Definition 2 formalises this notion.

▶ **Definition 2.** *Let X be some of the types of conditions defined previously and let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Gamma, \mathcal{F})$ be a Muller automaton. We say that we can define a* condition of type X on top of $\mathcal{A}$ *if there is an X-condition over a set of colours Γ' and an automaton $\mathcal{A}' = (Q, \Sigma, q_0, \delta', \Gamma', X)$ verifying:*

- *$\mathcal{A}$ and $\mathcal{A}'$ have the same set of states and the same initial state.*
- *$\delta(q, a) = (p, c) \Rightarrow \delta'(q, a) = (p, c')$, for some $c' \in \Gamma'$, for every $q \in Q$ and $a \in \Sigma$ (that is, $\mathcal{A}$ and $\mathcal{A}'$ have the same transitions, except for the colours produced).*
- *$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

The next proposition, proven in [6], characterises automata that admit Rabin conditions on top of them. It will be a key property used throughout the paper.

▶ **Proposition 3** ([6]). *Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Gamma, \mathcal{F})$ be a Muller automaton. The following properties are equivalent:*

1. *We can define a Rabin condition on top of $\mathcal{A}$.*
2. *Any pair of cycles ℓ_1 and ℓ_2 in $\mathcal{A}$ verifying $States(\ell_1) \cap States(\ell_2) \neq \emptyset$ satisfies that if both ℓ_1 and ℓ_2 are rejecting, then $\ell_1 \cup \ell_2$ is also a rejecting cycle.*

The Zielonka tree of a Muller condition

In order to study the memory requirements of Muller conditions, Zielonka introduced in [29] the notion of split trees (later called Zielonka trees) of Muller conditions. The Zielonka tree of a Muller condition naturally provides a minimal parity automaton recognising the associated language [6, 21]. We will use this property to show that parity automata recognising Muller languages can be minimised in polynomial time in Proposition 16. We will come back to Zielonka trees in Section 4 to discuss the memory requirements of Muller conditions.

▶ **Definition 4.** *Let Γ be a set of labels. We define a* Γ-labelled-tree *by induction:*

- *$T = \langle A, \langle\emptyset\rangle\rangle$ is a Γ-labelled-tree for any $A \subseteq \Gamma$. In this case, we say that T is a* leaf *and A is its label.*
- *If $T_1, \ldots, T_n$ are Γ-labelled-trees, then $T = \langle A, \langle T_1, \ldots, T_n\rangle\rangle$ is a Γ-labelled-tree for any $A \subseteq \Gamma$. In that case, we say that A is the label of T and $T_1, \ldots, T_n$ are their children.*

▶ **Definition 5** ([29]). *Let $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ be a Muller condition. The* Zielonka tree *of $\mathcal{F}$, denoted $\mathcal{Z}_\mathcal{F}$, is the Γ-labelled-tree defined recursively as follows: let $A_1, \dots A_k$ be the maximal subsets of Γ (with respect to set inclusion) such that $A_i \in \mathcal{F} \Leftrightarrow \Gamma \notin \mathcal{F}$ (that is, producing an "alternation of the acceptance condition").*

- *If no such subset $A_i \subseteq \Gamma$ exists, then $\mathcal{Z}_\mathcal{F} = \langle \Gamma, \langle \emptyset \rangle \rangle$.*
- *Otherwise, $\mathcal{Z}_\mathcal{F} = \langle \Gamma, \langle \mathcal{Z}_{\mathcal{F}_1}, \dots, \mathcal{Z}_{\mathcal{F}_k} \rangle \rangle$, where $\mathcal{Z}_{\mathcal{F}_i}$ is the Zielonka tree for the condition $\mathcal{F}_i = \mathcal{F} \cap \mathcal{P}(A_i)$ over the set of colours A_i.*

An example of a Zielonka tree can be found in Figure 1 (page 15).

▶ **Proposition 6** ([6, 21]). *Let $\mathcal{F}$ be a Muller condition and $\mathcal{Z}_\mathcal{F}$ its Zielonka tree. We can build in linear time in the representation of $\mathcal{Z}_\mathcal{F}$ a parity automaton recognising $L_\mathcal{F}$ that has as set of states the leaves of $\mathcal{Z}_\mathcal{F}$. This automaton is minimal, that is, any other parity automaton recognising $L_\mathcal{F}$ has at least as many states as the number of leaves of $\mathcal{Z}_\mathcal{F}$.*

2.2 Minimising transition-based Rabin and Streett automata is NP-complete

This section is devoted to proving the NP-completeness of the minimisation of transition-based Rabin automata, stated in Theorem 14.

For the containment in NP, we use the fact that we can test language equivalence of Rabin automata in polynomial time [7].

▶ **Proposition 7** ([7]). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two Rabin automata over Σ. We can decide in polynomial time in the representation of the automata if $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$. (We recall that all considered automata are deterministic).*

▶ **Corollary 8.** *Given a Rabin automaton $\mathcal{A}$ and a positive integer k, we can decide in non-deterministic polynomial time whether there is an equivalent Rabin automaton of size k.*

Proof. A non-deterministic Turing machine just has to guess an equivalent automaton $\mathcal{A}_k$ of size k, and by Proposition 7 it can check in polynomial time whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_k)$. ◀

In order to prove the NP-hardness, we will describe a reduction from the chromatic number problem (one of 21 Karp's NP-complete problems) to the minimisation of transition-based Rabin automata. Moreover, this reduction will only use languages that are Muller languages of parity index 3.

▶ **Definition 9.** *Let $G = (V, E)$ be a simple undirected graph. A* colouring *of size k of G is a function $c : V \to [1, k]$ such that for any pair of vertices $v, v' \in V$, if $(v, v') \in E$ then $c(v) \neq c(v')$. The* chromatic number *of a simple undirected G, written $\chi(G)$, is the smallest number k such that there exists a colouring of size k of G.*

▶ **Lemma 10** ([16]). *Deciding whether a simple undirected graph has a colouring of size k is* NP*-complete.*

Let $G = (V, E)$ be a simple undirected graph, n be its number of vertices and m its number of edges. We consider the language L_G over the alphabet V given by:

$$L_G = \bigcup_{(v,u) \in E} V^*(v^+u^+)^\omega.$$

That is, a sequence $w \in V^\omega$ is in L_G if eventually it alternates between exactly two vertices connected by an edge in G.

▶ Remark 11. For any simple undirected graph G, L_G is a Muller language over V, that is, whether a word $w \in V^\omega$ belongs to L_G or not only depends on $Inf(w)$. Moreover, the parity index of this language is at most 3.

However, we cannot extend this reduction to show NP-hardness of the minimisation of transition-based parity automata, as we will show in Section 2.3 that we can minimise parity automata recognising Muller languages in polynomial time.

In order to show that this is indeed a polynomial-time reduction, we have to be able to build a Rabin automaton recognising L_G in polynomial time in the representation of G. This is indeed the case, since we can consider a Rabin automaton that has as set of states the vertices of G, and such that, from any state, when reading a letter $v \in V$ we go to the state v. We use the information about the edges of G to define a Rabin condition over this automaton so that it recognises L_G. The details of this construction can be found in the full version [5].

▶ **Lemma 12.** *We can build a Rabin automaton of size n recognising L_G in $\mathcal{O}(mn^2)$.*

▶ **Lemma 13.** *Let $G = (V, E)$ be a simple undirected graph. Then, the size of a minimal Rabin automaton recognising L_G coincides with the chromatic number of G, $\chi(G)$.*

Proof.

$\mathtt{rabin}(L_G) \leq \chi(G)$: Let $c : V \to [1, k]$ be a colouring of size k of G. We will define a Muller automaton of size k recognising L_G and then use Proposition 3 to show that we can put a Rabin condition on top of it. Let $\mathcal{A}_c = (Q, V, q_0, \delta, V, \mathcal{F})$ be the Muller automaton defined by:

- $Q = \{1, 2, \dots, k\}$.
- $q_0 = 1$.
- $\delta(q, x) = (c(x), x)$ for $q \in Q$ and $x \in V$.
- A set $C \subseteq V$ belongs to $\mathcal{F}$ if and only if $C = \{v, u\}$ for two vertices $v, u \in V$ such that $(v, u) \in E$.

The language recognised by $\mathcal{A}_c$ is clearly L_G, since the output produced by a word $w \in V^\omega$ is w itself, and the acceptance condition $\mathcal{F}$ is exactly the Muller condition defining the language L_G.

Let $G(\mathcal{A}_c) = (Q, E_{\mathcal{A}_c})$ be the graph associated to $\mathcal{A}_c$. We will prove that the union of any pair of rejecting cycles of $\mathcal{A}_c$ that have some state in common must be a rejecting cycle. By Proposition 3 this implies that we can define a Rabin condition on top of $\mathcal{A}_c$. Let $\ell_1, \ell_2 \subseteq E_{\mathcal{A}_c}$ be two cycles such that $\gamma(\ell_i) \notin \mathcal{F}$ for $i \in \{1, 2\}$ and such that $States(\ell_1) \cap States(\ell_2) \neq \emptyset$. We distinguish 3 cases:

- $|\gamma(\ell_i)| \geq 3$ for some $i \in \{1, 2\}$. In this case, their union also has more than 3 colours, so it must be rejecting.
- $\gamma(\ell_i) = \{u, v\}$, $(u, v) \notin E$ for some $i \in \{1, 2\}$. In that case, $\gamma(\ell_1 \cup \ell_2)$ also contains two vertices that are not connected by an edge, so it must be rejecting.
- $\gamma(\ell_1) = \{v_1\}$ and $\gamma(\ell_2) = \{v_2\}$. In this case, since from every state q of $\mathcal{A}_c$ and every $v \in V$ we have that $\delta(q, v) = (c(v), v)$, each of the cycles contains only one state: $States(\ell_1) = \{c(v_1)\}$ and $States(\ell_2) = \{c(v_2)\}$. As ℓ_1 and ℓ_2 share some state, we deduce that $c(v_1) = c(v_2)$. If $v_1 = v_2$, $\ell_1 \cup \ell_2$ is rejecting because $|\gamma(\ell_1 \cup \ell_2)| = 1$. If $v_1 \neq v_2$, it is also rejecting because $c(v_1) = c(v_2)$, and therefore $(v_1, v_2) \notin E$.

Since $\gamma(\ell_i)$ is rejecting, it does not consist on two vertices connected by some edge and we are always in some of the cases above. We conclude that we can put a Rabin condition on top of $\mathcal{A}_c$, obtaining a Rabin automaton recognising L_G of size k.

$\chi(G) \leq \mathbf{rabin}(L_G)$**:** Let $\mathcal{A} = (Q, V, q_0, \delta, \Gamma, R)$ be a Rabin automaton of size k recognising L_G and let $G(\mathcal{A}) = (Q, E_{\mathcal{A}})$ be its graph. We will define a colouring of size k of G, $c : V \to Q$. For each $v \in V$ we define a subset $Q_v \subseteq Q$ as:

$$Q_v = \{q \in Q \ : \text{ there is a cycle } \ell \text{ containing } q \text{ and } \gamma(\ell) = \{v\}\}.$$

For every $v \in V$, the set Q_v is non-empty, as it must exist a (non-accepting) run over v^ω in $\mathcal{A}$. For each $v \in V$ we pick some $q_v \in Q_v$, and we define the colouring $c : V \to Q$ given by $c(v) = q_v$.
In order to prove that it is indeed a colouring, we we will show that any two vertices $v, u \in V$ such that $(v, u) \in E$ verify that $Q_v \cap Q_u = \emptyset$, and therefore they also verify $c(v) \neq c(u)$. Suppose by contradiction that there is some $q \in Q_v \cap Q_u$. We write ℓ_x for a cycle containing q labelled with x, for $x \in \{v, u\}$ (they exist by the definition of Q_x). By the definition of L_G, both cycles ℓ_v and ℓ_u have to be rejecting as $x^\omega \notin L_G$ for any $x \in V$. However, since $(u, v) \in E$, their union is accepting, contradicting Proposition 3. ◀

We deduce the NP-completeness of the minimisation of Rabin automata.

▶ **Theorem 14.** *Given a Rabin automaton $\mathcal{A}$ and a positive integer k, deciding whether there is an equivalent Rabin automaton of size k is* NP*-complete.*

▶ **Corollary 15.** *Given a Streett automaton $\mathcal{A}$ and a positive integer k, deciding whether there is an equivalent Streett automaton of size k is* NP*-complete.*

2.3 Parity and generalised Büchi automata recognising Muller languages can be minimised in polynomial time

In Section 2.2 we have proven the NP-hardness of the minimisation of Rabin automata showing a reduction that uses Muller languages, that is, whether an infinite word w belongs to the language only depends on $Inf(w)$. We may wonder whether Muller languages could be used to prove NP-hardness of the minimisation of parity or generalised Büchi automata. We shall see now that this is not the case.

▶ **Proposition 16.** *Let $\mathcal{F} \subseteq \mathcal{P}(\Sigma)$ be a Muller condition. Given a parity automaton recognising $L_{\mathcal{F}}$, we can build in polynomial time a minimal parity automaton recognising $L_{\mathcal{F}}$.*

As stated in Proposition 6, a minimal parity automaton recognising a Muller language can be obtained in linear time from the Zielonka tree of the condition, so it suffices to give a polynomial-time algorithm building the Zielonka tree of the Muller condition $\mathcal{F}$ from a parity automaton $\mathcal{A}$ recognising $L_{\mathcal{F}}$. The details of this algorithm are included in the full version [5]. We give next the main ideas of it.

We start by labelling the root of $\mathcal{Z}_{\mathcal{F}}$ with Σ. Next, we try to find the maximal subsets of Σ that are alternating (that is, Σ is in $\mathcal{F}$ if and only if they are not). To do so, we remove the transitions of $\mathcal{A}$ corresponding to the maximal priority (that we suppose even), and we compute a decomposition in strongly connected components of the obtained graph. We keep the ergodic components (that is, those without edges leaving them), and we recursively repeat this process in those components with a maximal even priority, until obtaining a set of strongly connected components with maximal odd priorities. For each of these components, we take the set of input letters that appear on their transitions. The maximal sets of letters among them will constitute the children of the root of $\mathcal{Z}_{\mathcal{F}}$. We continue recursively until we do not find any new "alternating components".

▶ **Proposition 17.** *Let $\mathcal{F} \subseteq \mathcal{P}(\Sigma)$ be a Muller condition. If $L_{\mathcal{F}}$ can be recognised by a generalised Büchi (resp. generalised co-Büchi) automaton, then, it can be recognised by one such automaton with just one state. Moreover, this minimal automaton can be built in polynomial time from any generalised Büchi (resp. generalised co-Büchi) automaton recognising $L_{\mathcal{F}}$.*

The proof of Proposition 17 appears in the full version [5].

3 Memory in games

In this section, we introduce the definitions of games, memories and chromatic memories for games, as well as ε-free games. We show in Section 3.4 that the memory requirements for games where we allow ε-transitions might differ from those for ε-free games.

3.1 Games

A *game* is a tuple $\mathcal{G} = (V = V_E \uplus V_A, E, v_0, \gamma : E \to \Gamma \cup \{\varepsilon\}, Acc)$ where (V, E) is a directed graph together with a partition of the vertices $V = V_E \uplus V_A$, v_0 is an initial vertex, γ is a colouring of the edges and Acc is a winning condition defining a subset $\mathbb{W} \subseteq \Gamma^\omega$. The letter ε is a neutral letter, and we impose that there is no cycle in $\mathcal{G}$ labelled exclusively with ε. We say that vertices in V_E belong to *Eve* (also called the *existential player*) and those in V_A to *Adam* (*universal player*). We suppose that each vertex in V has at least one outgoing edge. A game that uses a winning condition of type X (as defined in Section 2.1) is called an X-game.

A *play* in $\mathcal{G}$ is an infinite path $\varrho \in E^\omega$ produced by moving a token along edges starting in v_0: the player controlling the current vertex chooses what transition to take. Such a play produces a word $\gamma(\varrho) \in (\Gamma \cup \{\varepsilon\})^\omega$. Since no cycle in $\mathcal{G}$ consists exclusively of ε-colours, after removing the occurrences of ε from $\gamma(\varrho)$ we obtain a word in Γ^ω, that we will call the *output* of the play and we will also denote $\gamma(\varrho)$ whenever no confusion arises. The play is *winning* for Eve if the output belongs to the set $\mathbb{W}$ defined by the acceptance condition, and winning for Adam otherwise. A *strategy* for Eve in $\mathcal{G}$ is a function prescribing how Eve should play. Formally, it is a function $\sigma : E^* \to E$ that associates to each partial play ending in a vertex $v \in V_E$ some outgoing edge from v. A play $\varrho \in E^\omega$ *adheres* to the strategy σ if for each partial play $\varrho' \in E^*$ that is a prefix of ϱ and ends in some state of Eve, the next edge played coincides with $\sigma(\varrho')$. We say that Eve *wins* the game $\mathcal{G}$ if there is some strategy σ for her such that any play that adheres to σ is a winning play for her (in this case we say that σ is a *winning strategy)*.

We will also study games without ε-transitions. We say that a game $\mathcal{G}$ is *ε-free* if $\gamma(e) \neq \varepsilon$ for all edges $e \in E$.

3.2 Memory structures

We give the definitions of the following notions from the point of view of the existential player, Eve. Symmetric definitions can be given for the universal player (Adam), and all results of Section 4 can be dualised to apply to the universal player.

A *memory structure for the game* $\mathcal{G}$ is a tuple $\mathcal{M}_{\mathcal{G}} = (M, m_0, \mu)$ where M is a set of states, $m_0 \in M$ is an initial state and $\mu : M \times E \to M$ is an update function (where E denotes the set of edges of the game). Its *size* is $|M|$. We extend the function μ to $M \times E^*$ in the natural way. We can use such a memory structure to define a strategy for Eve using a

function $\texttt{next-move} : V_E \times M \to E$, verifying that $\texttt{next-move}(v, m)$ is an outgoing edge from v. After each move of a play on $\mathcal{G}$, the state of the memory $\mathcal{M}_\mathcal{G}$ is updated using μ; and when a partial play arrives to a vertex v controlled by Eve she plays the edge indicated by the function $\texttt{next-move}(v, m)$, where m is the current state of the memory. We say that the memory structure $\mathcal{M}_\mathcal{G}$ *sets a winning strategy* in $\mathcal{G}$ if there exists such a function $\texttt{next-move}$ defining a winning strategy for Eve.

We say that $\mathcal{M}_\mathcal{G}$ is a *chromatic memory* if there is some function $\mu' : M \times \Gamma \to M$ such that $\mu(m, e) = \mu'(m, \gamma(e))$ for every edge $e \in E$ such that $\gamma(e) \neq \varepsilon$, and $\mu(m, e) = m$ if $\gamma(e) = \varepsilon$. That is, the update function of $\mathcal{M}_\mathcal{G}$ only depends on the colours of the edges of the game.

Given a winning condition $\mathbb{W} \subseteq \Gamma^\omega$, we say that $\mathcal{M} = (M, m_0, \mu\colon M \times \Gamma \to M)$ is an *arena-independent memory* for $\mathbb{W}$ if for any $\mathbb{W}$-game $\mathcal{G}$ won by Eve, there exists some function $\texttt{next-move}_\mathcal{G} : V_E \times M \to E$ setting a winning strategy in $\mathcal{G}$. We remark that such a memory is always chromatic.

Given a Muller condition $\mathcal{F}$, we write $\mathfrak{mem}_{gen}(\mathcal{F})$ (resp. $\mathfrak{mem}_{chrom}(\mathcal{F})$) for the least number n such that for any $\mathcal{F}$-game that is won by Eve, she can win it using a memory (resp. a chromatic memory) of size n. We call $\mathfrak{mem}_{gen}(\mathcal{F})$ (resp. $\mathfrak{mem}_{chrom}(\mathcal{F})$) the *general memory requirements (resp. chromatic memory requirements) of* $\mathcal{F}$. We write $\mathfrak{mem}_{ind}(\mathcal{F})$ for the least number n such that there exists an arena-independent memory for $\mathcal{F}$ of size n.

We define respectively all these notions for ε-free $\mathcal{F}$-games. We write $\mathfrak{mem}^{\varepsilon\text{-free}}_{gen}(\mathcal{F})$, $\mathfrak{mem}^{\varepsilon\text{-free}}_{chrom}(\mathcal{F})$ and $\mathfrak{mem}^{\varepsilon\text{-free}}_{ind}(\mathcal{F})$ to denote, respectively, the minimal general memory requirements, minimal chromatic memory requirements and minimal size of an arena-independent memory for ε-free $\mathcal{F}$-games.

▶ Remark 18. We remark that these quantities verify that $\mathfrak{mem}_{gen}(\mathcal{F}) \leq \mathfrak{mem}_{chrom}(\mathcal{F}) \leq \mathfrak{mem}_{ind}(\mathcal{F})$ and that $\mathfrak{mem}^{\varepsilon\text{-free}}_{X}(\mathcal{F}) \leq \mathfrak{mem}_X(\mathcal{F})$ for $X \in \{gen, chrom, ind\}$.

A family of games is *half-positionally determined* if for every game in the family that is won by Eve, she can win it using a strategy given by a memory structure of size 1.

▶ **Lemma 19** ([17, 29]). *Rabin-games are half-positionally determined.*

If $\mathcal{A}$ is a Rabin automaton recognising the Muller language associated to the condition $\mathcal{F}$, given an $\mathcal{F}$-game $\mathcal{G}$ we can perform a standard product construction $\mathcal{G} \ltimes \mathcal{A}$ to obtain an equivalent game using a Rabin condition that is therefore half-positionally determined. This allows us to use the automaton $\mathcal{A}$ as an arena-independent memory for $\mathcal{F}$-games.

▶ **Lemma 20** (Folklore). *Let $\mathcal{F}$ be a Muller condition. We can use a Rabin automaton $\mathcal{A}$ recognising $L_\mathcal{F}$ as an arena-independent memory for $\mathcal{F}$-games.*

3.3 The general memory requirements of Muller conditions

The Zielonka tree (see Definition 5) was introduced by Zielonka in [29], and in [12] it was used to characterise the general memory requirements of Muller games as we show next.

▶ **Definition 21.** *Let $\mathcal{F}$ be a Muller condition and $\mathcal{Z}_\mathcal{F} = \langle \Gamma, \langle \mathcal{Z}_{\mathcal{F}_1}, \dots \mathcal{Z}_{\mathcal{F}_k} \rangle \rangle$ its Zielonka tree. We define the number $\mathfrak{m}_{\mathcal{Z}_\mathcal{F}}$ recursively as follows:*

$$\mathfrak{m}_{\mathcal{Z}_\mathcal{F}} = \begin{cases} 1 & \text{if } \mathcal{Z}_\mathcal{F} \text{ is a leaf.} \\ \max\{\mathfrak{m}_{\mathcal{Z}_{\mathcal{F}_1}}, \dots, \mathfrak{m}_{\mathcal{Z}_{\mathcal{F}_k}}\} & \text{if } \Gamma \notin \mathcal{F} \text{ and } \mathcal{Z}_\mathcal{F} \text{ is not a leaf.} \\ \sum_{i=1}^{k} \mathfrak{m}_{\mathcal{Z}_{\mathcal{F}_i}} & \text{if } \Gamma \in \mathcal{F} \text{ and } \mathcal{Z}_\mathcal{F} \text{ is not a leaf.} \end{cases}$$

▶ **Proposition 22** ([12]). *For every Muller condition $\mathcal{F}$, $\mathfrak{mem}_{gen}(\mathcal{F}) = \mathfrak{m}_{\mathcal{Z}_\mathcal{F}}$. That is,*

1. *If Eve wins an $\mathcal{F}$-game, she can win it using a strategy given by a (general) memory structure of size at most $\mathfrak{m}_{\mathcal{Z}_\mathcal{F}}$.*
2. *There exists an $\mathcal{F}$-game (with ε-transitions) won by Eve such that she cannot win it using a strategy given by a memory structure of size strictly smaller than $\mathfrak{m}_{\mathcal{Z}_\mathcal{F}}$.*

3.4 Memory requirements of ε-free games

In [29] and [18] it was noticed that there can be major differences regarding the memory requirements of winning conditions depending on the way the games are coloured. We can differentiate 4 classes of games, corresponding to the combinations of two parameters: state-coloured or transition-coloured, and allowing or not ε-transitions. In [29], Zielonka showed that there are Muller conditions that are half-positional over state-coloured ε-free games, but they are not half-positional over general state-coloured games (that is, games where some states may be left uncoloured), and he exactly characterises half-positional Muller conditions in both cases.

However, when considering transition-coloured games, this is no longer the case: in both general games and ε-free games, half-positional Muller conditions correspond exactly to Rabin conditions (Lemma 23). Nevertheless, the matching upper bounds for the memory requirements of Muller conditions appearing in [12] are given by transition-labelled games using ε-transitions. An interesting question is whether we can produce upper-bound examples using ε-free games. In this section we answer this question negatively. We show in Proposition 24 that there are Muller conditions $\mathcal{F}$ for which the memory required by Eve in ε-free $\mathcal{F}$-games is strictly smaller than the memory she needs in general $\mathcal{F}$-games, and the difference can be arbitrarily large. In Section 4.1 we will see that this is not the case for chromatic memories: $\mathfrak{mem}_{chrom}(\mathcal{F}) = \mathfrak{mem}^{\varepsilon\text{-free}}_{chrom}(\mathcal{F})$ for any Muller condition $\mathcal{F}$.

The details of the proofs of Lemma 23 and Proposition 24 can be found in the full version of this paper [5].

▶ **Lemma 23.** *For any Muller condition $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$, $\mathcal{F}$ is half-positional determined over transition-coloured ε-free games if and only if $\mathcal{F}$ is half-positional determined over general transition-coloured games. That is, $\mathfrak{mem}_{gen}(\mathcal{F}) = 1$ if and only if $\mathfrak{mem}^{\varepsilon\text{-free}}_{gen}(\mathcal{F}) = 1$.*

▶ **Proposition 24.** *For any integer $n \geq 2$, there is a set of colours Γ_n and a Muller condition $\mathcal{F}_n \subseteq \mathcal{P}(\Gamma_n)$ such that $\mathfrak{mem}^{\varepsilon\text{-free}}_{gen}(\mathcal{F}) = 2$ and $\mathfrak{mem}_{gen}(\mathcal{F}) = n$.*

Proof. Let us consider the set of colours $\Gamma_n = \{1, \dots, n\}$ and the Muller condition $\mathcal{F}_n = \{A \subseteq \Gamma_n \,:\, |A| > 1\}$. The characterisation of [12] (Proposition 22) gives that $\mathfrak{mem}_{gen}(\mathcal{F}) = n$. However, if Eve wins some ε-free game $\mathcal{G}$, she can force a victory using only 2 memory states. The idea is the following: since the game is ε-free, from each position of the game, Eve can directly produce one colour $c \in \Gamma_n$. Moreover, as she wins the game $\mathcal{G}$, she also has a strategy to force to see a different colour c' from that position. She just has to remember if she has to follow the strategy to see c', or if she can directly produce the colour c. This can be done with just two memory states, ensuring that the produced play will have at least two different colours. ◀

▶ Remark 25. The condition of the previous proof also provides an example of a condition that is half-positional over ε-free state-coloured arenas, but for which we might need memory n in general state-coloured arenas (other examples for state-coloured games can be found in [29, 18]).

However, the question raised in [18] of whether there can be conditions (that cannot be Muller ones) that are half-positional only over ε-free games remains open.

4 The chromatic memory requirements of Muller conditions

In this section we present the main contributions concerning the chromatic memory requirements of Muller conditions. In Section 4.1, we prove that the chromatic memory requirements of a Muller condition (even for ε-free games) coincide with the size of a minimal Rabin automaton recognising the Muller condition (Theorem 27). In Section 4.2 we deduce that determining the chromatic memory requirements of a Muller condition is NP-complete, for different representations of the condition. Finally, this results allow us to answer in Section 4.3 the question appearing in [18, 19] of whether the chromatic memory requirements coincide with the general memory requirements of winning conditions.

4.1 Chromatic memory and Rabin automata

In this section we prove Theorem 27, establishing the equivalence between the chromatic memory requirements of a Muller condition (also for ε-free games) and the size of a minimal Rabin automaton recognising the associated Muller language.

Lemma 26 appears in Kopczyński's PhD thesis [19, Proposition 8.9] (unpublished). We present a similar proof here.

▶ **Lemma 26** ([19]). *Let $\mathcal{F}$ be a Muller condition. Then, $\mathfrak{mem}_{chrom}(\mathcal{F}) = \mathfrak{mem}_{ind}(\mathcal{F})$. That is, there is an $\mathcal{F}$-game $\mathcal{G}$ won by Eve such that any chromatic memory for $\mathcal{G}$ setting a winning strategy has size at least $\mathfrak{mem}_{ind}(\mathcal{F})$, where $\mathfrak{mem}_{ind}(\mathcal{F})$ is the minimal size of an arena-independent memory for $\mathcal{F}$.*

The same result holds for ε-free games: $\mathfrak{mem}^{\varepsilon\text{-free}}_{chrom}(\mathcal{F}) = \mathfrak{mem}^{\varepsilon\text{-free}}_{ind}(\mathcal{F})$.

Proof. We present the proof for $\mathfrak{mem}_{chrom}(\mathcal{F}) = \mathfrak{mem}_{ind}(\mathcal{F})$, the proof for the ε-free case being identical, since we do not add any ε-transition to the games we consider.

It is clear that $\mathfrak{mem}_{chrom}(\mathcal{F}) \leq \mathfrak{mem}_{ind}(\mathcal{F})$, since any arena-independent memory for $\mathcal{F}$ has to be chromatic. We will prove that it is not the case that $\mathfrak{mem}_{chrom}(\mathcal{F}) < \mathfrak{mem}_{ind}(\mathcal{F})$. Let $\mathcal{M}_1, \cdots, \mathcal{M}_n$ be an enumeration of all chromatic memory structures of size strictly less than $\mathfrak{mem}_{ind}(\mathcal{F})$. By definition of $\mathfrak{mem}_{ind}(\mathcal{F})$, for any of the memories $\mathcal{M}_j$ there is some $\mathcal{F}$-game $\mathcal{G}_j = (V_j, E_j, v_{0_j}, \gamma_j)$ won by Eve such that no function $\texttt{next-move}_{\mathcal{G}_j} : M_j \times V_j \to E_j$ setting a winning strategy in $\mathcal{G}_j$ exists. We define the disjoint union of these games, $\mathcal{G} = \biguplus_{i=1}^{n} \mathcal{G}_i$, as the game with an initial vertex v_0 controlled by Adam, from which he can choose to go to the initial vertex of any of the games $\mathcal{G}_i$ producing the letter $a \in \Gamma$ (for some $a \in \Gamma$ fixed arbitrarily), and such the rest of vertices and transitions of $\mathcal{G}$ is just the disjoint union of those of the games $\mathcal{G}_i$. Eve can win this game, since no matter the choice of Adam we arrive to some game where she can win. However, we show that she cannot win using a chromatic memory strictly smaller than $\mathfrak{mem}_{ind}(\mathcal{F})$. Suppose by contradiction that she wins using a chromatic memory $\mathcal{M} = (M, m_0, \mu)$, $|\mathcal{M}| < \mathfrak{mem}_{ind}(\mathcal{F})$. We let $m_0' = \mu(m_0, a)$, and we consider the memory structure $\mathcal{M}' = (M, m_0', \mu)$. Since $|\mathcal{M}'| < \mathfrak{mem}_{ind}(\mathcal{F})$, $\mathcal{M}' = \mathcal{M}_i$ for some $i \in \{1, \dots, n\}$, and therefore Adam can choose to take the transition leading to $\mathcal{G}_i$, where Eve cannot win using this memory structure. This contradicts the fact that Eve wins $\mathcal{G}$ using $\mathcal{M}$. ◀

▶ **Theorem 27.** *Let $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ be a Muller condition. The following quantities coincide:*

1. *The size of a minimal deterministic Rabin automaton recognising $L_{\mathcal{F}}$, $\mathfrak{rabin}(L_{\mathcal{F}})$.*
2. *The size of a minimal arena-independent memory for $\mathcal{F}$, $\mathfrak{mem}_{ind}(\mathcal{F})$.*
3. *The size of a minimal arena-independent memory for ε-free $\mathcal{F}$-games, $\mathfrak{mem}^{\varepsilon\text{-free}}_{ind}(\mathcal{F})$.*
4. *The chromatic memory requirements of $\mathcal{F}$, $\mathfrak{mem}_{chrom}(\mathcal{F})$.*
5. *The chromatic memory requirements of $\mathcal{F}$ for ε-free games, $\mathfrak{mem}^{\varepsilon\text{-free}}_{chrom}(\mathcal{F})$.*

Proof. The previous Lemma 26, together with Lemma 20, prove that

$$\mathfrak{mem}_{ind}^{\varepsilon\text{-free}}(\mathcal{F}) = \mathfrak{mem}_{chrom}^{\varepsilon\text{-free}}(\mathcal{F}) \leq \mathfrak{mem}_{chrom}(\mathcal{F}) = \mathfrak{mem}_{ind}(\mathcal{F}) \leq \mathfrak{rabin}(L_{\mathcal{F}}).$$

In order to prove that $\mathfrak{rabin}(L_{\mathcal{F}}) \leq \mathfrak{mem}_{ind}^{\varepsilon\text{-free}}(\mathcal{F})$, we are going to show that we can put a Rabin condition on top of any arena-independent memory for ε-free $\mathcal{F}$-games $\mathcal{M}$, obtaining a Rabin automaton recognising $L_{\mathcal{F}}$ and having the same size than $\mathcal{M}$.

Let $\mathcal{M} = (M, m_0, \mu : M \times \Gamma \to M)$ be an arena-independent memory for ε-free $\mathcal{F}$-games. First, we remark that we can suppose that every state of $\mathcal{M}$ is accessible from m_0 by some sequence of transitions. We define a Muller automaton $\mathcal{A}_{\mathcal{M}}$ using the underlying structure of $\mathcal{M}$: $\mathcal{A}_{\mathcal{M}} = (M, \Gamma, m_0, \delta, \Gamma, \mathcal{F})$, where the transition function δ is defined as $\delta(m, a) = (\mu(m, a), a)$, for $a \in \Gamma$. Since the output produced by any word $w \in \Gamma^\omega$ is w itself and the accepting condition is $\mathcal{F}$, this automaton trivially accepts the language $L_{\mathcal{F}}$. We are going to show that the Muller automaton $\mathcal{A}_{\mathcal{M}}$ satisfies the second property in Proposition 3, that is, that for any pair of cycles in $\mathcal{A}_{\mathcal{M}}$ with some state in common, if both are rejecting then their union is also rejecting. This will prove that we can put a Rabin condition on top of $\mathcal{A}_{\mathcal{M}}$.

Let ℓ_1 and ℓ_2 be two rejecting cycles in $\mathcal{A}_{\mathcal{M}}$ such that $m \in M$ is contained in both ℓ_1 and ℓ_2. We suppose by contradiction that their union $\ell_1 \cup \ell_2$ is an accepting cycle. We will build an ε-free $\mathcal{F}$-game that is won by Eve, but where she cannot win using the memory $\mathcal{M}$, leading to a contradiction. Let $a_0 a_1 \dots a_k \in \Gamma^*$ be a word labelling a path to m from m_0 in $\mathcal{M}$, that is, $\mu(m_0, a_0 \dots a_k) = m$. We define the ε-free $\mathcal{F}$-game $\mathcal{G} = (V = V_E, E, v_0, \gamma : E \to \Gamma, \mathcal{F})$ as the game where there is a sequence of transitions labelled with $a_0 \dots a_k$ from v_0 to one vertex v_m controlled by Eve (the only vertex in the game where some player has to make a choice). From v_m, Eve can choose to see all the transitions of ℓ_1 before coming back to m (producing the corresponding colours), or to see all the transitions of ℓ_2 before coming back to m.

First, we notice that Eve can win the game $\mathcal{G}$: since $\ell_1 \cup \ell_2$ is accepting, she only has to alternate between the two choices in the state v_m. However, there is no function $\texttt{next-move} : M \times V_E \to E$ setting up a winning strategy for Eve. Indeed, for every partial play ending in v_m and labelled with $a_0 a_1 \dots a_s$, it is clear that $\mu(m_0, a_0 \dots a_s) = m$ (the memory is at state m). If $\texttt{next-move}(m, v_m)$ is the edge leading to the cycle corresponding to ℓ_1, no matter the value `next-move` takes at the other pairs, all plays will stay in ℓ_1, so the set of colours produced infinitely often would be $\gamma(\ell_1)$ which is loosing for Eve. The result is symmetric if $\texttt{next-move}(m, v_m)$ is the edge leading to the other cycle. We conclude that $\mathcal{M}$ cannot be used as a memory structure for $\mathcal{G}$, a contradiction. ◀

4.2 The complexity of determining the chromatic memory requirements of a Muller condition

As shown in [12], the Zielonka tree of a Muller condition directly gives its general memory requirements. In this section, we see that it follows from the previous results that determining the chromatic memory requirements of a Muller condition is NP-complete, even if it is represented by its Zielonka tree. The proofs can be found in the full version [5].

▶ **Proposition 28.** *Given the Zielonka tree $\mathcal{Z}_{\mathcal{F}}$ of a Muller condition $\mathcal{F}$ (resp. a parity automaton $\mathcal{P}$ recognising $L_{\mathcal{F}}$), we can compute in $\mathcal{O}(|\mathcal{Z}_{\mathcal{F}}|)$ (resp. polynomial time in $|\mathcal{P}|$) the memory requirements for $\mathcal{F}$-games, $\mathfrak{mem}_{gen}(\mathcal{F})$.*

▶ **Theorem 29.** *Given a positive integer $k > 0$ and a Muller condition $\mathcal{F}$ represented as either:*

a) *The Zielonka tree $\mathcal{Z}_{\mathcal{F}}$.*

b) *A parity automaton recognising $L_{\mathcal{F}}$.*

c) *A Rabin automaton recognising $L_{\mathcal{F}}$.*

The problem of deciding whether $\mathfrak{mem}_{chrom}(\mathcal{F}) \geq k$ (or equivalently, $\mathfrak{mem}_{ind}(\mathcal{F}) \geq k$) is NP*-complete.*

The proof consists in showing that the reduction presented in Lemma 13 can also be applied if the Muller condition is given by any of the representations considered in Theorem 29.

4.3 Chromatic memories require more states than general ones

In his PhD Thesis [18, 19], Kopczyński raised the question of whether the general memory requirements of every winning condition coincides with its chromatic memory requirements. In this section we prove that this is not the case: Eve needs strictly more memory if she is restricted to use chromatic memories, and the difference can be arbitrarily large.

▶ **Proposition 30.** *For each integer $n \geq 2$, there exists a set of colours Γ_n and a Muller condition $\mathcal{F}_n$ over Γ_n such that for any $\mathcal{F}_n$-game won by Eve, she can win it using a memory of size 2, but there is an $\mathcal{F}_n$-game $\mathcal{G}$ where Eve needs a chromatic memory of size n to win. Moreover, the game $\mathcal{G}$ can be chosen to be ε-free.*

Proof. Let $\Gamma_n = \{1, 2, \dots, n\}$ be a set of n colours, and let us define the Muller condition $\mathcal{F}_n = \{A \subseteq \Gamma_n \ : \ |A| = 2\}$. The Zielonka tree of $\mathcal{F}_n$ is depicted in Figure 1, where round nodes represent nodes whose label is an accepting set, and rectangular ones, nodes whose label is a rejecting set.

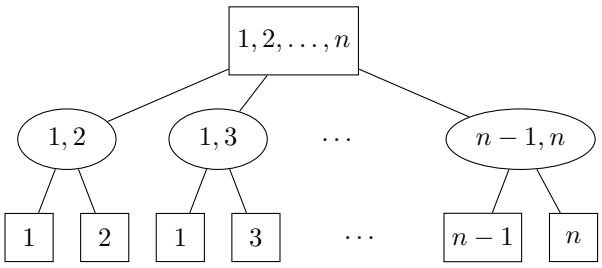


Figure 1 Zielonka tree for the condition $\mathcal{F}_n = \{A \subseteq \{1, 2, \dots, n\} \ : \ |A| = 2\}$. Square nodes are associated with rejecting sets ($A \notin \mathcal{F}_n$) and round nodes with accepting ones ($A \in \mathcal{F}_n$).

The characterisation of the memory requirements of Muller conditions from Proposition 22 gives that $\mathfrak{mem}_{gen}(\mathcal{F}_n) = 2$.

On the other hand, the language $L_{\mathcal{F}_n}$ associated to this condition coincides with the language L_G (defined in Section 2.2) associated to a graph G that is a clique of size n. By Lemma 13, the size of a minimal Rabin automaton recognising $L_{\mathcal{F}_n}$ (and therefore, by Theorem 27, the chromatic memory requirements of $\mathcal{F}_n$) coincides with the chromatic number of G. Since G is a clique of size n, its chromatic number is n. ◀

In the full version [5] we provide an explicit example of such a game.

5 Conclusions and open questions

In this work, we have fully characterised the chromatic memory requirements of Muller conditions, proving that arena-independent memory structures for a given Muller condition correspond to Rabin automata recognising that condition. We have also answered several open questions concerning the memory requirements of Muller conditions when restricting ourselves to chromatic memories or to ε-free games. We have proven the NP-completeness of the minimisation of transition-based Rabin automata and that we can minimise parity automata recognising Muller languages in polynomial time, advancing in our understanding on the complexity of decision problems related to transition-based automata.

The question of whether we can minimise transition-based parity or Büchi automata in polynomial time remains open. The contrast between the results of Abu Radi and Kupferman [1, 2], showing that we can minimise GFG transition-based co-Büchi automata in polynomial time and those of Schewe [27], showing that minimising GFG state-based co-Büchi automata is NP-complete; as well as the contrast between Theorem 14 and Proposition 16, make of this question a very intriguing one.

Regarding the memory requirements of games, we have shown that forbidding ε-transitions might cause a reduction in the memory requirements of Muller conditions. However, the question raised by Kopczyński in [18] remains open: are there prefix-independent winning conditions that are half-positional when restricted to ε-free games, but not when allowing ε-transitions?

References

1 Bader Abu Radi and Orna Kupferman. Minimizing GFG transition-based automata. In *ICALP*, volume 132, pages 100:1–100:16, 2019. doi:10.4230/LIPIcs.ICALP.2019.100.

2 Bader Abu Radi and Orna Kupferman. Canonicity in GFG and transition-based automata. In *GandALF*, volume 326, pages 199–215, 2020. doi:10.4204/EPTCS.326.13.

3 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. In *CONCUR*, volume 171 of *LIPIcs*, pages 24:1–24:22, 2020. doi:10.4230/LIPIcs.CONCUR.2020.24.

4 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO*, pages 495–506, 1999. doi:10.1051/ita:1999129.

5 Antonio Casares. On the minimisation of transition-based Rabin automata and the chromatic memory requirements of Muller conditions. *CoRR*, abs/2105.12009, 2021. arXiv:2105.12009.

6 Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of games and automata using Muller conditions. In *ICALP*, volume 198, pages 123:1–123:14, 2021. doi:10.4230/LIPIcs.ICALP.2021.123.

7 Edmund M. Clarke, I. A. Draghicescu, and Robert P. Kurshan. A unified approch for showing language inclusion and equivalence between various types of omega-automata. *Inf. Process. Lett.*, 46(6):301–308, 1993. doi:10.1016/0020-0190(93)90069-L.

8 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, Cham, 2018. doi:10.1007/978-3-319-10575-8_2.

9 Thomas Colcombet, Nathanaël Fijalkow, and Florian Horn. Playing safe. In *FSTTCS*, volume 29, pages 379–390, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.379.

10 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theoretical Computer Science*, 352(1):190–196, 2006. doi:10.1016/j.tcs.2005.10.046.

11 Thomas Colcombet and Konrad Zdanowski. A tight lower bound for determinization of transition labeled Büchi automata. In *ICALP*, pages 151–162, 2009. doi:10.1007/978-3-642-02930-1_13.

12 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997. doi:10.1109/LICS.1997.614939.

13 Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of ltl formulae to Büchi automata. In *FORTE*, pages 308–326, 2002.

14 Hugo Gimbert and Wieslaw Zielonka. Games where you can play optimally without any memory. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442, 2005. doi:10.1007/11539452_33.

15 John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford University, 1971. doi:10.5555/891883.

16 Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. The IBM Research Symposia Series. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.

17 Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Annals of Pure and Applied Logic*, 69(2):243–268, 1994. doi:10.1016/0168-0072(94)90086-8.

18 Eryk Kopczyński. Half-positional determinacy of infinite games. In *ICALP*, pages 336–347, 2006. doi:10.1007/11787006_29.

19 Eryk Kopczyński. Half-positional determinacy of infite games. PhD Thesis. *University of Warsaw*, 2008.

20 Christof Löding. Efficient minimization of deterministic weak omega-automata. *Inf. Process. Lett.*, 79(3):105–109, 2001. doi:10.1016/S0020-0190(00)00183-6.

21 Philipp Meyer and Salomon Sickert. On the optimal and practical conversion of Emerson-Lei automata into parity automata. *Personal Communication*, 2021.

22 David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF*, pages 180–194, 2017. doi:10.4204/EPTCS.256.13.

23 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.

24 Schmuel Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327, 1988. doi:10.1109/SFCS.1988.21948.

25 Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, pages 167–181, 2009. doi:10.1007/978-3-642-00596-1_13.

26 Sven Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In *FSTTCS*, volume 8 of *LIPIcs*, pages 400–411, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.400.

27 Sven Schewe. Minimising Good-For-Games automata is NP-complete. In *FSTTCS*, volume 182, pages 56:1–56:13, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.56.

28 Klaus Wagner. On ω-regular sets. *Information and control*, 43(2):123–177, 1979. doi:10.1016/S0019-9958(79)90653-3.

29 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.

Fuzzy Algebraic Theories

Davide Castelnovo
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Marino Miculan
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Abstract

In this work we propose a formal system for *fuzzy algebraic reasoning*. The sequent calculus we define is based on two kinds of propositions, capturing equality and existence of terms as members of a fuzzy set. We provide a sound semantics for this calculus and show that there is a notion of free model for any theory in this system, allowing us (with some restrictions) to recover models as Eilenberg-Moore algebras for some monad. We will also prove a completeness result: a formula is derivable from a given theory if and only if it is satisfied by all models of the theory. Finally, leveraging results by Milius and Urbat, we give HSP-like characterizations of subcategories of algebras which are categories of models of particular kinds of theories.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases categorical logic, fuzzy sets, algebraic reasoning, equational axiomatisations, monads, Eilenberg-Moore algebras

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.13

Related Version *Full Version*: https://arxiv.org/abs/2110.10970

Funding *Marino Miculan*: Supported by the Italian MIUR PRIN 2017FTXR7S *IT MATTERS*.

1 Introduction

One of the most fruitful and influential lines of research of Logic in Computer Science is the algebraic study of computation. After Moggi's seminal work [18] showed that notions of computation can be represented as monads, Plotkin and Power [21] approached the problem using operations and equations, i.e., Lawvere theories. Since then, various extensions of the notion of Lawvere theory have been introduced in order to accommodate an ever increasing number of computational notions within this framework; see, e.g., [22, 11, 20], and more recently [3, 4] for *quantitative* algebraic reasoning for probabilistic computations.

Along this line of research, in this work we study algebraic reasoning on *fuzzy sets*. Algebraic structures on fuzzy sets are well known since the seventies (see e.g., [24, 16, 1, 19]). Fuzzy sets are very important in computer science, with applications ranging from pattern recognition to decision making, from system modeling to artificial intelligence. So, one may wonder if it is possible to use an approach similar to above for *fuzzy algebraic reasoning*.

In this paper we answer positively to this question. We propose a sequent calculus based on two kind of propositions, one expressing equality of terms and the other the existence of a term as a member of a fuzzy set. These sequents have a natural interpretation in categories of fuzzy sets endowed with operations. This calculus is sound and complete for such a semantics: a formula is satisfied by all the models of a given theory if and only if it is derivable from it.

It is possible to go further. Both in the classical and in the quantitative settings there is a notion of free model for a theory; we show that is also true for theories in our formal system for fuzzy sets. In general the category of models of a given theory will not be equivalent to the category of Eilenberg-Moore algebras for the induced monad, but we will show that this equivalence holds for theories with sufficiently simple axioms. Finally we will use the techniques developed in [17] to prove two results analogous to Birkhoff's theorem.

Synopsis. In Section 2 we recall the category $\mathbf{Fuz}(H)$ of fuzzy sets over a frame H. Section 3 introduces the syntax and the rules of fuzzy theories. Then, in Section 4 we introduce the notions of algebras for a signature and of models for a theory; in this section we will also show that the calculus proposed is sound and complete. Section 5 is devoted to free models and it is shown that if a theory is *basic* then its category of models arose as the category of Eilenberg-Moore algebras for a monad on $\mathbf{Fuz}(H)$. In Section 6 we use the results of [17] to prove two HSP-like theorems for our calculus. Finally, Section 7 draws some conclusions and directions for future work. Complete proofs are in the extended version [8].

2 Fuzzy sets

In this section we will recall the definition and some well-known properties of the category of fuzzy sets over a frame H (i.e. a complete Heyting algebra [12]).

▶ **Definition 2.1** ([26, 27])**.** *Let H be a frame. A H-*fuzzy set *is a pair (A, μ_A) consisting in a set A and a* membership function *$\mu_A : A \to H$. The* support *of μ_A is the set $\mathsf{supp}(A, \mu_A)$ of elements $x \in A$ such that $\mu_A(x) \neq \bot$. An arrow $f : (A, \mu_A) \to (B, \mu_B)$ is a function $f : A \to B$ such that $\mu_A(x) \leq \mu_B(f(x))$ for all $x \in A$.*

We denote by $\mathbf{Fuz}(H)$ the category of H-fuzzy sets and their arrows. We will often drop the explicit reference to the frame H when there is no danger of confusion.

▶ **Proposition 2.2.** *For any frame H, the forgetful functor $\mathcal{V} : \mathbf{Fuz}(H) \to \mathbf{Set}$ has both a left and a right adjoint ∇ and Δ endowing a set X with the function constantly equal to the bottom and the top element of H, respectively.*

Proof. If $\nabla(X)$ and $\Delta(X)$ are, respectively $(X, c_\bot)$ and $(X, c_\top)$, where $c_\bot$ and $c_\top$ are the functions $X \to H$ constant in $\bot$ and $\top$, then for any $X \in \mathbf{Set}$, $\mathrm{id}_X : \mathcal{V}(\Delta(X)) = X \to X = \mathcal{V}(\nabla(X))$ is the counit of $\mathcal{V} \dashv \Delta$ and the unit of $\nabla \dashv \mathcal{V}$. ◀

▶ **Definition 2.3.** *Let $e : A \to B$ and $m : C \to D$ be two arrows in a category $\mathbf{C}$, we say that m has* the left lifting property with respect to *e if for any two arrows $f : A \to C$ and $g : B \to D$ such that $m \circ f = g \circ e$ there exists a unique $k : B \to C$ with $m \circ k = g$.*

A strong monomorphism *is an arrow m which has the left lifting property with respect to all epimorphisms.*

▶ **Proposition 2.4.** *Let $f : (A, \mu_A) \to (B, \mu_B)$ be an arrow of $\mathbf{Fuz}(H)$, then:*

1. *f is a monomorphism iff it is injective; f is an epimorphism iff it is surjective;*
2. *f is a strong monomorphism iff it is injective and $\mu_B(f(x)) = \mu_A(x)$ for all $x \in A$;*
3. *f is a split epimorphism iff for any $b \in B$ there exists $a_b \in f^{-1}(b)$ such that $\mu_B(b) = \mu_A(a_b)$.*

▶ **Definition 2.5** ([13])**.** *A* proper factorization system *on a category $\mathbf{C}$ is a pair $(\mathcal{E}, \mathcal{M})$ given by two classes of arrows such that:*

- *$\mathcal{E}$ and $\mathcal{M}$ are closed under composition;*
- *every isomorphism belongs to both $\mathcal{E}$ and $\mathcal{M}$;*
- *every $e \in \mathcal{E}$ is an epimorphism and every $m \in \mathcal{M}$ is a monomorphism;*
- *every $m \in \mathcal{M}$ has the left lifting property with respect to every $e \in \mathcal{E}$;*
- *every arrow of $\mathbf{C}$ is equal to $m \circ e$ for some $m \in \mathcal{M}$ and $e \in \mathcal{E}$.*

▶ **Lemma 2.6.** *For any frame H, $\mathbf{Fuz}(H)$ has all products. Moreover the classes of epimorphisms and strong monomorphisms form a proper factorization system on it.*

▶ Remark 2.7. Completeness and the existence of both adjoints to $\mathcal{V}$ can be deduced directly from the fact that $\mathbf{Fuz}(H)$ is topological over $\mathbf{Set}$ [26, Prop. 71.3].

3 Fuzzy Theories

In this section we introduce the syntax and logical rules of fuzzy theories. The first step is to introduce an appropriate notion of signature. Differently from usual signatures, in fuzzy theories constants cannot be seen simply as 0-arity operations, because , as we will see in Section 4, these are interpreted as fuzzy morphisms from the terminal object, and these correspond only to elements whose membership degree is $\top$.

▶ **Definition 3.1.** *A* signature $\Sigma = ()O, \mathsf{ar}, C)$ *is a set* O *of* operations *with an* arity function $\mathsf{ar} : O \to \mathbb{N}_+$ *and a set* C *of* constants. *Signatures form a category* **Sign** *in which an arrow* $\Sigma_1 = (O_1, \mathsf{ar}_1, C_1) \to \Sigma_2 = (O_2, \mathsf{ar}_2, C_2)$ *is a pair* $\mathbf{F} = (F_1, F_2)$ *of functions:* $F_1 : O_1 \to O_2$ *and* $F_2 : C_1 \to C_2$ *with the property that* $\mathsf{ar}_2 \circ F_1 = \mathsf{ar}_1$.

An algebraic language $\mathcal{L}$ *is a pair* (Σ, X) *where* Σ *is a signature and* X *a set. The category* **Lng** *of algebraic languages is just* **Sign** $\times$ **Set**.

▶ **Example 3.2.** The *signature of semigroups* Σ_S in which $O = \{\cdot\}$, $\mathsf{ar}(\cdot) = 2$ and $C = \emptyset$.

▶ **Example 3.3.** The *signature of groups* Σ_G is equal to Σ_S plus an operation $(-)^{-1}$ of arity 1 and a constant e.

Given a language $\mathcal{L}$ we can inductively apply the operations to the set of variables to construct terms, and once terms are built we can introduce equations.

▶ **Definition 3.4.** *Given a language* $\mathcal{L} = (\Sigma, X)$, *the set* $\mathcal{L}$-Terms *is the smallest set s.t.*

- $X \sqcup C \subset \mathcal{L}$-Terms*;*
- *if* $f \in O$ *and* $t_1, \ldots, t_{\mathsf{ar}(f)} \in \mathcal{L}$-Terms *then* $f\big(t_1, \ldots, t_{\mathsf{ar}(f)}\big) \in \mathcal{L}$-Terms.

▶ **Proposition 3.5.** *There exists a functor* $\mathsf{Terms} : \mathbf{Lng} \to \mathbf{Set}$ *sending* $\mathcal{L}$ *to* $\mathcal{L}$-Terms.

▶ **Definition 3.6** (Formulae)**.** *For any language* $\mathcal{L}$ *we define the sets* $\mathsf{Eq}(\mathcal{L})$ *of* equations *as the product* $\mathsf{Eq}(\mathcal{L}) \coloneqq \mathcal{L}$-Terms $\times$ $\mathcal{L}$-Terms *and the set* $\mathsf{M}(\mathcal{L})$ *of* membership propositions *as* $\mathsf{M}(\mathcal{L}) \coloneqq H \times \mathcal{L}$-Terms. *We will write* $s \equiv t$ *for* $(s, t) \in \mathsf{Eq}(\mathcal{L})$ *and* $\mathsf{E}(l, t)$ *for* $(l, t) \in \mathsf{M}(\mathcal{L})$. *The set* $\mathsf{Form}(\mathcal{L})$ *of formulae is then defined as* $\mathsf{Eq}(\mathcal{L}) \sqcup \mathsf{M}(\mathcal{L})$.

Clearly, a proposition $s \equiv t$ means "s and t are equivalent and hence interchangeable"; on the other hand, $\mathsf{E}(l, t)$ intuitively means "the degree of existence of t is at least l".

▶ **Definition 3.7** (Sequent ant fuzzy theory)**.** *A* sequent $\Gamma \vdash \psi$ *is an element* (Γ, ψ) *of* $\mathsf{Seq}(\mathcal{L}) \coloneqq \mathcal{P}(\mathsf{Form}(\mathcal{L})) \times \mathsf{Form}(\mathcal{L})$, *where* $\mathcal{P}$ *is the (covariant) power set functor. A* fuzzy theory in the language $\mathcal{L}$ *is a subset* $\Lambda \subset \mathsf{Seq}(\mathcal{L})$ *and we will use* $\mathsf{Th}(\mathcal{L})$ *for the set* $\mathcal{P}(\mathsf{Seq}(\mathcal{L}))$.

▶ **Notation.** *We will write* $\vdash \phi$ *for* $\emptyset \vdash \phi$.

For any function $\sigma : X \to \mathcal{L}$-Terms *and* $t \in \mathcal{L}$-Terms *we denote* $t[\sigma]$ *the term obtained substituting* $\sigma(x)$ *to any occurence of* x *in* t. *Moreover, for any formula* $\phi \in \mathsf{Form}(\mathcal{L})$ *we define* $\phi[\sigma]$ *as* $t[\sigma] \equiv s[\sigma]$ *if* ϕ *is* $t \equiv s$ *or as* $\mathsf{E}(l, t[\sigma])$ *if* ϕ *is* $\mathsf{E}(l, t)$. *Finally, given* $\Gamma \subset \mathcal{P}(\mathsf{Form}(\mathcal{L}))$ *we put* $\Gamma[\sigma] \coloneqq \{\phi[\sigma] \mid \phi \in \Gamma\}$.

▶ **Definition 3.8.** *For any* $\mathcal{L}$, *the* fuzzy sequent calculus *is given by the rules in Figure 1.*

Given a fuzzy theory Λ, *its* deductive closure $\Lambda^{\vdash}$ *is the smallest subset of* $\mathsf{Seq}(\mathcal{L})$ *which contains* Λ *and it is closed under the rules of fuzzy sequent calculus. A sequent is* derivable from Λ *(or simply derivable if* $\Lambda = \emptyset$*) if it belongs to* $\Lambda^{\vdash}$. *We will write* $\vdash_\Lambda \phi$ *if* $\vdash \phi \in \Lambda^{\vdash}$.

Finally we say that two theories Λ *and* Θ *are* deductively equivalent *if* $\Lambda^{\vdash} = \Theta^{\vdash}$.

$$\frac{\phi \in \Gamma}{\Gamma \vdash \phi}\ \textsc{A} \qquad \frac{\Gamma \vdash \phi}{\Gamma \cup \Delta \vdash \phi}\ \textsc{Weak} \qquad \frac{\{\Gamma \vdash \phi \mid \phi \in \Phi\} \qquad \Phi \vdash \psi}{\Gamma \vdash \psi}\ \textsc{Cut}$$

$$\frac{}{\Gamma \vdash s \equiv s}\ \textsc{Refl} \qquad \frac{\Gamma \vdash s \equiv t}{\Gamma \vdash t \equiv s}\ \textsc{Sym} \qquad \frac{\Gamma \vdash s \equiv t \qquad \Gamma \vdash t \equiv u}{\Gamma \vdash t \equiv u}\ \textsc{Trans}$$

$$\frac{\sigma : X \to \mathcal{L}\text{-Terms} \qquad \Gamma \vdash \psi}{\Gamma[\sigma] \vdash \psi[\sigma]}\ \textsc{Sub} \qquad \frac{f \in O \qquad \{\Gamma \vdash t_i \equiv s_i\}_{i=1}^{\mathsf{ar}(f)}}{\Gamma \vdash f\big(t_1, \ldots, t_{\mathsf{ar}(f)}\big) \equiv f\big(s_1, \ldots, s_{\mathsf{ar}(f)}\big)}\ \textsc{Cong}$$

$$\frac{}{\Gamma \vdash \mathsf{E}(\bot, t)}\ \textsc{Inf} \qquad \frac{\Gamma \vdash \mathsf{E}(l, t)}{\Gamma \vdash \mathsf{E}(l \wedge l', t)}\ \textsc{Mon} \qquad \frac{\{\Gamma \vdash \mathsf{E}(l_i, t_i)\}_{i=1}^{\mathsf{ar}(f)}}{\Gamma \vdash \mathsf{E}\big(\inf\big(\{l_i\}_{i=1}^{n}\big), f\big(t_1, \ldots, t_{\mathsf{ar}(f)}\big)\big)}\ \textsc{Exp}$$

$$\frac{S \subset H \qquad \{\Gamma \vdash \mathsf{E}(l, t)\}_{l \in S}}{\Gamma \vdash \mathsf{E}(\sup(S), t)}\ \textsc{Sup} \qquad \frac{\Gamma \vdash t \equiv s \qquad \Gamma \vdash \mathsf{E}(l, t)}{\Gamma \vdash \mathsf{E}(l, s)}\ \textsc{Fun}$$

Figure 1 Derivation rules for the fuzzy sequent calculus.

The next result shows how maps between languages are lifted to theories.

▶ **Proposition 3.9.** *For any* $\mathbf{F} : \mathcal{L}_1 \to \mathcal{L}_2$ *arrow of* **Lng***:*

1. *there exists a Galois connection* $\mathbf{F}_* \dashv \mathbf{F}^*$ *between* $(\mathsf{Th}(\mathcal{L}_1), \subset)$ *and* $(\mathsf{Th}(\mathcal{L}_2), \subset)$*;*
2. $\mathbf{F}_*\big(\Lambda_1^{\vdash}\big) \subset (\mathbf{F}_*(\Lambda_1))^{\vdash}$ *and* $(\mathbf{F}^*(\Lambda_2))^{\vdash} \subset \mathbf{F}^*\big(\Lambda_2^{\vdash}\big)$ *for any* $\Lambda_1 \in \mathbf{Th}(\mathcal{L}_1)$ *and* $\Lambda_2 \in \mathbf{Th}(\mathcal{L}_2)$*.*

Usually, logics enjoy the so-called "deduction lemma", which allows us to treat elements of a theory on a par with assumptions in sequents. In fuzzy theories, this holds in a slightly restricted form, as proved next.

▶ **Lemma 3.10** (Partial deduction lemma)**.** *Let* Λ *be in* $\mathsf{Th}(\mathcal{L})$ *and* $\Gamma \in \mathcal{P}(\mathsf{Form}(\mathcal{L}))$*, let also* $\Lambda[\Gamma]$ *be the theory* $\Lambda \cup \{\emptyset \vdash \phi \mid \phi \in \Gamma\}$*. Then the following are true:*

1. $\Gamma \cup \Delta \vdash \psi$ *in* $\Lambda^{\vdash}$ *implies* $\Delta \vdash \psi$ *in* $(\Lambda[\Gamma])^{\vdash}$*;*
2. *if* $\Delta \vdash \psi$ *is derivable from* $\Lambda[\Gamma]$ *without using rule* Sub *then* $\Gamma \cup \Delta \vdash \psi$ *is in* $\Lambda^{\vdash}$*.*

Proof.

1. By hypothesis $\Gamma \cup \Delta \vdash \psi$ is in $\Lambda^{\vdash}$ then, since $\Lambda \subset \Lambda[\Gamma]$, it is also in $(\Lambda[\Gamma])^{\vdash}$. Now, for any $\phi \in \Gamma$ and $\theta \in \Delta$ rules Weak and A give

$$\frac{\vdash \phi}{\Delta \vdash \phi}\ \textsc{Weak} \qquad \frac{}{\Delta \vdash \theta}\ \textsc{A}$$

so $\{\Delta \vdash \phi \mid \phi \in \Gamma \cup \Delta\}$ is contained in $(\Lambda[\Gamma])^{\vdash}$ and we can apply Cut to get the thesis:

$$\frac{\{\Delta \vdash \phi \mid \phi \in \Gamma \cup \Delta\} \qquad \Gamma \cup \Delta \vdash \psi}{\Delta \vdash \psi}\ \textsc{Cut}$$

2. Let us proceed by induction on a derivation of $\Delta \vdash \psi$ from $\Lambda[\Gamma]$.
   - If $\Delta \vdash \psi \in \Lambda[\Gamma]$ then or $\Delta \vdash \psi \in \Lambda$ and we are done, or $\psi \in \Gamma$ and we can conclude since $\Gamma \cup \Delta \vdash \phi_i$ is in $\Lambda^{\vdash}$ by rules A and Weak
   - If $\Delta \vdash \psi$ follows from the application of one of the rules A, Inf or Refl then it belongs to the closure of any theory, by Weak this is true even for $\Gamma \cup \Delta \vdash \psi$ which, in particular, it belongs to $\Lambda^{\vdash}$.

- Suppose that $\Delta \vdash \psi$ comes from an application of Weak, then there exists Ψ and Φ such that $\Psi \cup \Phi = \Delta$ and $\Psi \vdash \phi$ is in $(\Lambda[\Gamma])^{\vdash}$. By inductive hypothesis we have the following derivation of $\Gamma \cup \Delta \vdash \psi$ from Λ:

$$\frac{\Gamma \cup \Psi \vdash \psi}{\Gamma \cup \Psi \cup \Phi \vdash \psi}\ \textsc{Weak}$$

- If $\Delta \vdash \psi$ is derived with an application of Cut as last rule then there exists a set Θ such that $\{\Delta \vdash \theta \mid \theta \in \Theta\} \cup \{\Theta \vdash \psi\} \subseteq (\Lambda[\Gamma])^{\vdash}$, therefore, by the inductive hypothesis, we have that $\{\Gamma \cup \Delta \vdash \theta \mid \theta \in \Theta\} \cup \{\Gamma \cup \Theta \vdash \psi\}$ is conteined in $\Lambda^{\vdash}$. Now, $\{\Gamma \cup \Delta \vdash \phi \mid \phi \in \Gamma \cup \Theta\} \subset \Lambda^{\vdash}$ by rule A so an application of Cut gives the thesis:

$$\frac{\{\Gamma \cup \Delta \vdash \phi \mid \phi \in \Gamma \cup \Theta\} \qquad \Gamma \cup \Theta \vdash \psi}{\Gamma \cup \Delta \vdash \psi}\ \textsc{Cut}$$

- Any other rule is of the form

$$\frac{\{\Psi \vdash \xi_j\}_{j \in J}}{\Psi \vdash \xi}\ \mathrm{R}$$

therefore, if $\Delta \vdash \psi$ is derived with an application of one of this rules then the set of its premises must be an element of $(\Lambda[\Gamma])^{\vdash}$ of type $\{\Delta \vdash \xi_j\}_{j \in J}$, so by inductive hypothesis $\{\Gamma \cup \Delta \vdash \theta_j\}_{j \in J} \subseteq \Lambda^{\vdash}$ and then the thesis is witnessed by

$$\frac{\{\Gamma \cup \Delta \vdash \xi_j\}_{j \in J}}{\Gamma \cup \Delta \vdash \psi}\ \mathrm{R}$$

◀

▶ **Example 3.11.** Our first set of running examples is inspired by [19]. Let Σ_S be the signature of semigroups and X a countable set. The theory of *fuzzy semigroups* Λ_S is simply the usual theory of semigroups, i.e given by the sequent (using infix notation)

$$\vdash (x \cdot y) \cdot z \equiv x \cdot (y \cdot z)$$

We get the theory of *left ideal* Λ_{LI} if we add the axioms (one for any $l \in L$):

$$\mathsf{E}(l, y) \vdash \mathsf{E}(l, x \cdot y)$$

Similarly the theory Λ_{RI} of *right ideal* is obtained from the axioms:

$$\mathsf{E}(l, x) \vdash \mathsf{E}(l, x \cdot y)$$

Finally we get the theory of *(bilateral) ideal* Λ_I taking the union of the above theories.

▶ **Example 3.12** ([24, 1, 2])**.** Let Σ_G be the signature of groups and X a countable set. The theory Λ_G of *fuzzy groups* is simply the usual theory of groups, i.e that given by the sequents

$$\vdash x \cdot x^{-1} \equiv e \quad \vdash x^{-1} \cdot x \equiv e \quad \vdash e \cdot x \equiv x \quad \vdash x \cdot x \equiv x \quad \vdash (x \cdot y) \cdot z \equiv x \cdot (y \cdot z)$$

We get the theory Λ_N of *normal fuzzy groups* ([16]) if we add the axioms:

$$\mathsf{E}(l, x) \vdash \mathsf{E}\big(l, y \cdot (x \cdot y^{-1})\big)$$

It can be shown that the sequents $\mathsf{E}(l, x) \vdash \mathsf{E}(l, e)$ and $\mathsf{E}\big(l, y \cdot (x \cdot y^{-1})\big) \vdash \mathsf{E}(l, x)$ are derivable, respectively, from Λ_G and from Λ_N.

4 Fuzzy algebras and semantics

In this section we provide a sound and complete semantics to the syntax and sequents introduced in Section 3. The first step is to define the semantic counterpart of a signature.

▶ **Definition 4.1.** *Given a signature Σ, a* Σ-fuzzy algebra $\mathcal{A} := ((A, \mu_A), \Sigma^{\mathcal{A}})$ *is a fuzzy set (A, μ_A) and a collection $\Sigma^{\mathcal{A}} = \{f^{\mathcal{A}} \mid f \in O\} \sqcup \{c^{\mathcal{A}} \mid c \in C\}$ of arrows:*

$$f^{\mathcal{A}} : (A, \mu_A)^{\mathsf{ar}(f)} \to (A, \mu_A) \qquad c^{\mathcal{A}} : (1, c_\perp) \to (A, \mu_A)$$

where $c_\perp$ is the constant function in $\perp$. A morphism of Σ-fuzzy algebras $\mathcal{A} \to \mathcal{B}$ *is an arrow $g : (A, \mu_A) \to (B, \mu_B)$ such that $g \circ c^{\mathcal{A}} = c^{\mathcal{B}}$ and $f^{\mathcal{B}} \circ g^{\mathsf{ar}(f)} = g \circ f^{\mathcal{A}}$ for every $c \in C$ and $f \in O$. We will write Σ-Alg for the resulting category of Σ-fuzzy algebras.*

▶ Remark 4.2. We will not distinguish between a function from the singleton and its value.

▶ **Definition 4.3.** *Let $\mathcal{L} = (\Sigma, X)$ be a language and $\mathcal{A} = \big((A, \mu_A), \Sigma^{\mathcal{A}}\big)$ be a Σ-algebra.*

An assignment *is simply a function $\iota : X \to A$. We define the* evaluation in $\mathcal{A}$ with respect to ι *as the function $(\text{-})^{\mathcal{A},\iota} : \mathcal{L}\text{-}\mathsf{Terms} \to A$ by induction:*

- $x^{\mathcal{A},\iota} := \iota(x)$ *if $x \in X$;*
- $c^{\mathcal{A},\iota} := c^{\mathcal{A}}$ *if $c \in C$;*
- $(f(t_1, \dots, t_{\mathsf{ar}(f)}))^{\mathcal{A},\iota} := f^{\mathcal{A}}(t_1^{\mathcal{A},\iota}, \dots, t_{\mathsf{ar}(f)}^{\mathcal{A},\iota})$ *if $f \in O$.*

▶ **Proposition 4.4.** *Let $\mathcal{A}$ be a Σ-algebra. Given a function $\sigma : X \to \mathcal{L}\text{-}\mathsf{Terms}$ and an assignment $\iota : X \to A$ define $\iota_\sigma : X \to A$ as the assignment sending x to $(\sigma(x))^{\mathcal{A},\iota}$. Then $\mathcal{A} \vDash_\iota \phi[\sigma]$ if and only if $\mathcal{A} \vDash_{\iota_\sigma} \phi$.*

Proof. This follows at once noticing that $\big(t[\sigma]\big)^{\mathcal{A},\iota} = t^{\mathcal{A},\iota_\sigma}$ holds for every term t. ◀

▶ **Definition 4.5.** $\mathcal{A}$ satisfies $\phi \in \mathsf{Form}(\mathcal{L})$ with respect to ι, *and we write $\mathcal{A} \vDash_\iota \phi$, if ϕ is $\mathsf{E}(l, t)$ and $l \leq \mu_A(t^{\mathcal{A},\iota})$ or if ϕ is $t \equiv s$ and $t^{\mathcal{A},\iota} = s^{\mathcal{A},\iota}$.*

$\mathcal{A}$ satisfies ϕ *if $\mathcal{A} \vDash_\iota \phi$ for all $\iota : X \to A$, and we write $\mathcal{A} \vDash \phi$, similarly, given $\Gamma \subset \mathsf{Form}(\mathcal{L})$, $\mathcal{A} \vDash \Gamma$ ($\mathcal{A} \vDash_\iota \Gamma$) means $\mathcal{A} \vDash \phi$ ($\mathcal{A} \vDash_\iota \phi$) for any $\phi \in \Gamma$.*

Finally, given a sequent $\Gamma \vdash \phi$ we say that $\mathcal{A}$ satisfies it with respect to ι *and we will write $\Gamma \vDash_{\mathcal{A},\iota} \phi$ if $\mathcal{A} \vDash_\iota \phi$ whenever $\mathcal{A} \vDash_\iota \Gamma$; if this happens for all assignments ι we say that $\mathcal{A}$* satisfies *the sequent and we will write $\Gamma \vDash_{\mathcal{A}} \phi$.*

We say that a Σ-fuzzy algebra $\mathcal{A}$ is a model *of a fuzzy theory $\Lambda \in \mathsf{Th}(\mathcal{L})$ if it satisfies all the sequents in it.* $\mathbf{Mod}(\Lambda)$ *denotes the full subcategory of Σ-Alg given by the models of Λ.*

Clearly $\Sigma\text{-}\mathsf{Alg} = \mathbf{Mod}(\emptyset)$. For any $\Lambda \in \mathsf{Th}(\mathcal{L})$ there exist two forgetful functors $\mathcal{U}_\Lambda : \mathbf{Mod}(\Lambda) \to \mathbf{Fuz}(L)$ and $\mathcal{V}_\Lambda : \mathbf{Mod}(\Lambda) \to \mathbf{Set}$. We will write $\mathcal{U}_\Sigma$ and $\mathcal{V}_\Sigma$ for $\mathcal{U}_\emptyset$ and $\mathcal{V}_\emptyset$.

▶ **Proposition 4.6.** *For any signature Σ, $\mathcal{V}_\Sigma$ has a left adjoint $\mathcal{F}_\Sigma^{\mathbf{Set}} : \mathbf{Set} \to \mathbf{Mod}(\Lambda)$.*

Proof. For any set X take the language $\mathcal{L}_X$ and define $\mathcal{F}_\Sigma^{\mathbf{Set}}(X)$ has $(\nabla(\mathcal{L}_X\text{-}\mathbf{Terms}), \Sigma^{\mathcal{F}_\Sigma^{\mathbf{Set}}(X)})$ where $c^{\mathcal{F}_\Sigma^{\mathbf{Set}}(X)} := c$ and for any $f \in O$,

$$f^{\mathcal{F}_\Sigma^{\mathbf{Set}}(X)} : \nabla(\mathcal{L}_X\text{-}\mathbf{Terms})^{\mathsf{ar}(f)} \to \nabla(\mathcal{L}_X\text{-}\mathbf{Terms}) \qquad \big(t_1, \dots, t_{\mathsf{ar}(f)}\big) \mapsto f(t_1, \dots, t_{\mathsf{ar}(f)})$$

It is easy to see that for any $\iota : X \to \mathcal{V}_\Sigma(\mathcal{A})$ the evaluation $(\text{-})^{\mathcal{A},\iota}$ is the unique morphism of Σ-Alg that composed with the inclusion $X \to \mathcal{L}_X\text{-}\mathbf{Terms}$ gives back ι. ◀

We now provide two technical results about interpretations. The first describes how interpretations are moved along morphisms of algebras.

▶ **Proposition 4.7.** *Let $\mathcal{L} = (\Sigma, X)$ be a language, $\Lambda \in \mathsf{Th}(\mathcal{L})$ and $\mathcal{A} = \big((A, \mu_A), \Sigma^{\mathcal{A}}\big)$, $\mathcal{B} = \big((B, \mu_B), \Sigma^{\mathcal{B}}\big)$ be two Σ-algebras. Let also $f : \mathcal{A} \to \mathcal{B}$ be a morphism between them, then:*

1. *$\mathcal{A}$ is a model of Λ if and only if it is a model of $\Lambda^{\vdash}$;*
2. *$f \circ (-)^{\mathcal{A},\iota} = (-)^{\mathcal{B}, f\circ\iota}$ for every assignment $\iota : X \to A$;*
3. *for any assignment $\iota : X \to A$, $\mathcal{A} \vDash_\iota \phi$ entails $\mathcal{B} \vDash_{f\circ\iota} \phi$;*
4. *if $\mathcal{U}_\Sigma(f)$ is a strong monomorphism in $\mathbf{Fuz}(H)$ and $\iota : X \to A$ is an assignment then, for any formula ϕ, $\mathcal{A} \vDash_\iota \phi$ if and only if $\mathcal{B} \vDash_{f\circ\iota} \phi$;*
5. *if $\mathcal{U}_\Sigma(f)$ is a strong monomorphism in $\mathbf{Fuz}(H)$ and $\mathcal{B} \in \mathbf{Mod}(\Lambda)$ then $\mathcal{A} \in \mathbf{Mod}(\Lambda)$.*

We can also move interpretations and theories along morphisms of signatures.

▶ **Definition 4.8.** *For any $\mathbf{F} : \Sigma_1 \to \Sigma_2$ arrow of $\mathbf{Sign}$ and any $\mathcal{A} = \big((A, \mu_A), \Sigma_2^{\mathcal{A}}\big) \in \Sigma_2\text{-}\mathsf{Alg}$, we define $\mathsf{r}_{\mathbf{F}}(\mathcal{A}) = \Big((A, \mu_A), \Sigma_1^{\mathsf{r}_{\mathbf{F}}(\mathcal{A})}\Big) \in \Sigma_1\text{-}\mathsf{Alg}$ putting, for any $f \in O_1$*

$$f^{\mathsf{r}_{\mathbf{F}}(\mathcal{A})} : (A, \mu_A)^{\mathsf{ar}(f)} \to (A, \mu_A) \qquad \big(a_1, \dots, a_{\mathsf{ar}(f)}\big) \mapsto F_2(f)^{\mathcal{A}}\big(a_1, \dots, a_{\mathsf{ar}(f)}\big)$$

and $c^{\mathsf{r}_{\mathbf{F}}(\mathcal{A})} := F_3(c)^{\mathcal{A}}$ for every $c \in C_1$.

▶ **Lemma 4.9.** *Let $\mathcal{L}_1 = (\Sigma_1, X)$ and $\mathcal{L}_2 = (\Sigma_2, Y)$ and $\mathbf{F} = ((F_1, F_2), g) : \mathcal{L}_1 \to \mathcal{L}_2$, then:*

1. *there exists a functor $\mathsf{r}_{\mathbf{F}} : \Sigma_2\text{-}\mathsf{Alg} \to \Sigma_1\text{-}\mathsf{Alg}$ sending $\mathcal{A}$ to $\mathsf{r}_{\mathbf{F}}(\mathcal{A})$;*
2. *$t^{\mathsf{r}_{\mathbf{F}}(\mathcal{A}),\iota\circ g} = (\mathsf{Terms}(\mathbf{F})(t))^{\mathcal{A},\iota}$ for any assignment $\iota : Y \to A$ and $t \in \mathcal{L}_1\text{-}\mathsf{Terms}$;*
3. *for any assignment $\iota : Y \to A$, $\mathsf{r}_{\mathbf{F}}(\mathcal{A}) \vDash_{\iota\circ g} \phi$ if and only if $\mathcal{A} \vDash_\iota \mathsf{Form}(\mathbf{F})(\phi)$;*
4. *If $X = Y$ and $g = \mathrm{id}_X$ then $\mathsf{r}_{\mathbf{F}}$ restricts to a functor $\mathsf{r}_{\mathbf{F},\Lambda} : \mathbf{Mod}(\Lambda) \to \mathbf{Mod}(\mathbf{F}^*(\Lambda))$.*

▶ **Example 4.10.** The models for Λ_S, Λ_{LI}, Λ_{RI} and Λ_I (Example 3.11) are precisely the structures defined in [19], while the models for Λ_G (Example 3.12) are precisely the fuzzy groups as in [24] and those of Λ_N are the structures called *normal fuzzy subgroups* in [2, 1, 16].

Soundness. Now we can proceed proving the soundness of the rules in Figure 1.

▶ **Lemma 4.11.** *Let $\mathcal{L} = (\Sigma, X)$ be a language and $\mathcal{A} = ((A, \mu_A), \Sigma^{\mathcal{A}})$ a Σ-algebra, then:*

1. *for any assignment $\iota : X \to A$ and rule*

$$\frac{\{\Psi_i \vdash \xi_i\}_{i\in I}}{\Psi \vdash \xi}\ R$$

different from Sub, if $\Psi_i \vDash_{\mathcal{A},\iota} \xi_i$ for all $i \in I$ then $\Psi \vDash_{\mathcal{A},\iota} \xi$ too;
2. *for any $\sigma : X \to \mathcal{L}\text{-}\mathsf{Terms}$, if $\Gamma \vDash_{\mathcal{A}} \psi$ then $\Gamma[\sigma] \vDash_{\mathcal{A}} \psi[\sigma]$.*

▶ **Corollary 4.12** (Soundness)**.** *If a Σ-algebra satisfies all the premises of a rule of the fuzzy sequent calculus then it satisfies also its conclusion.*

▶ Remark 4.13. Let us see why the deduction lemma (Lemma 3.10) cannot be extended to rule Sub. Take Σ to be the empty set, $X = \{x, y, z\}$ and $H = \{0, 1\}$. Notice that $\Sigma\text{-}\mathsf{Alg} = \mathbf{Fuz}(H)$. We have the derivation

$$\frac{\vdash x \equiv y}{\vdash x \equiv z}\ \textsc{Sub}$$

If the deduction lemma held for Sub, $x \equiv y \vdash x \equiv z$ would be in $\emptyset^{\vdash}$, hence satisfied by any fuzzy set, but (H, id_H) with $\iota : X \to H$ sending x and y to 0 and z to 1 is a counterexample.

▶ Remark 4.14. Let us take $\Sigma = \emptyset$ and $H = \{0, 1\}$ and $X = \{x, y, z\}$ and the derivation as in Remark 4.13. Now, a fuzzy set (A, μ_A) satisfies $\vdash_\iota x \equiv y$ if and only if $\iota(x) = \iota(y)$, thus, if we take (H, id_H) and the assignment ι of the previous example, then $(H, \mathrm{id}_H) \vdash_\iota x \equiv y$ but it does not satisfy $x \equiv z$ with respect to ι.

Completeness. Now we prove that the calculus we have provided in Section 3 is complete. Let us start with the following observation.

▶ Remark 4.15. For any $\Lambda \in \mathsf{Th}(\mathcal{L})$ the relation $\sim_\Lambda$ given by all t and s such that $\vdash_\Lambda t \equiv s$, is an equivalence relation on $\mathcal{L}$-Terms.

Using this equivalence, we can define the model of terms, as done next.

▶ **Definition 4.16.** *Let $\mathcal{L} = (\Sigma, X)$ be a language and $\Lambda \in \mathsf{Th}(\mathcal{L})$, we define $\mathsf{Terms}(\Lambda)$ to be the quotient of $\mathcal{L}$-Terms by $\sim_\Lambda$, moreover, by rule FUN, the function*

$$\hat{\mu} : \mathcal{L}\text{-}\mathsf{Terms} \to H \qquad t \mapsto \sup\{l \in H \mid \vdash_\Lambda \mathsf{E}(l,t)\}$$

induces a function $\mu_\Lambda : \mathsf{Terms}(\Lambda) \to H$. For any $f \in O$ and $c \in C$ putting $c^{\mathcal{T}_\Lambda} \coloneqq [c]$ and

$$f^{\mathcal{T}_\Lambda} : \mathsf{Terms}(\Lambda)^{\mathsf{ar}(f)}, \to \mathsf{Terms}(\Lambda) \qquad \big([t_1], \ldots, [t_{\mathsf{ar}(f)}]\big) \mapsto \big[f\big(t_1, \ldots, t_{\mathsf{ar}(f)}\big)\big]$$

gives us a Σ-algebra $\mathcal{T}_\Lambda = \big((\mathsf{Terms}(\Lambda), \mu_\Lambda), \Sigma^{\mathcal{T}_\Lambda}\big)$, called the Σ-algebra of terms in Λ. *The* canonical assignment *is the function $\iota_{can} : X \to \mathsf{Terms}(\Lambda)$ sending x to its class $[x]$.*

▶ Remark 4.17. Rule CONG assures us that $f^{\mathcal{T}_\Lambda}$ is well defined while EXP implies that it is an arrow of $\mathbf{Fuz}(H)$.

The following Lemma will be needed to prove completeness.

▶ **Lemma 4.18.** *Let $\mathcal{L} = (\Sigma, X)$ be a language and $\Lambda \in \mathsf{Th}(\mathcal{L})$, then:*

1. *for any $\phi \in \mathsf{Form}(\mathcal{L})$ the following are equivalent:*
   a. $\mathcal{T}_\Lambda \vDash \phi$,
   b. $\mathcal{T}_\Lambda \vDash_{\iota_{can}} \phi$,
   c. $\vdash_\Lambda \phi$;
2. *for any assignment $\iota : X \to \mathsf{Terms}(\Lambda)$ and formula ϕ, $\mathcal{T}_\Lambda \vDash_\iota \phi$ if and only if $\vdash_\Lambda \phi[\sigma \circ \iota]$ for one (and thus any) section σ of the quotient $\mathcal{L}$-$\mathsf{Terms} \to \mathsf{Terms}(\Lambda)$;*
3. *$\mathcal{T}_\Lambda = \big((\mathsf{Terms}(\Lambda), \mu_\Lambda), \Sigma^{\mathcal{T}_\Lambda}\big)$ is a model of Λ.*

Let us start with a technical result.

▶ **Proposition 4.19.** *Let $\mathcal{L} = (\Sigma, X)$ be a language, $\Lambda \in \mathsf{Th}(\mathcal{L})$, and $\sigma : \mathsf{Terms}(\Lambda) \to \mathcal{L}$-$\mathsf{Terms}$ a section of the quotient $\mathcal{L}$-$\mathsf{Terms} \to \mathsf{Terms}(\Lambda)$. The equation $t^{\mathcal{T}_\Lambda, \iota} = [t[\sigma \circ \iota]]$ holds for any assignment $\iota : X \to \mathsf{Terms}(\Lambda)$ and $t \in \mathcal{L}$-Terms. In particular $t^{\mathcal{T}_\Lambda, \iota_{can}} = [t]$.*

Now we can proceed with the proof of Lemma 4.18.

Proof of Lemma 4.18.

1. Let us show the three implications. (a)⇒(b) follows from the definition. For the implication (b)⇒(c) we split the cases.
   - ϕ is $t \equiv s$. Then $\mathcal{T}_\Lambda \vDash_{\iota_{can}} \phi$ means
     $$[t] = t^{\mathcal{T}_\Lambda, \iota_{can}} = s^{\mathcal{T}_\Lambda, \iota_{can}} = [s]$$
     thus $t \sim_\Lambda s$ i.e. $\vdash_\Lambda t \equiv s$.
   - ϕ is $\mathsf{E}(l,t)$. Let S be $\{l' \in H \mid \Lambda \vdash \mathsf{E}(l',t)\}$, by hypothesis $\mathcal{T}_\Lambda \vDash_{\iota_{can}} \phi$, so
     $$l \leq \mu_\Lambda\Big(t^{\mathcal{T}_\Lambda, \iota_{can}}\Big) = \mu_\Lambda([t]) = \sup(S)$$
     hence $l = l \wedge \sup(S)$ and, since H is a frame, $l = \sup(\{l \wedge l' \mid l' \in S\})$, by rule MON we know that that $\vdash_\Lambda \mathsf{E}(l \wedge l', t)$ for all $l' \in S$ and so rule SUP gives us $\vdash_\Lambda \mathsf{E}(l,t)$.

Finally, for (c)⇒(a), let $\iota : X \to \mathsf{Terms}(\Lambda)$ be an assignment and σ a section as in the hypothesis; by rule Sub we get $\vdash_\Lambda \phi[\sigma \circ \iota]$, and by Proposition 4.19 follows the thesis.

2. By Proposition 4.19 we have

$$t^{\mathcal{T}_\Lambda,\iota} = [t[\sigma \circ \iota]] = (t[\sigma \circ \iota])^{\mathcal{T}_\Lambda,\iota_{can}}$$

we can conclude using the previous point.

3. Let $\Gamma \vdash \psi$ be a sequent in Λ with $\Gamma = \{\phi_i\}_{i=1}^n$ and $\iota : X \to \mathsf{Terms}(\Lambda)$ an assignment such that $\mathcal{T}_\Lambda \vDash_\iota \Gamma$. By point 1 above this means that $\vdash_\Lambda \Gamma[\sigma \circ \iota]$ and applying Sub and Cut we can conclude that $\vdash_\Lambda \psi[\sigma \circ \iota]$. By the previous point this is equivalent to $\mathcal{T}_\Lambda \vDash_\iota \psi$. ◀

Since satisfaction of a formula by $\mathcal{T}_\Lambda$ entails its derivability from Λ we can deduce immediately a form of completeness.

▶ **Corollary 4.20** (Completeness for formulae). *For any theory $\Lambda \in \mathsf{Th}(\mathcal{L})$, $\mathcal{A} \vDash \phi$ for all $\mathcal{A} \in \mathbf{Mod}(\Lambda)$ if and only if $\vdash_\Lambda \phi$.*

5 From theories to monads

Given a language $\mathcal{L} = (\Sigma, X)$ and a fuzzy theory $\Lambda \in \mathsf{Th}(\mathcal{L})$ we have a forgetful functor: $\mathcal{U}_\Lambda : \mathbf{Mod}(\Lambda) \to \mathbf{Fuz}(L)$. In this section we first show that it has a left adjoint (Section 5.1) and that for a specific class of theories, models correspond to Eilenberg-Moore algebras for the monad induced by this adjunction (Section 5.2).

5.1 The free fuzzy algebra on a fuzzy set

To give the definition of free models (Definition 5.8) we need some preliminary constructions.

▶ **Definition 5.1.** *Let $\mathcal{A}$ be a Σ-algebra and $f : (B, \mu_B) \to \mathcal{U}_\Sigma(\mathcal{A})$ an arrow in $\mathbf{Fuz}(H)$, a Σ-algebra* generated by *f in $\mathcal{A}$ is a morphism $\epsilon : \langle B, \mu_B\rangle_{\mathcal{A},f} \to \mathcal{A}$ such that:*

- *$\mathcal{U}_\Sigma(\epsilon)$ is strong mono;*
- *there exists $\bar{f} : (B, \mu_B) \to \langle B, \mu_B\rangle_{\mathcal{A},f}$ such that $\mathcal{U}_\Sigma(\epsilon) \circ \bar{f} = f$;*
- *if $g : \mathcal{C} \to \mathcal{A}$ is a morphism such that $\mathcal{U}_\Sigma(g)$ is strong monomorphism and $\mathcal{U}_\Sigma(g) \circ h = f$ for some h then there exists a unique $k : \langle B, \mu_B\rangle_{\mathcal{A},f} \to \mathcal{C}$ such that $g \circ k = \epsilon$.*

We can construct $\langle B, \mu_B\rangle_{\mathcal{A},f}$ closing $f(B)$ under the iterated images of the functions $g^{\mathcal{A}}$, when g varies between the operations in O, so we get easily the following.

▶ **Proposition 5.2.** *For any signature Σ, Σ-algebra $\mathcal{A}$ and $f : (B, \mu_B) \to \mathcal{U}_\Sigma(\mathcal{A})$, $\langle B, \mu_B\rangle_{\mathcal{A},f}$ exists and it is unique up to isomorphism.*

▶ Remark 5.3. Proposition 4.7 implies that, given a model $\mathcal{A} = \big((A, \mu_A), \Sigma^{\mathcal{A}}\big)$ of a theory $\Lambda \in \mathsf{Th}(\mathcal{L})$, and a morphism $f : (B, \mu_B) \to (A, \mu_A)$, the Σ-algebra $\langle B, \mu_B\rangle_{\mathcal{A},f}$ is in $\mathbf{Mod}(\Lambda)$.

The next result follows at once noticing that $\langle B, \mu_B\rangle_{\mathcal{A},f}$ is built from $f(B)$ closing it under the interpretation of elements of O.

▶ **Proposition 5.4.** *Let $\mathcal{A}$ be a Σ-algebra and $f : (B, \mu_B) \to \mathcal{U}_\Sigma(\mathcal{A})$, then, for any other Σ-algebra $\mathcal{C}$ and $h : (B, \mu_B) \to \mathcal{U}_\Sigma(\mathcal{C})$ there exists at most one $k : \langle B, \mu_B\rangle_{\mathcal{A},f} \to \mathcal{C}$ such that $k \circ \bar{f} = h$.*

The next definition explains how to extend a theory in a given language with the data of a fuzzy set.

▶ **Definition 5.5.** *Let (M, μ_M) be a fuzzy set, $\mathcal{L} = (\Sigma, X)$ a language with $\Sigma = (O, \mathsf{ar}, C)$. We define $\Sigma[M, \mu_M]$ to be $(O, \mathsf{ar}, C \sqcup M)$ and $\mathcal{L}_{(M,\mu_M)}$ to be $(\Sigma[M, \mu_M], X)$. We have an obvious morphism $\mathbf{I} : \mathcal{L} \to \mathcal{L}_{(M,\mu_M)}$ given by the identities and the inclusion $i_C : C \to C \sqcup M$.*

For any $\Lambda \in \mathsf{Th}(\mathcal{L})$ we define $\Lambda[M, \mu_M] \in \mathcal{L}_{(M,\mu_M)}$ as $\mathbf{I}_*(\Lambda) \cup \overline{(M, \mu_M)}$ where $\overline{(M, \mu_M)} = \{\vdash \mathsf{E}(l, m) \mid m \in M, l \in L \text{ and } \mu_M(m) \geq l\}$.

▶ Remark 5.6. It is immediate to see that $\mathbf{I}^*(\Lambda[M, \mu_M]) = \Lambda$.

In the next proposition we show that, for any theory Λ, a fuzzy set can be mapped into the term model of the theory Λ extended with it. Hence, the natural candidate to be the free model is the algebra generated by such map.

▶ **Proposition 5.7.** *For any fuzzy set (M, μ_M) and any theory $\Lambda \in \mathsf{Th}(\mathcal{L})$:*

1. *the function $\bar{\eta}_{(M,\mu_M)}$ sending m to the class $[m]$ of the corresponding constant defines an arrow of fuzzy sets $\bar{\eta}_{(M,\mu_M)} : (M, \mu_M) \to \mathsf{Terms}(\Lambda[M, \mu_M])$;*
2. *any element in $\langle M, \mu_M \rangle_{\mathcal{T}_{\Lambda[M,\mu_M]}, \bar{\eta}_{(M,\mu_M)}}$ has a representative without variables;*
3. *$\langle M, \mu_M \rangle_{\mathcal{T}_{\Lambda[M,\mu_M]}, \bar{\eta}_{(M,\mu_M)}}$ is the initial object of $\mathbf{Mod}(\Lambda[M, \mu_M])$.*

▶ **Definition 5.8.** *For any language $\mathcal{L}$, $\Lambda \in \mathsf{Th}(\mathcal{L})$ and $(M, \mu_M) \in \mathbf{Fuz}(H)$ we define the* free model $\mathcal{F}_\Lambda(M, \mu_M)$ of Λ generated by (M, μ_M) *to be* $\mathsf{r}_{\mathbf{I}, \Lambda[M,\mu_M]}\Big(\langle M, \mu_M \rangle_{\mathcal{T}_{\Lambda[M,\mu_M]}, \bar{\eta}_{(M,\mu_M)}}\Big)$. *We set $\mathsf{T}_\Lambda(M, \mu_M)$ to be $\mathcal{U}_\Lambda(\mathcal{F}_\Lambda(M, \mu_M))$.*

Now it is enough to check that the free model just defined actually provides the left adjoint.

▶ **Theorem 5.9.** *For any language $\mathcal{L}$ and $\Lambda \in \mathsf{Th}(\mathcal{L})$ the functor $\mathcal{U}_\Lambda : \mathbf{Mod}(\Lambda) \to \mathbf{Fuz}(L)$ has a left adjoint $\mathcal{F}_\Lambda$.*

Proof. By construction $\bar{\eta}_{(M,\mu_M)}$ factors through $\eta_{(M,\mu_M)} : (M, \mu_M) \to \mathsf{T}_\Lambda(M, \mu_M)$ which sends m to $[m]$. Let now $g : (M, \mu_M) \to \mathcal{U}_\Lambda(\mathcal{B})$ be another arrow in $\mathbf{Fuz}(H)$, we can turn $\mathcal{B}$ into a $\Sigma[M, \mu_M]$-algebra $\mathcal{B}^g$ setting $m^{\mathcal{B}^g}$ to be $g(m)$ for any $m \in M$.

Let us show that $\mathcal{B}^g$ is a model of $\Lambda[M, \mu_M]$. Surely it is a model of Λ since $\mathcal{B}$ is, let $\vdash \mathsf{E}(l, m)$ be a sequent in $\overline{(M, \mu_M)}$, then for any assignment $\iota : V \to B$:

$$\mathcal{B}^g \vDash_\iota \mathsf{E}(l, m) \iff l \leq \mu_B(m^{\mathcal{B}^g, \iota}) l \leq \mu_\Lambda\Big(t^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota}\Big) \iff l \leq \mu_B(g(m))$$

but g is an arrow of $\mathbf{Fuz}(H)$ so $\mu_B(g(m)) \geq \mu_M(m)$ and we are done.

Now, since $\mathcal{B}^g$ is a model of $\Lambda[M, \mu_M]$, we can take $\bar{g}$ to be the image through $\mathsf{r}_{\mathbf{I}, \Lambda[M,\mu_M]}$ of the unique arrow $\langle M, \mu_M \rangle_{\mathcal{T}_{\Lambda[(M,\mu_M)]}, \bar{\eta}_{(M,\mu_M)}} \to \mathcal{B}^g$, by construction

$$\bar{g}(\eta_{(M,\mu_M)}(m)) = \bar{g}([m]) = m^{\mathcal{B}^g} = g(m)$$

so $\mathcal{U}_\Lambda(\bar{g}) \circ \eta_{(M,\mu_M)} = g$. Uniqueness follows from Proposition 5.7. ◀

▶ **Definition 5.10.** *Given a theory $\Lambda \in \mathsf{Th}(\mathcal{L})$, its* associated monad $\mathsf{T}_\Lambda : \mathbf{Fuz}(H) \to \mathbf{Fuz}(H)$ *is the composite $\mathcal{U}_\Lambda \circ \mathcal{F}_\Lambda$.*

▶ Remark 5.11. If Λ is the empty theory (in any language), then, by Proposition 4.6, the compositttion $\mathcal{F}_\emptyset \circ \nabla$ gives us a functor isomorphic to $\mathcal{F}_\Sigma^{\mathbf{Set}}$.

▶ **Notation.** *We will denote by $\mathcal{F}_\emptyset$ with $\mathcal{F}_\Sigma$ and with T_Σ the monad $\mathsf{T}_\emptyset = \mathcal{U}_\Sigma \circ \mathcal{F}_\Sigma$.*

In this setting we can provide a result similar to Lemma 4.18.

▶ **Lemma 5.12.** *For any language $\mathcal{L} = (\Sigma, X)$ we define the* natural assignment ι_{nat} *as the adjoint to the unit $\nabla(X) \to \mathsf{T}_\Lambda(\nabla(X))$. Then $\mathcal{F}_\Lambda(\nabla(X)) \vDash_{\iota_{nat}} \phi$ if and only if $\vdash_\Lambda \phi$.*

Proof. The implication from the right to the left follows immediately since $\mathcal{F}_\Lambda(\nabla(X))$ is a model for Λ. By adjointness he canonical assignment ι_{can} induces an arrow $\nabla(X) \to \mathcal{U}_{\Lambda[\nabla(X)]}(\mathcal{T}_{\Lambda[\nabla(X)]})$, which, in turn, induces a morphism $e : \mathcal{F}_\Lambda(\nabla(X)) \to \mathcal{T}_{\Lambda[\nabla(X)]}$ of Σ-algebras such that, as function between sets, $e \circ \iota_{nat} = \iota_{can}$. Recalling that $\mathbf{I}$ is the arrow $(\Sigma, X) \to (\Sigma[\nabla(X)], X)$ and using Proposition 4.7, Lemma 4.9 and Lemma 4.18:

$$\begin{aligned}
&\mathcal{F}_\Lambda(\nabla X) \vDash_{\iota_{nat}} \phi \iff \mathsf{r}_{\mathbf{I},\Lambda[\nabla(X)]}\big(\langle\nabla(X)\rangle_{\mathcal{T}_{\Lambda[\nabla(X)]},\bar{\eta}_{\nabla(X)}}\big) \vDash_{\iota_{nat}} \phi \\
&\iff \mathsf{r}_{\mathbf{I}}\big(\langle\nabla(X)\rangle_{\mathcal{T}_{\Lambda[\nabla(X)]},\bar{\eta}_{\nabla(X)}}\big) \vDash_{\iota_{nat}} \phi \iff \langle\nabla(X)\rangle_{\mathcal{T}_{\Lambda[\nabla(X)]},\bar{\eta}_{\nabla(X)}} \vDash_{\iota_{nat}} \phi \\
&\implies \mathcal{T}_{\Lambda[\nabla(X)]} \vDash_{e\circ\iota_{nat}} \phi \iff \mathcal{T}_{\Lambda[\nabla(X)]} \vDash_{\iota_{can}} \phi \iff \vdash_{\Lambda[\nabla(X)]} \phi
\end{aligned}$$

Now, by definition $\overline{\nabla(X)}$ is equal to $\{\vdash \mathsf{E}(\bot, x) \mid x \in X\}$, therefore $(\Lambda[\nabla(X)])^\vdash = \Lambda^\vdash$ and we get the thesis. ◀

5.2 Eilenberg-Moore algebras and models

In this section we will compare the category $\mathbf{Mod}(\Lambda)$ of models of some $\Lambda \in \mathsf{Th}(\mathcal{L})$ and $\mathbf{Alg}(\mathsf{T}_\Lambda)$ of Eilenberg-Moore algebras for the corresponding monad T_Λ. First of all we recall the following classic lemma ([7, Prop. 4.2.1] and [14, Theorem VI.3.1]).

▶ **Lemma 5.13.** *Let $\mathcal{L} : \mathbf{C} \to \mathbf{D}$ be a functor with right adjoint $\mathcal{R}$ and let $\mathsf{T} = \mathcal{R} \circ \mathcal{L}$ be its associated monad, then there exists a* comparison functor *$\mathcal{K} : \mathbf{D} \to \mathbf{Alg}(\mathsf{T})$ such that $\mathcal{U}_\mathsf{T} \circ \mathcal{K} = \mathcal{R}$, where $\mathcal{U}_\mathsf{T} : \mathbf{Alg}(\mathsf{T}) \to \mathbf{C}$ is the forgetful functor. $\mathcal{K}$ sends D in $(\mathcal{R}(D), \mathcal{R}(\epsilon_D))$, where ϵ is the counit of the adjunction.*

As a consequence, for any theory Λ we have a functor from $\mathbf{Mod}(\Lambda)$ to $\mathbf{Alg}(\mathsf{T}_\Lambda)$. We want to construct an inverse of such functor.

▶ **Definition 5.14.** *Let Λ be in $\mathsf{Th}(\mathcal{L})$ and $\xi : \mathsf{T}_\Lambda(M, \mu_M) \to (M, \mu_M)$ an object of $\mathbf{Alg}(\mathsf{T}_\Lambda)$, we define its* associated algebra *$\mathcal{H}(\xi) = \Big((M, \mu_M), \Sigma^{\mathcal{H}(\xi)}\Big)$ putting, for every $c \in C$ and $f \in O$:*

$$c^{\mathcal{H}(\xi)} := \xi\Big(c^{\mathcal{F}_\Lambda(X,\mu_X)}\Big) \qquad f^{\mathcal{H}(\xi)} := \xi \circ f^{\mathcal{F}_\Lambda(X,\mu_X)} \circ \eta^{\mathsf{ar}(f)}_{(M,\mu_M)}$$

▶ **Lemma 5.15.** *For any Eilenberg-Moore algebra $\xi : \mathsf{T}_\Lambda(M, \mu_M) \to (M, \mu_M)$, ξ itself is an arrow $\mathcal{F}_\Lambda(X, \mu_X) \to \mathcal{H}(\xi)$ of Σ-Alg. In particular, for every term t and assignment ι:*

$$t^{\mathcal{H}(\xi),\iota} = \xi\Big(t^{\mathcal{F}_\Lambda(M,\mu_M),\eta_{(M,\mu_M)}\circ\iota}\Big)$$

Proof. The proof of the first half is a straightforward calculation. The second half follows from point 2 of Proposition 4.7 applied to ξ noticing that $\iota = \xi \circ \eta_{(M,\mu_M)} \circ \iota$. ◀

In general $\mathcal{H}(\xi)$ is not a model of Λ, but we can restrict to a class of theories such this holds. As in [4, 15], we consider theories whose sequents' premises contain only variables.

▶ **Definition 5.16.** *A theory $\Lambda \in \mathsf{Th}(\mathcal{L})$ is* basic[1] *if, for any sequent $\Gamma \vdash \phi$ in it, all the formulae in Γ contain only variables.*

▶ **Example 5.17.** Fuzzy groups, fuzzy normal groups, fuzzy semigroups and left, right, bilateral ideals (Examples 3.11 and 3.12) are all examples of basic theories.

[1] In [3] such theories are called *simple*.

▶ **Lemma 5.18.** *$\mathcal{H}(\xi)$ is a model of Λ for any basic theory $\Lambda \in \mathsf{Th}(\mathcal{L})$ and Eilenberg-Moore algebra $\xi : \mathsf{T}_\Lambda(M, \mu_M) \to (M, \mu_M)$.*

Proof. We start observing that if $\Gamma \vdash \phi$ is in Λ and $\iota : X \to M$ is an assignment such that $\mathcal{H}(\xi) \vDash_\iota \Gamma$ then $\mathcal{F}_\Lambda(M, \mu_M) \vDash_{\eta_{(M,\mu_M)} \circ \iota} \Gamma$. Indeed, ψ in Γ can be or $x \equiv y$, and in such case $\iota(x) = \iota(y)$ implies the thesis, or ψ is $\mathsf{E}(l, x)$, but then we can conclude since the membership degree of $\eta_{(M,\mu_M)}(\iota(x))$ in $\mathsf{T}_\Lambda(M, \mu_M)$ is greater than $\mu_M(\iota(x))$. Therefore, we know that $\mathcal{F}_\Lambda(M, \mu_M) \vDash_{\eta_{(M,\mu_M)} \circ \iota} \phi$. Let us split again the two cases.

- ϕ is $t \equiv s$. In this case, $t^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota} = s^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota}$, therefore

$$t^{\mathcal{H}(\xi),\iota} = \xi\left(t^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota}\right) = \xi\left(s^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota}\right) = s^{\mathcal{H}(\xi),\iota}$$

- ϕ is $\mathsf{E}(l, t)$. This means that $l \leq \mu_\Lambda\left(t^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota}\right)$, hence, thus:

$$l \leq \mu_\Lambda\left(t^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota}\right) \leq \mu_M\left(\xi\left(t^{\mathcal{F}_\Lambda(M,\mu_M), \eta_{(M,\mu_M)} \circ \iota}\right)\right) = \mu_M\left(t^{\mathcal{H}(\xi),\iota}\right)$$

and we can conclude. ◀

▶ **Theorem 5.19.** *For any basic theory $\Lambda \in \mathsf{Th}(\mathcal{L})$, the functor $\mathcal{K} : \mathbf{Mod}(\Lambda) \to \mathbf{Alg}(\mathsf{T}_\Lambda)$ has an inverse $\mathcal{H} : \mathbf{Alg}(\mathsf{T}_\Lambda) \to \mathbf{Mod}(\Lambda)$ sending $\xi : \mathsf{T}_\Lambda(M, \mu_M) \to (M, \mu_M)$ to $\mathcal{H}(\xi)$.*

Proof (sketches). We have already constructed the inverse $\mathcal{H}$ on objects. If Λ is basic it can be extended to a functor $\mathbf{Alg}(\mathsf{T}_\Lambda) \to \mathbf{Mod}(\Lambda)$ defining its action on arrows as the identity. A straightforward calculation now shows that $\mathcal{K} \circ \mathcal{H} = \mathrm{id}_{\mathbf{Alg}(\mathsf{T}_\Lambda)}$ and $\mathcal{H} \circ \mathcal{K} = \mathrm{id}_{\mathbf{Mod}(\Lambda)}$. ◀

▶ **Corollary 5.20.** *For any basic theory $\Lambda \in \mathsf{Th}(\mathcal{L})$, $\mathbf{Alg}(\mathsf{T}_\Lambda)$ and $\mathbf{Mod}(\Lambda)$ are isomorphic, and thus equivalent, categories.*

6 Equational axiomatizations

In this section we prove two results for our calculus analogous to the classic HSP theorem [5], using the results by Milius and Urbat [17].

The abstract framework. Let us start recalling the tools introduced in [17], adapted to our situation. In the following we will fix a tuple[2] $\left(\mathbf{C}, \left(\mathcal{E}, \mathcal{M}\right), \mathcal{X}\right)$ where $\mathbf{C}$ is a category, $\left(\mathcal{E}, \mathcal{M}\right)$ is a proper factorization system on $\mathbf{C}$ and $\mathcal{X}$ is a class of objects of $\mathbf{C}$.

▶ **Definition 6.1.** *An object X of $\mathbf{C}$ is* projective with respect to an arrow $f : A \to B$ *if for any $h : X \to B$ there exists a $k : X \to A$ such that $f \circ k = h$.*

*We define $\mathcal{E}_\mathcal{X}$ as the class of $e \in \mathcal{E}$ such that for every $X \in \mathcal{X}$, X is projective with respect to e. A $\mathcal{E}_\mathcal{X}$-*quotient *is just an arrow in $\mathcal{E}_\mathcal{X}$.*

In the rest of the section, we assume that $\left(\mathbf{C}, \left(\mathcal{E}, \mathcal{M}\right), \mathcal{X}\right)$ satisfies the following requirements:

- $\mathbf{C}$ has all (small) products;
- for any $X \in \mathcal{X}$, the class $X{\downarrow}\mathbf{C}$ of all $e \in \mathcal{E}$ with domain X is essentially small, i.e. there is a set $\mathcal{J} \subset X{\downarrow}\mathbf{C}$ such that for any $e : X \to C \in X{\downarrow}\mathbf{C}$ there exists $e' : X \to D \in \mathcal{J}$ and an isomorphism ϕ such that $\phi \circ e = e'$;
- for every object C of $\mathbf{C}$ there exists $e : X \to C$ in $\mathcal{E}_\mathcal{X}$ with $X \in \mathcal{X}$.

[2] In their work Milius and Urbat additionaly require a full subcategory of $\mathbf{C}$ and a fixed class of cardinals, but we will not need this level of generality.

▶ **Definition 6.2.** *An* $\mathcal{X}$-equation *is an arrow* $e \in X{\downarrow}\mathbf{C}$ *with* $X \in \mathcal{X}$*. We say that an object* A *of* $\mathbf{C}$ satisfies $e : X \to C$*, and we write* $A \vDash_{\mathcal{X}} e$*, if for every* $h : X \to A$ *there exists* $q : C \to A$ *such that* $q \circ e = h$*. Given a class* $\mathbb{E}$ *of* $\mathcal{X}$*-equations, we define* $\mathcal{V}(\mathbb{E})$ *as the full subcategory of* $\mathbf{C}$ *given by objects that satisfy* e *for every* $e \in \mathbb{E}$*. A full subcategory* $\mathbf{V}$ *is* $\mathcal{X}$-equationally presentable *if there exists* $\mathbb{E}$ *such that* $\mathbf{V} = \mathcal{V}(\mathbb{E})$*.*

▶ Remark 6.3. The definition of equation in [17, Def. 3.3] is given in terms of suitable subclasses of $X{\downarrow}\mathbf{C}$. However in our setting Milius and Urbat's definition reduces to ours (cfr. [17, Remark 3.4]).

▶ **Theorem 6.4** ([17, Th. 3.15, 3.16])**.** *A full subcategory* $\mathbf{V}$ *of* $\mathbf{C}$ *is* $\mathcal{X}$*-equationally presentable if and only if it is closed under* $\mathcal{E}_{\mathcal{X}}$*-quotients,* $\mathcal{M}$*-subobjects and (small) products.*

Application to fuzzy algebras. In order to apply the results above to Σ-Alg, we need to define the required inputs, i.e., to specify a factorization system and a class of Σ-algebras.

▶ **Lemma 6.5.** *For any* Σ*, the classes* $\mathcal{E}_\Sigma \coloneqq \{e \text{ map of } \Sigma\text{-Alg} \mid \mathcal{U}_\Sigma(e) \text{ is epi}\}$ *and* $\mathcal{M}_\Sigma \coloneqq \{m \text{ map of } \Sigma\text{-Alg} \mid \mathcal{U}_\Sigma(m) \text{ is strong mono}\}$ *form a proper factorization system on* Σ-Alg*.*

▶ **Definition 6.6.** *We define the following two classes of* Σ*-algebras:*

$$\mathcal{X}_0 \coloneqq \left\{\mathcal{F}_\Sigma^{\mathbf{Set}}(X) \mid X \in \mathbf{Set}\right\}$$
$$\mathcal{X}_{\mathsf{E}} \coloneqq \{\mathcal{F}_\Sigma(X, \mu_X) \mid (X, \mu_X) \in \mathbf{Fuz}(H)\}$$

We will use $\mathcal{E}_{\Sigma,\mathcal{X}_0}$ *(resp.,* $\mathcal{E}_{\Sigma,\mathcal{X}_{\mathsf{E}}}$*) for the class of* $e \in \mathcal{E}$ *such that every* $X \in \mathcal{X}_0$ *(resp.* $X \in \mathcal{X}_{\mathsf{E}}$*) is projective with respect to* e*.*

▶ Remark 6.7. $\mathcal{X}_0 = \{\mathcal{F}_\Sigma(X, \mu_X) \mid \mathsf{supp}(X, \mu_X) = \emptyset\}$.

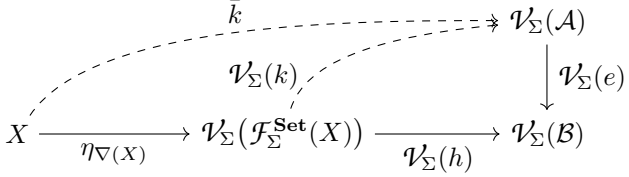
We have now all the ingredients needed to use the results recalled above.

▶ **Lemma 6.8.** *With the above definitions:*

1. $\mathcal{E}_{\Sigma,\mathcal{X}_0} = \mathcal{E}_\Sigma$*;*
2. $\mathcal{E}_{\Sigma,\mathcal{X}_{\mathsf{E}}} = \{e \in \mathcal{E}_\Sigma \mid \mathcal{U}_\Sigma(e) \text{ is split}\}$*;*
3. $(\Sigma\text{-Alg}, (\mathcal{E}_\Sigma, \mathcal{M}_\Sigma), \mathcal{X}_0)$ *and* $(\Sigma\text{-Alg}, (\mathcal{E}_\Sigma, \mathcal{M}_\Sigma), \mathcal{X}_{\mathsf{E}})$ *satisfy the conditions of our settings.*

Proof.

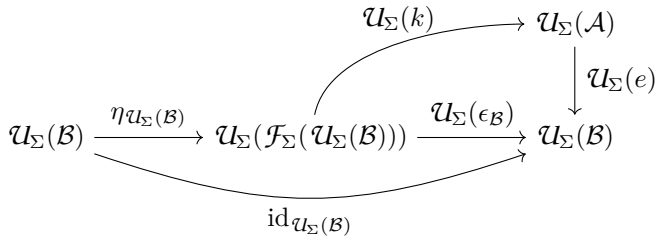
1. Let $e : \mathcal{A} \to \mathcal{B}$ be an arrow in $\mathcal{E}_\Sigma$ and let $h : \mathcal{F}_\Sigma^{\mathbf{Set}}(X) \to \mathcal{B}$ be any morphism of Σ-Alg. By point 2 of Proposition 2.4 e is surjective so for any $x \in X$ we can take a $a_x \in e^{-1}(h(\eta_X(x)))$, where η is the unit of the adjunction $\mathcal{F}_\Sigma^{\mathbf{Set}} \dashv \mathcal{V}_\Sigma$ of Proposition 4.6, and define $\bar{k} : X \to A$ mapping x to a_x, where $\mathcal{A} = \big((A, \mu_A), \Sigma^{\mathcal{A}}\big)$, this induces $k : \mathcal{F}_\Sigma^{\mathbf{Set}}(X) \to \mathcal{A}$ and $(e \circ k) \circ \eta_X = e \circ \bar{k} = h \circ \eta_X$, thus $e \circ k = h$.



2. Let $e : \mathcal{A} \to \mathcal{B}$ be in $\mathcal{E}_\Sigma$ such that $\mathcal{U}_\Sigma(e)$ is split and let s be a section in $\mathbf{Fuz}(H)$, then, for any $h : \mathcal{F}_\Sigma(X, \mu_X) \to \mathcal{B}$ we can consider the arrow $s \circ h \circ \eta_{(X,\mu_X)}$, which, by adjointness provides a $k : \mathcal{F}_\Sigma(X, \mu_X) \to \mathcal{A}$, moreover:

$$e \circ k \circ \eta_{(X,\mu_X)} = e \circ s \circ h \circ \eta_{(X,\mu_X)} = (e \circ s) \circ \big(h \circ \eta_{(X,\mu_X)}\big) = h \circ \eta_{(X,\mu_X)}$$

so k is the wanted lifting. On the other hand, if e is in $\mathcal{E}_{\Sigma,\mathcal{X}_1}$ we can take the diagram:



where $\epsilon_\mathcal{B}$ is the component of the counit $\epsilon : \mathcal{F}_\Sigma \circ \mathcal{U}_\Sigma \to \mathrm{id}_{\Sigma\text{-}\mathsf{Alg}}$ and k its lifting. Taking $\mathcal{U}_\Sigma(k) \circ \eta_{\mathcal{U}_\Sigma(\mathcal{B})}$ we get the desired section of $\mathcal{U}_\Sigma(e)$.

3. First, notice that $\mathbf{Fuz}(H)$ has all products by Lemma 2.6. Moeover, it can be shown that $X{\downarrow}\mathbf{C}$ is essentially small. For any fuzzy set (X, μ_X) we can consider the identity $\mathrm{id}_{(X,\mu_X)} : (X, \mu_X) \to (X, \mu_X)$ and the counit $\epsilon_{(X,\mu_X)} : \nabla(X) \to (X, \mu_X)$ of the adjunction $\nabla \dashv \mathcal{U}$ of Proposition 2.2. They induce arrows $e_0 : \mathcal{F}_\Sigma^{\mathbf{Set}}(X) \to (X, \mu_X)$ and $e_\mathsf{E} : \mathcal{F}_\Sigma(X, \mu_X) \to (X, \mu_X)$ such that $\mathcal{U}_\Sigma(e_0) \circ \eta_{\nabla(X)} = \epsilon_{(X,\mu_X)}$ and $\mathcal{U}_\Sigma(e_H) \circ \eta_{(X,\mu_X)} = \mathrm{id}_{(X,\mu_X)}$. So $\mathcal{U}_\Sigma(e_H)$ is split and, since $\epsilon_{(X,\mu_X)}$ is surjective, point 2 of Proposition 2.4 allows us to conclude that $\mathcal{U}_\Sigma(e_0)$ is an epimorphism. ◀

We want now to translate formulae of our sequent calculus into $\mathcal{X}_0$- and $\mathcal{X}_\mathsf{E}$-equations. To this end, we have to restrict to two classes of theories, which we introduce next.

▶ **Definition 6.9.** *Let $\mathcal{L} = (\Sigma, X)$ be a language, a theory $\Lambda \in \mathsf{Th}(\mathcal{L})$ is said to be:*

- unconditional *([17, App. B.5]) if any sequent in Λ is of the form $\vdash \phi$ for some formula ϕ;*
- of type E *if any sequent in Λ is of the form $\{\mathsf{E}(l_i, x_i)\}_{i\in I} \vdash \phi$ for some formula ϕ, $\{x_i\}_{i\in I} \subset X$ and $\{l_i\}_{i\in I} \subset H$.*

▶ **Lemma 6.10.** *For any signature Σ and $\mathcal{X}_\mathsf{E}$-equation $e : \mathcal{F}_\Sigma(X, \mu_X) \to \mathcal{B}$ there exists a type E theory Λ_e such that, for every Σ-algebra $\mathcal{A}$, $\mathcal{A} \vDash_{\mathcal{X}_1} e$ if and only if $\mathcal{A} \in \mathbf{Mod}(\Lambda_e)$. Moreover $|\Gamma| \leq |\mathsf{supp}(X, \mu_X)|$ for any $\Gamma \vdash \phi \in \Lambda_e$.*

Proof. Let $\mathcal{L}_e$ be (Σ, X). We define $\Gamma_X := \{\mathsf{E}(\mu_X(x), x) \mid x \in \mathsf{supp}(X, \mu_X)\}$ and $\Lambda_e \in \mathsf{Th}(\mathcal{L})$ as $\Lambda_e^1 \cup \Lambda_e^2$ where

$$\Lambda_e^1 := \{\Gamma_X \vdash \mathsf{E}(l, t) \mid l \leq \mu_B(e([t]))\}$$
$$\Lambda_e^2 := \{\Gamma_X \vdash [s] \equiv [t] \mid e([t]) = e([s])\}$$

and (B, μ_B) is $\mathcal{U}_\Sigma(\mathcal{B})$. Let us show the two implications.

⇒ Any $\iota : X \to A$ such that $\mathcal{A} \vDash_\iota \Gamma_X$ defines an arrow $\bar{\iota}(X, \mu_X) \to \mathcal{U}_\Sigma(\mathcal{A})$. By adjointness we have a homomorphism $h_\iota : \mathcal{F}_\Sigma(X, \mu_X) \to \mathcal{A}$ hence, by hypothesis, there exists $q_\iota : \mathcal{B} \to \mathcal{A}$ such that $q_\iota \circ e = h_\iota$. Now, notice that (see Theorem 5.9, and Proposition 5.7(4)) $h_\iota([t]) = t^{\mathcal{A},\iota}$. Take a sequent $\Gamma_X \vdash \psi$ in Λ_e, we have two cases, depending on ψ.

- If $\psi = \mathsf{E}(l, t) \in \Lambda_e^{me}$ we have

$$l \leq \mu_B(e([t])) \leq \mu_A(q_\iota(e([t]))) = \mu_A(h_\iota([t])) = t^{\mathcal{A},\iota}$$

therefore $\mathcal{A} \vDash_\iota \psi$.

- If $\phi = [s] \equiv [t] \in \Lambda_e^{eq}$ then

$$t^{\mathcal{A},\iota} = h_\iota([t]) = q_\iota(e([t])) = q_\iota(e([s])) = h_\iota([t]) = s^{\mathcal{A},\iota}$$

  and even in this case $\mathcal{A} \vDash_\iota \psi$.

$\Leftarrow$ Take $h : \mathcal{F}_\Sigma(X, \mu_X) \to \mathcal{A}$, $\mathcal{U}_\Sigma(h) \circ \eta_{\nabla(X)}$ is an arrow $(X, \mu_X) \to \mathcal{U}_\Sigma(\mathcal{A})$, so forgetting the fuzzy set structure too gives us an assignment $\iota_h : X \to A$ such that $\mathcal{A} \vDash_{\iota_h} \Gamma_X$. As before $h([t]) = t^{\mathcal{A},\iota_h}$ for every $[t] \in \mathcal{F}_\Sigma(X, \mu_X)$. Since $\mathcal{A} \in \mathbf{Mod}(\Lambda_e)$ we have

- $t^{\mathcal{A},\iota_h} = s^{\mathcal{A},\iota_h}$ for all terms t and s such that $e([t]) = e([s])$;
- $l \leq \mu_A(t^{\mathcal{A},\iota_h})$ for all terms t such that $l \leq \mu_B(e([t]))$.

So, the function $q : B \to A$ which sends $b \in B$ to $h([t])$ for some $[t] \in e^{-1}(b)$, provides us with an arrow $\mathcal{U}_\Sigma(\mathcal{B}) \to \mathcal{U}_\Sigma(\mathcal{A})$ such that $q \circ e = h$ and a straightforward computation shows that it is an arrow of Σ-Alg. ◀

▶ **Corollary 6.11.** *For any signature Σ and $\mathcal{X}_0$-equation $e : \mathcal{F}_\Sigma^{\mathbf{Set}}(X) \to \mathcal{B}$ there exists an unconditional theory Λ_e such that, for any Σ-algebra $\mathcal{A}$, $\mathcal{A} \vDash_{\mathcal{X}_0} e$ if and only if $\mathcal{A} \in \mathbf{Mod}(\Lambda_e)$.*

Finally, from the results above we can easily conclude HSP-like results for Σ-Alg.

▶ **Theorem 6.12.** *Let $\mathbf{V}$ be a full subcategory of Σ-Alg, then*

1. *$\mathbf{V}$ is closed under epimorphisms, (small) products and strong monomorphisms if and only if there exists a class of unconditional theories $\{\Lambda_e\}_{e \in \mathbb{E}}$ such that $\mathcal{A} \in \mathbf{V}$ if and only if $\mathcal{A} \in \mathbf{Mod}(\Lambda_e)$ for all $e \in \mathbb{E}$.*
2. *$\mathbf{V}$ is closed under split epimorphisms, (small) products and strong monomorphisms if and only if there exists a class of type E theories $\{\Lambda_e\}_{e \in \mathbb{E}}$ such that $\mathcal{A} \in \mathbf{V}$ if and only if $\mathcal{A} \in \mathbf{Mod}(\Lambda_e)$ for all $e \in \mathbb{E}$.*

Proof. Straightforward in light of Theorem 6.4, Lemma 6.10 and Corollary 6.11. ◀

▶ Remark 6.13. We cannot arrange the collection $\{\Lambda_e\}_{e \in \mathbb{E}}$ into a unique theory since in order to write down all the sequents we need a proper class of variables. A possible way to deal with this issue is to fix two Grothendieck universes ([25]) $\mathbf{U}_1 \subset \mathbf{U}_2$ and allow for a proper class (i.e., an element of $\mathbf{U}_2$) of variables in Definition 3.1. All the proofs of this paper can be repeated verbatim in this context carefully distinguishing between fuzzy *sets* (i.e., those defined on an element of $\mathbf{U}_1$) and fuzzy *classes* (i.e., those defined on an element of $\mathbf{U}_2$). Then the algebras of terms will be a fuzzy class in general but it is possible to show, using the explicit construction, that $\mathsf{T}_\Lambda(X, \mu_X)$ is a fuzzy set if $X \in \mathbf{U}_1$ and so we can retain all the results of Section 5.

The issue mentioned in the previous remark can be avoided if the family $\{\Lambda_e\}_{e \in \mathbb{E}}$ satisfies a boundedness property about the premises of the sequents belonging to each Λ_e.

▶ **Definition 6.14.** *Given a cardinal κ we say that a $\mathcal{X}_\mathsf{E}$-equation $e : \mathcal{F}_\Sigma(X, \mu_X) \to \mathcal{B}$ is* κ-supported *if $|\mathsf{supp}(X, \mu_X)| < \kappa$.*

▶ **Proposition 6.15.** *Let $\mathbf{V} = \mathcal{V}(\mathbb{E})$ be an $\mathcal{X}_\mathsf{E}$-equational defined subcategory of Σ-Alg and suppose every $e \in \mathbb{E}$ is κ-supported, then there exists a theory $\Lambda \in \mathsf{Th}(\mathcal{L})$, where $\mathcal{L} = (\Sigma, \kappa)$, such that $\mathbf{V} = \mathbf{Mod}(\Lambda)$.*

Proof. For any $e : \mathcal{F}_\Sigma(X_e, \mu_{X_e}) \to \mathcal{B}_e$ in $\mathbb{E}$ we can fix an injection $i_e : \mathsf{supp}(X_e, \mu_{X_e}) \to \kappa$ and an extension let $\bar{i}_e : X \to \kappa$ of it, fix also morphisms $\mathbf{I}^e : \mathcal{L}_e \to \mathcal{L}$ given by $(\mathrm{id}_\Sigma, \bar{i}_e)$. Let now $\{\Lambda_e\}_{e \in \mathbb{E}}$ be the collection of theories given by Corollary 6.11 and Theorem 6.12, since each $\Lambda_e \in \mathbf{Form}(\mathcal{L}_e)$ we can define:

$$\Lambda \coloneqq \bigcup_{e \in \mathbb{E}} \mathbf{I}^e_*(\Lambda_e)$$

We have to show that $\mathcal{A} \in \mathbf{V}$ if and only if $\mathcal{A} \in \mathbf{Mod}(\Lambda)$.

$\Rightarrow$ Let $\mathsf{Form}(\mathbf{I}^e)(\Gamma_{X_e}) \vdash \mathsf{Form}(\mathbf{I}^e)(\psi)$ be a sequent in Λ and let $\iota : \kappa \to A$ an assignment such that $\mathcal{A} \vDash_\iota \mathsf{Form}(\mathbf{I}^e)(\Gamma_{X_e})$. By point 3 of Lemma 4.9 this implies $\mathcal{A} \vDash_{\iota \circ \bar{i}_e} \Gamma_{X_e}$, therefore $\mathcal{A} \vDash_{\iota \circ \bar{i}_e} \psi$ and we conclude applying lemma 4.9 again.

$\Leftarrow$ If $\mathcal{U}_\Sigma(\mathcal{A}) = (\emptyset, \mathsf{i}_H)$, ($\mathsf{i}_H$ being the empty map $\emptyset \to H$) then there are no assignment $\kappa \to A$ and so $\mathcal{A}$ is in $\mathbf{Mod}(\Lambda)$. In the other cases let $\Gamma_{X_e} \vdash \psi$ be in Λ_e and $\iota : X_e \to A$ such that $\mathcal{A} \vDash_\iota \Gamma_{X_e}$, since $A \neq \emptyset$ there exists $\hat{\iota} : \kappa \to A$ such that $\hat{\iota} \circ \bar{i}_e = \iota$ as in the previous point Lemma 4.9 implies $\mathcal{A} \vDash_{\hat{\iota}} \mathsf{Form}(\mathbf{I}^e)(\Gamma_{X_e})$, so $\mathcal{A} \vDash_{\hat{\iota}} \mathsf{Form}(\mathbf{I}^e)(\psi)$ and again this is equivalent to $\mathcal{A} \vDash_\iota \psi$. ◀

▶ **Corollary 6.16.** $\mathbf{V}$ *is closed under epimorphisms, (small) products and strong monomorphisms if and only if there exists a language* $\mathcal{L}$ *and an unconditional theory* $\Lambda \in \mathsf{Th}(\mathcal{L})$ *such that* $\mathbf{V} = \mathbf{Mod}(\Lambda)$.

7 Conclusions and future work

In this paper we have introduced a *fuzzy sequent calculus* to capture equational aspects of fuzzy sets. While equalities are captured by usual equations, information contained in the membership function is captured by *membership proposition* of the form $\mathsf{E}(l, t)$, to be interpreted as "the membership degree of t is at least l". We have used a natural concept of *fuzzy algebras* to provide a sound and complete semantics for such calculus, in the sense that a formula is satisfied by all the models of a given theory if and only if it is derivable from it using the rules of our sequent calculus.

As in the classical and quantitative contexts, there is a notion of *free model* of a theory Λ and thus an associated monad T_Λ on the category $\mathbf{Fuz}(H)$ of fuzzy sets over a frame H. However, in general Eilenberg-Moore algebras for such monad are not equivalent to models of Λ, but we have shown that this equivalence holds if Λ is *basic*. In this direction it would be interesting to better understand the categorical status of our approach, investigating possible links between our notion of fuzzy theory and $\mathbf{Fuz}(H)$-Lawvere theories as introduced in full generality by Nishizawa and Power in [20]. A difference between the two approaches is that for us arities are simply finite sets, while following [20] a $\mathbf{Fuz}(H)$-Lawvere theory arities would be given by finite fuzzy sets. A possible underlying concept to both approaches is that of *discrete Lawvere theories* [23, 10].

Finally, using the results provided in [17] we have proved that, given a signature Σ, subcategories of Σ-Alg which are closed under products, strong monomorphisms and epimorphic images correspond precisely with categories of models for *unconditional theories*, i.e. theories axiomatised by sequents without premises. Moreover, using the same results, we have also proved that the categories of models of *theories of type* E, i.e. those whose axioms' premises contain only membership propositions involving variables, are exactly those subcategories closed under products, strong monomorphisms and split epimorphisms.

Our category $\mathbf{Fuz}(H)$ of fuzzy sets has crisp arrows and crisp equality: arrows are ordinary functions between the underlying sets and equalities can be judged to be either true or false. A way to further "fuzzifying" concepts is to use the topos of *H-sets* over the frame H introduced in [9]: this is equivalent to the topos of sheaves over H and contains $\mathbf{Fuz}(H)$ as a (non full) subcategory. By construction, equalities and functions are "fuzzy". It would be interesting to study an application of our approach to this context. A promising feature is that in an H-set the membership degree function is built-in as simply the equality relation, so it would not be necessary to distinguish between equations and membership propositions. Even more generally, we can replace H with an arbitrary quantale $\mathcal{V}$ and consider the category of sets endowed with a "$\mathcal{V}$-valued equivalence relation" [6].

References

1 N. Ajmal. Homomorphism of fuzzy groups, correspondence theorem and fuzzy quotient groups. *Fuzzy sets and systems*, 61(3):329–339, 1994.
2 N. Ajmal and A. S. Prajapati. Fuzzy cosets and fuzzy normal subgroups. *Information sciences*, 64(1-2):17–25, 1992.
3 G. Bacci, R. Mardare, P. Panangaden, and G. Plotkin. An algebraic theory of Markov processes. In *33rd Symposium on Logic in Computer Science (LICS)*, pages 679–688, 2018.
4 G. Bacci, R. Mardare, P. Panangaden, and G. Plotkin. Quantitative equational reasoning. *Foundations of Probabilistic Programming*, page 333, 2020.
5 G. Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 10:433–454, 1935.
6 Filippo Bonchi, Barbara König, and Daniela Petrisan. Up-to techniques for behavioural metrics via fibrations. In *CONCUR*, volume 118 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
7 F. Borceux. *Handbook of Categorical Algebra: Volume 2, Categories and Structures*, volume 2. Cambridge University Press, 1994.
8 Davide Castelnovo and Marino Miculan. Fuzzy algebraic theories. *CoRR*, abs/2110.10970, 2021. `arXiv:2110.10970`.
9 M. P Fourman and D. S. Scott. Sheaves and logic. In *Applications of sheaves*, pages 302–401. Springer, 1979.
10 Martin Hyland and John Power. Discrete lawvere theories and computational effects. *Theoretical Computer Science*, 366(1-2):144–162, 2006.
11 Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electron. Notes Theor. Comput. Sci.*, 172:437–458, 2007. `doi:10.1016/j.entcs.2007.02.019`.
12 Peter T Johnstone. *Stone spaces*, volume 3. Cambridge University Press, 1982.
13 Gregory Maxwell Kelly. A note on relations relative to a factorization system. In *Category Theory*, pages 249–261. Springer, 1991.
14 S. MacLane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.
15 R. Mardare, P. Panangaden, and G. Plotkin. On the axiomatizability of quantitative algebras. In *32nd Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.
16 A. S. Mashour, M. H. Ghanim, and F. I. Sidky. Normal fuzzy subgroups. *Information Sciences*, 20:53–59, 1990.
17 S. Milius and H. Urbat. Equational axiomatization of algebras with structure. In *International Conference on Foundations of Software Science and Computation Structures*, pages 400–417. Springer, 2019.
18 Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. `doi:10.1016/0890-5401(91)90052-4`.
19 J. N. Mordeson, D. S. Malik, and N. Kuroki. *Fuzzy semigroups*, volume 131. Springer, 2012.
20 K. Nishizawa and J. Power. Lawvere theories enriched over a general base. *Journal of Pure and Applied Algebra*, 213(3):377–386, 2009.
21 Gordon D. Plotkin and John Power. Notions of computation determine monads. In *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.
22 Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Appl. Categorical Struct.*, 11(1):69–94, 2003. `doi:10.1023/A:1023064908962`.
23 John Power. Discrete lawvere theories. In *International Conference on Algebra and Coalgebra in Computer Science*, pages 348–363. Springer, 2005.
24 A. Rosenfeld. Fuzzy groups. *Journal of mathematical analysis and applications*, 35(3):512–517, 1971.
25 N. H. Williams. On Grothendieck universes. *Compositio Mathematica*, 21(1):1–3, 1969.
26 O. Wyler. *Lecture notes on topoi and quasitopoi*. World Scientific, 1991.
27 O. Wyler. Fuzzy logic and categories of fuzzy sets. In *Non-Classical Logics and Their Applications to Fuzzy Subsets*, pages 235–268. Springer, 1995.

Realising Intensional S4 and GL Modalities

Liang-Ting Chen
Institute of Information Science, Academia Sinica, Taipei, Taiwan

Hsiang-Shang Ko
Institute of Information Science, Academia Sinica, Taipei, Taiwan

Abstract

There have been investigations into type-theoretic foundations for metaprogramming, notably Davies and Pfenning's (2001) treatment in **S4** modal logic, where code evaluating to values of type A is given the modal type `Code` A ($\Box A$ in the original paper). Recently Kavvos (2017) extended PCF with `Code` A and intensional recursion, understood as the deductive form of the **GL** (Gödel-Löb) axiom in provability logic, but the resulting type system is logically inconsistent. Inspired by staged computation, we observe that a term of type `Code` A is, in general, code to be evaluated in a next stage, whereas **S4** modal type theory is a special case where code can be evaluated in the current stage, and the two types of code should be discriminated. Consequently, we use two separate modalities $\boxtimes$ and $\Box$ to model **S4** and **GL** respectively in a unified categorical framework while retaining logical consistency. Following Kavvos' (2017) novel approach to the semantics of intensionality, we interpret the two modalities in the $\mathscr{P}$-category of assemblies and trackable maps. For the **GL** modality $\Box$ in particular, we use guarded type theory to articulate what it means by a "next" stage and to model intensional recursion by guarded recursion together with Kleene's second recursion theorem. Besides validating the **S4** and **GL** axioms, our model better captures the essence of intensionality by refuting congruence (so that two extensionally equal terms may not be intensionally equal) and internal quoting (both $A \to \Box A$ and $A \to \boxtimes A$). Our results are developed in (guarded) homotopy type theory and formalised in Agda.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases provability, guarded recursion, realisability, modal types, metaprogramming

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.14

Supplementary Material The results were formalised in Agda 2.6.2 in the guarded cubical mode: *Software (Source Code)*: `https://doi.org/10.5281/zenodo.5602771`

Funding This work was partially supported by EPSRC grant number EP/N028139/1 and supported by the Ministry of Science and Technology of Taiwan under grant MOST 109-2222-E-001-002-MY3.

Acknowledgements We are grateful to Alex Kavvos and Tsung-Ju Chiang for insightful discussions. We would also like to thank Jacques Carette, Martín Escardó, Tom de Jong, Churn-Jung Liau, Rasmus Ejlers Møgelberg, Chad Nester, Anton Setzer, Andrea Vezzosi, Ren-June Wang, and Zhixuan Yang for useful exchanges. Finally, we thank the anonymous reviewers for their thoughtful comments.

1 Introduction

Metaprogramming is the activity of writing metaprograms that manipulate program code. Executing a metaprogram can result in another program to be executed, and these successive executions are abstractly referred to as computation *stages*. A particular form of metaprogramming is *staged computation*, where fragments of a program are *internally* marked to be evaluated in multiple stages, so that the program can be partially evaluated to produce more efficient code. The stratification of computation stages forms possible worlds and can be ideally reasoned about by modal logic. Therefore, there have been investigations into type-theoretic foundations for staged computation with modalities [9, 10, 16, 22], which have influenced the design of practical implementations [17, 27, 28] to varying degrees.

Let `Code` A denote the type of code that evaluates to values of type A in a next stage. In Davies and Pfenning's analysis of staged computation [10], `Code` corresponds to the modality $\boxtimes$ in the intuitionistic modal logic **S4**. More specifically, the **4** axiom $\boxtimes A \to \boxtimes\boxtimes A$ corresponds to the use of code in the stage after the next, so code can be shared across all stages, a.k.a. *cross-stage persistence*; the **T** axiom $\boxtimes A \to A$ corresponds to the *evaluation* of code to its value in the same stage. For instance, in λ-calculus, it is well-known [4] that there are terms encoding the Gödel code of a code and evaluating a code respectively.

Recently, Kavvos proposed *intensional recursion* [16] for `Code` to construct a value recursively from its own code (intension). Logically, intensional recursion amounts to the **G**ödel-**L**öb axiom $\Box(\Box A \to A) \to \Box A$ for the modal logic **GL** [5], where $\Box A$ stands for "A is provable". Computationally, the behaviour of the **GL** axiom mirrors Kleene's second recursion theorem. In contrast to general recursion $(A \to A) \to A$, which constructs a value recursively from its value (extension), intensional recursion alone does not lead to logical inconsistency. Kavvos explored the computational capabilities of a variant of PCF extended with `Code` viewed as both **S4** and **GL** modalities. Unfortunately, when `Code` is designed in this way, the type system is inconsistent so that it cannot be treated as a logical foundation.

To make **S4** and **GL** coexist in a single system while maintaining logical consistency, our approach is to keep the two modalities $\boxtimes$ and $\Box$ separate. Intuitively, while both modalities model code and appear similar, there are crucial differences: In general, programs are to be evaluated in a *next* stage, but **S4** is a special case where next stages include the current one, so the result of evaluating a program can be immediately used in the current stage, as witnessed by **T**. On the other hand, a program constructed with **GL** can recursively refer to its own code, which must not be evaluated within the same stage or risk non-termination computationally and inconsistency logically, so stage distinction has to be kept for **GL**.

To illuminate the difference, we present in this paper a denotational semantics of two types of code – a type $\boxtimes A$ of code that can be evaluated in the current stage and a type $\Box A$ of code to be evaluated in a next stage. To distinguish between intensions and extensions, we build upon the previous work using $\mathscr{P}$-categories [8, 15], which have an additional partial equivalence relation on morphisms that models extensional equality, while the underlying equality on morphisms models intensional equality. We revisit elements of realisability and construct a $\mathscr{P}$-category of assemblies on λ-calculus. Roughly speaking, an assembly X on λ-calculus is a set $|X|$ of extensions associated with at least an intension in the form of a λ-term (the existence of such an intension is merely a property that holds for the extension), while trackable maps are pairs of a function and one of its intensions, and can be equipped with both extensional and intensional equalities (taking only the function part or both parts into account). The denotations $\boxtimes X$ and $\Box X$ of the two types of code both consist of pairs (M, x) of an extension x and an associated λ-term M whose associated intensions are λ-terms that are reducible to the Gödel code $\ulcorner M \urcorner$. The choice of using $\ulcorner M \urcorner$ (rather than M as chosen by Kavvos [15]) as the intensions of (M, x) prevents $X \to \boxtimes X$ from having the meaning of generic quoting. The difference between $\boxtimes X$ and $\Box X$ is what extensions are.

- For $\boxtimes X$, the extension part x in (M, x) coincides with the values of the set $|X|$. We can then validate the **S4** axioms, for example $\boxtimes X \to X$ (natural in X) by just projection.
- For $\Box X$, the extension x does not come from $|X|$ but a different set $\triangleright|X|$ whose elements denote values of $|X|$ that are available in a next, or *later*, stage. This set $\triangleright|X|$ can be expressed directly in a *guarded type theory* which features Nakano's later modality $\triangleright$ and *guarded recursion* [21]. By working in guarded type theory, $\Box X \to X$ can no longer be validated by projection (because x from (M, x) is available later rather than now) and is actually not possible; also intensional recursion can be modelled by guarded recursion.

The above constructions do not give rise to $\mathscr{P}$-functors but "exposures" [15] so that congruence (the preservation of extensional equality) is not required by definition and is actually false. Hence, by refuting both congruence and generic quoting, which previous work [12, 15] did not achieve, our denotational semantics is more intensional in the sense that it provides a finer-grained equality which allows us to distinguish computationally equivalent intensions.

We use homotopy type theory (HoTT) [29] as our metalanguage, but this is only for access to a small subset of convenient HoTT features, which in particular does not include univalence. Specifically, our work is built upon the reformulated set theory and logic within HoTT, which enable us to be precise about notions such as sets and propositions, and existence with explicit witnesses versus "mere" existence (for example, within HoTT we can easily distinguish between "pairs of an extension and an intension" and "extensions associated with at least an intension"). Moreover, implementations of HoTT are readily available, so we are able to formalise and verify our constructions, for which HoTT provides essential features that some other type theories lack, notably function extensionality: $(\forall(x : A).\, f\, x =_B g\, x) \to f =_{A\to B} g$. Indeed, our work has been formalised in Agda [11], which implements HoTT and guarded type theory respectively in the forms of cubical type theory [7, 32] and ticked cubical type theory [31] with clock quantification [18]. However, we do not use cubical arguments.

Plan of the paper

After recalling preliminaries on homotopy type theory, untyped λ-calculus, and $\mathscr{P}$-categories in Section 2, we present the $\mathscr{P}$-category of assemblies on λ-calculus and trackable maps developed in HoTT in Section 3 and the denotational semantics of $\boxtimes A$ and $\Box A$ in Sections 4 and 5 respectively, and discuss related work in Section 6 and future work in Section 7.

2 Preliminaries

Most, if not all, of the materials in this section are standard, so we do not go into details.

2.1 Homotopy type theory

In type theory, between every two inhabitants x and y of a type A, there is a type $x =_A y$ of proofs that x and y are (propositionally) equal; given an equality proof $p : x =_A y$, any $z : B(x)$ can be converted to $\mathsf{transport}(p, z) : B(y)$. Between equality proofs $p, q : x =_A y$ there is again a higher-dimensional equality type $p =_{x=_A y} q$, and so on. HoTT identifies this infinite-dimensional structure of equality types as an abstract form of homotopy theory, where types are interpreted as spaces and equality proofs as paths; in particular, equality types are path spaces, and paths between paths are homotopies. We do not make use of the full generality of HoTT but work exclusively with *propositions* and *sets*, whose equality structure degenerates at higher dimensions.

A *proposition* P is a type whose equality types $x =_P y$, for x and $y : P$, are all inhabited – all inhabitants of P are deemed equal, so all we care about P is whether it is inhabited, not its specific inhabitants. Hence, we will simply write $\star$ when referring to an inhabitant of any proposition. If P and Q are propositions, then so is their product $P \times Q$. Similarly, if a family $R(x)$ of types indexed by $x : A$ (where A is any type) are all propositions, then so is the product $\prod_{(x:A)} R(x)$. Logically these give us conjunction and universal quantification; therefore we may write $\forall(x : A).\, R(x)$ in place of $\prod_{(x:A)} R(x)$. Function types $A \to P$ are a special case of $\prod$, so we have implication too. On the other hand, the disjoint sums $P + Q$ and $\sum_{(x:A)} R(x)$ are usually not propositions. This reflects the fact that they are constructive disjunction and existential quantification. In particular, from a proof of $\sum_{(x:A)} R(x)$ we can project a witness $x : A$ and a proof of $R(x)$.

For any type A, the *propositional truncation* of A is a type $\|A\|$ for which there is an introduction rule that wraps any $a : A$ into $|a| : \|A\|$, and a higher one that introduces an inhabitant of $|a| =_{\|A\|} |b|$ for any $a, b : A$, equating all inhabitants of $\|A\|$ and making $\|A\|$ a proposition; its recursion principle maps $\|A\|$ to a proposition P provided that $A \to P$. With truncation, we introduce a propositional version of existential quantification by defining $\exists(x : A).\, R(x) :\equiv \left\|\sum_{(x:A)} R(x)\right\|$, from which we cannot project a witness x and a proof of $R(x)$ in general, but can only derive another proposition P provided that $\left(\sum_{(x:A)} R(x)\right) \to P$. Following the HoTT convention, if an existential quantification in an informal statement is truncated, we will use the words "mere" or "merely" to make it clear.

Sets are types whose equality types are all propositions, so for any two inhabitants of a set, there is at most one way for them to be equal. It is easy to check that the types we work with in this paper are sets: types constructed from sets and the type formers $\times$, $+$, $\to$, $\prod$, and $\sum$ are all sets, and *Hedberg's theorem* is useful for proving that a base type A is a set – if A has decidable equality, that is, $\prod_{(x,y:A)}(x =_A y) + \neg(x =_A y)$, then A is a set (which we will write as $A : \mathsf{Set}$ for short).

2.2 λ-calculus, Gödel encoding, and the second recursion theorem

For λ-calculus we only fix notations. The details are left to, for example, the classic textbook by Barendregt [4]. Terms are defined informally by

$$M :\equiv \mathtt{x} \mid M\, N \mid \lambda\mathtt{x}.\, M$$

where variables $\mathtt{x}$'s are in the typewriter font. Λ denotes the type of terms and Λ_n the type of terms with at most n free variables. In particular, Λ_0 is the type of *closed terms*. Our formalisation uses the de Bruijn representation, so the α-equivalence $=_\alpha$ coincides with the equality type $=_\Lambda$ by construction. For the presentation in this paper, the variable with index i is written $\mathtt{x}_i$, and given $F : \Lambda_{n+1}$ we write $F[M]$ instead of $F[M/\mathtt{x}_0]$ for the substitution for the first free variable $\mathtt{x}_0$. The type $M \longrightarrow\!\!\!\!\!\rightarrow_\beta N$ of reductions from M to N consists of sequences of reduction rules such as $\beta : (\lambda\mathtt{x}.\, M)\, N \longrightarrow_\beta M[N/\mathtt{x}]$; as a special case, the type $M \longrightarrow\!\!\!\!\!\rightarrow_\beta M$ has exactly one inhabitant $\mathsf{refl}_{\twoheadrightarrow} M$, or just $\mathsf{refl}_{\twoheadrightarrow}$, which can be understood as either the empty sequence or (the proof of) the reflexivity of reduction. The types Λ and $M \longrightarrow\!\!\!\!\!\rightarrow_\beta N$ have decidable equality, so they are sets by Hedberg's theorem.

There is a function between λ-terms $\ulcorner\cdot\urcorner : \Lambda \to \Lambda_0$ such that $\ulcorner M \urcorner$ is normal and $M =_\alpha N$ whenever $\ulcorner M \urcorner =_\alpha \ulcorner N \urcorner$. Moreover, there are $\mathtt{ap}, \mathtt{subst} \in \Lambda_2$ and $\mathtt{quote}, \mathtt{eval} \in \Lambda_1$ satisfying

$$\mathtt{ap}[\ulcorner M \urcorner][\ulcorner N \urcorner] \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner M\, N \urcorner \qquad \mathtt{subst}[\ulcorner F \urcorner][\ulcorner N \urcorner] \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner F[N] \urcorner$$
$$\mathtt{quote}[\ulcorner M \urcorner] \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner\ulcorner M \urcorner\urcorner \qquad \mathtt{eval}[\ulcorner M \urcorner] \longrightarrow\!\!\!\!\!\rightarrow_\beta M.$$

This function $\ulcorner\cdot\urcorner$ is called a *Gödel encoding*. Traditionally, a quoted term $\ulcorner M \urcorner$ is called a Gödel *number* since the encoding $\ulcorner\cdot\urcorner$ assigns to every term M a Church numeral $\mathbf{c}_{\#M}$. An encoding needs not be a number at all, however, so we simply call $\ulcorner M \urcorner$ a *code* of M rather than a number. For details on the axiomatic characterisation of encoding, see Polonsky [24].

Note that the term $\mathtt{quote}$ can only compute the code of a term $\ulcorner M \urcorner$ which is already in quoted form. Indeed, no term can compute the code of any arbitrary closed term.

▶ **Proposition 2.1.** *There is no* $\mathtt{Q} : \Lambda_1$ *such that* $\mathtt{Q}[M] \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner M \urcorner$ *for all* $M : \Lambda_0$.

Contrary to the well-known first recursion theorem, Kleene's second recursion theorem works for *code* instead of values and will be used to model the **GL** modality.

▶ **Theorem 2.2** (SRT)**.** *For every* $F : \Lambda_n$ *there exists* $M : \Lambda_n$ *such that* $M \longrightarrow\!\!\!\!\!\rightarrow_\beta F \ulcorner M \urcorner$.

2.3 $\mathscr{P}$-Categories and exposures

Instead of ordinary categories, we work with $\mathscr{P}$-categories pioneered by Čubrić et al. [8], where morphisms are equipped with an additional partial equivalence relation (PER) as another level of equality between morphisms. Kavvos [15] recently advocated its use and introduced a construct called *exposure*, which is similar to a ($\mathscr{P}$-)functor but does not enforce the preservation of PERs of $\mathscr{P}$-categories, to manifest the essence of intensionality.

▶ **Definition 2.3.** A *partial equivalence relation* is a symmetric and transitive relation. A *$\mathscr{P}$-set* $(X, \sim_X)$ is a set X with a PER $\sim_X$. A *$\mathscr{P}$-function* from $(X, \sim_X)$ to $(Y, \sim_Y)$ is a function $f\colon X \to Y$ which respects the relation $\sim$ in the sense that $f\,x \sim_Y f\,y$ whenever $x \sim_X y$. An element $x \in X$ is *well-defined* (with respect to $\sim$) if $x \sim x$.

The identity function id_X is a $\mathscr{P}$-function, and the composite of $\mathscr{P}$-functions is also a $\mathscr{P}$-function. Then we recall the notion of $\mathscr{P}$-categories as follows.

▶ **Definition 2.4** ([8, Definition 2.4])**.** A $\mathscr{P}$-category $\mathbf{C}$ consists of a class of objects, a $\mathscr{P}$-set $(\mathbf{C}(X, Y), \sim)$ for each pair of objects X and Y, and an identity morphism $id_X\colon X \to X$ for each object X satisfying the associativity and identity laws up to $\sim$ in the sense that

- **(i)** $id_X \sim id_X$ always,
- **(ii)** $g \circ f \sim g' \circ f'$ whenever $g \sim g'$ and $f \sim f'$,
- **(iii)** $id \circ f \sim f'$ and $f \circ id \sim f'$ whenever $f \sim f'$,
- **(iv)** $h \circ (g \circ f) \sim (h' \circ g') \circ f'$ whenever $h \sim h'$, $g \sim g'$, and $f \sim f'$.

A $\mathscr{P}$-category has two kinds of equality for morphisms – the underlying equality $=$ and the PER $\sim$, where the former can be used to model the *intensional equality* and the latter the *extensional equality* akin to the structure of multiple judgemental equalities in the modal type theory by Pfenning [23]. Having two different equalities $=$ and $\sim$ reflects the fact that, for example, α-equivalent terms are β-equivalent but not vice versa. For categorical semantics, where terms are interpreted as morphisms, an interpretation into a $\mathscr{P}$-category is able to discriminate these two kinds of equality, enabling us to model intensionality. To emphasise the categorical notions up to the extensional equality $\sim$, the "$\mathscr{P}$-" prefix are added so that we have $\mathscr{P}$-functors, $\mathscr{P}$-initiality, etc.

We recall the notion of exposures, which are like $\mathscr{P}$-functors but are intended to "expose" intensional differences at the extensional level: if an exposure is applied to intensionally different morphisms (which may or may not be identified extensionally), the resulting morphisms may be distinguished extensionally. Consequently, exposures are not required to preserve the extensional equality. Moreover, exposures are only supposed to refine the extensional equality and do not eliminate existing extensional differences, that is, exposures are faithful with respect to $\sim$. Put differently, intensionally equal morphisms, with respect to an exposure, should be extensionally equal. The precise definition is given as follows.

▶ **Definition 2.5.** Given $\mathscr{P}$-categories $\mathbf{C}$ and $\mathbf{D}$, an *exposure* $Q\colon \mathbf{C} \nrightarrow \mathbf{D}$ consists of *(a)* a mapping Q from objects X of $\mathbf{C}$ to objects QX of $\mathbf{D}$ and *(b)* from well-defined morphisms $f\colon X \to Y$ to well-defined morphisms $Qf\colon QX \to QY$ satisfying the following properties:

- **(i)** $Qid_X \sim id_{QX}$,
- **(ii)** $Q(g \circ f) \sim Qg \circ Qf$, and
- **(iii)** $f \sim g$ whenever $Qf \sim Qg$ for any two well-defined morphisms $f, g\colon X \to Y$.

The *identity exposure* $\mathcal{I}$ maps every object or morphism to itself. Composing two exposures in the usual way clearly gives us an exposure.

Similarly, the notion of natural transformations is introduced for exposures, sharing the same idea with ordinary natural transformations but only up to $\sim$.

▶ **Definition 2.6.** Given exposures $P, Q \colon \mathbf{C} \nrightarrow \mathbf{D}$, a *natural transformation of exposures* $t \colon P \to Q$ is a family of well-defined morphisms $t_X \colon PX \to QX$ such that $Qf \circ t_X \sim t_Y \circ Pf$ for every well-defined morphism $f \colon X \to Y$.

An *evaluator* for an endo-exposure Q is a natural transformation from Q to $\mathcal{I}$, modelling the **T** axiom $\boxtimes A \to A$. To model the **S4** modality, we may define *comonadic exposures* introduced by Kavvos [15] as an endo-exposure equipped with an evaluator and a natural transformation $\delta \colon Q \to Q^2$, modelling the **4** axiom $\boxtimes A \to \boxtimes\boxtimes A$, subject to comonad laws. In the presence of intensionality, however, we observe that the naturality is not always appropriate as discussed later in Remark 4.5.

3 $\mathscr{P}$-Category of assemblies on λ-calculus

Assemblies are used to accommodate the information of how extensions are *realised by* intensions. Accordingly an appropriate notion of morphisms between assemblies is introduced to form a $\mathscr{P}$-category, laying the technical foundation for Sections 4 and 5.

3.1 Assembly and trackable map

Traditionally, an assembly on natural numbers is a set $|X|$ with a realisability relation $\Vdash \subseteq \mathbb{N} \times |X|$ such that for every x in $|X|$ there exists some a with $a \Vdash x$, where a is said to *realise* x or a is a *realiser* of x. The modern notion of assemblies [30] is often defined on a partial combinatory algebra $(A, \cdot)$, called *PCA* for short, where $\cdot$ is a partial binary operation. For the sake of formalisation and potential applications in programming language design, we base our definition on λ-calculus subject to α-equivalence, which is more akin to the one based on an ordered PCA [14].

▶ **Definition 3.1.** An *assembly* X on λ-calculus consists of a *carrier* set $|X| : \mathsf{Set}$ and a family $\Vdash_X$ of sets indexed by Λ_0 and $|X|$ as its *realisability relation* such that *(a)* there is merely a realiser $M : \Lambda_0$ of every $x : |X|$, and *(b)* $M \Vdash_X x$ whenever $M \longrightarrow\!\!\!\!\!\rightarrow_\beta N$ and $N \Vdash_X x$. In other words, an assembly is a quadruple $(|X|, \Vdash_X, r_X, t_X)$ of type

$$\mathsf{Asm}_0 :\equiv \sum_{(|X| : \mathsf{Set})} \sum_{(\Vdash_X : \Lambda_0 \to |X| \to \mathsf{Set})} \mathsf{Respects}\,(\Vdash_X, \twoheadrightarrow_\beta) \times \mathsf{RightTotal}(\Vdash_X)$$

where

$$\mathsf{Respects}\,(\Vdash, \twoheadrightarrow_\beta) :\equiv \prod_{(M N : \Lambda_0)} \prod_{(x : |X|)} (M \twoheadrightarrow_\beta N) \to (N \Vdash x) \to (M \Vdash x) \tag{3.1}$$

$$\mathsf{RightTotal}(\Vdash) :\equiv \forall (x : |X|).\, \exists (M : \Lambda_0).\, M \Vdash x \tag{3.2}$$

Our type-theoretic formulation is almost a direct translation from the set-theoretic formulation except that the realisability relation $\Vdash$ is not really a relation but an indexed family of sets. As we would like to account for intensional equality in addition to extensional equality between terms, computationally equivalent terms should not be identified *a priori*. It turns out that formulating the interaction with reduction $\longrightarrow\!\!\!\!\!\rightarrow_\beta$ as (3.1) in line with the definition on an ordered PCA suffices to derive familiar properties.

▶ **Example 3.2.** The type Λ_0 of closed terms with $\longrightarrow\!\!\!\!\!\rightarrow_\beta$ as its realisability relation is an assembly $(\Lambda_0, \longrightarrow\!\!\!\!\!\rightarrow_\beta, r_{\Lambda_0}, t_{\Lambda_0})$ where r_{Λ_0} and t_{Λ_0} are given by the transitivity and the reflexivity of $\longrightarrow\!\!\!\!\!\rightarrow_\beta$. That is, each term M is realised by those reducible to M.

Note that the assembly Λ_0 does *not* yet model code. Indeed, in such case, M should be realised by its *code* $\ulcorner M \urcorner$ instead. This is exactly the point of forthcoming sections.

▶ **Example 3.3.** Every natural number $n : \mathbb{N}$ is realised by terms reducible to its Church numeral c_n. That is, the type $\mathbb{N}$ of natural numbers with $M \Vdash_{\mathbb{N}} n$ whenever $M \longrightarrow\!\!\!\!\!\rightarrow_\beta \mathsf{c}_n$ is an assembly where $r_{\mathbb{N}}$ and $t_{\mathbb{N}}$ are given by the transitivity and the reflexivity of $\longrightarrow\!\!\!\!\!\rightarrow_\beta$.

A morphism between assemblies on a PCA $(A, \cdot)$ is defined as a function f merely *tracked by* some $b \in A$ in the sense that there merely exists some b such that $b \cdot a \Vdash f\,x$ whenever $a \Vdash x$. In this case, b is called the *tracker* of f. It is noted by Kavvos [15] that to bring out intensionality the tracker should be considered as part of the structure instead of a property.

▶ **Definition 3.4.** Given assemblies X and Y, a *trackable map* f from X to Y consists of a function $|f| : |X| \to |Y|$ and a term $F : \Lambda_1$ such that $F[M] \Vdash |f|\,x$ whenever $M \Vdash x$. That is, the type $\mathsf{Asm}_1(X, Y)$ of trackable maps is $\sum_{(|f|:|X|\to|Y|)} \sum_{(F:\Lambda_1)} \mathsf{Tracks}_{X,Y}(F, |f|)$ where

$$\mathsf{Tracks}_{X,Y}(F, |f|) :\equiv \prod_{(M:\Lambda_0)} \prod_{(x:|X|)} (M \Vdash_X x) \to (F[M] \Vdash_Y |f|\,x)\,.$$

A *merely trackable map* is an inhabitant of $\sum_{(|f|:|X|\to|Y|)} \exists (F : \Lambda_1).\, \mathsf{Tracks}_{X,Y}(F, |f|)$.

By definition, a trackable map $f \equiv (f, F, \mathfrak{f})$ consists of not only a function $|f|$ between carriers but also its tracker F and a transformation $\mathfrak{f}$ of realisability.

▶ **Example 3.5.** Every assembly X has an identity map $id_X :\equiv (\mathsf{id}_{|X|}, \mathtt{x}_0, \mathsf{pr}_3)$ where

$$\mathsf{pr}_3 :\equiv \lambda M.\, \lambda x.\, \lambda r.\, r : \prod_{(M:\Lambda_0)} \prod_{(x:|X|)} (M \Vdash_X x) \to (M \Vdash_X x)$$

since $\mathtt{x}_0[M]$ is judgementally equal to M.

Now we proceed with defining the composition of trackable maps. Let $f\colon X \to Y$ and $g\colon Y \to Z$ be trackable maps. Then, the term substitution $(G, F) \mapsto G[F]$ can be thought of as (intensional) function composition, since $G[F[M]] =_{\Lambda_0} G[F][M]$ holds for any term M. Given any $r : M \Vdash_X x$, the inhabitant $\mathfrak{g}(\mathfrak{f}\,r)$ has type $G[F[M]]$ and its transportation along a witness $p : G[F[M]] =_{\Lambda_0} G[F][M]$ has type $G[F][M]$, defining a function $\lambda M\, x\, r.\, \mathsf{transport}(p, \mathfrak{g}(\mathfrak{f}\,r))$. The above discussion amounts to defining a composition operation $(g, f) \mapsto g \circ f$.

3.2 Extensional equality and $\mathscr{P}$-category of assemblies

We define the partial equivalence relation $\sim$, referred to as the extensional equality, on trackable maps by $f_1 \sim f_2$ (f_1 is *extensionally equal* to f_2) if $|f_1| = |f_2|$.

▶ **Proposition 3.6.** *The type* Asm_0 *of assemblies and the family of types* $\mathsf{Asm}_1(X, Y)$ *for any two assemblies* X *and* Y *with the extensional equality form a* $\mathscr{P}$*-category* $\mathsf{Asm}(\Lambda)$.

We now investigate some of its basic properties.

▶ **Example 3.7** ($\mathscr{P}$-Terminal object)**.** The unit $\top :\equiv (\mathbf{1}, \Vdash_\top, r_\top, t_\top)$ is $\mathscr{P}$-terminal where

- **(i)** $\mathbf{1}$ is the unit type,
- **(ii)** $\Vdash_\top$ a relation defined by $M \Vdash_\top \star :\equiv M \longrightarrow\!\!\!\!\!\rightarrow_\beta \mathtt{I}$ where $\mathtt{I} :\equiv \lambda \mathtt{x}.\, \mathtt{x}$,
- **(iii)** $r_\top : (M \longrightarrow\!\!\!\!\!\rightarrow_\beta N) \to (N \longrightarrow\!\!\!\!\!\rightarrow_\beta \mathtt{I}) \to (M \longrightarrow\!\!\!\!\!\rightarrow_\beta \mathtt{I})$ given by the transitivity of $\longrightarrow\!\!\!\!\!\rightarrow_\beta$,
- **(iv)** and $t_\top$ the fact that the only inhabitant $\star : \mathbf{1}$ has a realiser $\mathtt{I}$ (by reflexivity).

The finality follows from function extensionality.

The construction of binary $\mathscr{P}$-products is also typical – the carrier of a product is the cartesian product and a pair (x, y) is realised by M if its Church-encoded projections realise x and y. It follows that $\mathsf{Asm}(\Lambda)$ has finite $\mathscr{P}$-products.

Every inhabitant of an assembly X corresponds to a merely trackable map to X from the terminal object $\top$, which are called *(global) elements* of X, and distinct merely trackable maps can be separated by elements of X. In $\mathsf{Asm}(\Lambda)$, as trackers are part of trackable maps, an element has to be constructed with an intension.

▶ **Lemma 3.8.** *Let X be an assembly. Then the following statements hold:*
1. *Every inhabitant $x : |X|$ corresponds to a merely trackable map from $\top$ to X.*
2. *Every pair of $x : |X|$ and $M : \Lambda_0$ with $r : M \Vdash_X x$ defines a* closed *element of X, i.e. a trackable map $(\lambda\_.\, x, M, \lambda\_.\, \lambda\_.\, \lambda\_.\, r)$ from $\top$ to X.*

As expected, the $\mathscr{P}$-terminal object $\top$ in $\mathsf{Asm}(\Lambda)$ is a $\mathscr{P}$-separator in the sense that for any two trackable maps f_1 and f_2 we have $f_1 \sim f_2$ if $f_1 \circ x \sim f_2 \circ x$ for every element x of X. Even further, we can restrict to closed elements.

▶ **Proposition 3.9.** *Two trackable maps $f_1, f_2 \colon X \to Y$ are extensionally equal if and only if $f_1 \circ x \sim f_2 \circ x$ for every closed element $x \colon \top \to X$. In particular, the $\mathscr{P}$-terminal object $\top$ is a $\mathscr{P}$-separator in $\mathsf{Asm}(\Lambda)$.*

Proof sketch. The proof from left to right is trivial. For the proof from right to left, let f_1 and f_2 be two trackable maps. By function extensionality, to prove that $|f_1| = |f_2|$, we define for any inhabitant $x : |X|$ a closed element $\hat{x}$ of X constructed by Lemma 3.8 using M_x and $r : M_x \Vdash x$ given by the right totality t_X. By the recursion principle of propositional truncation and $|Y|$ being a set, it follows from our assumption that there exists a path $|f_1|\, x = |f_2|\, x$ independent of the choice of M_x and $\mathfrak{M}_x$. ◀

▶ **Example 3.10** (Initial object)**.** The empty assembly $\bot$ is $\mathscr{P}$-initial consisting of the empty type $\mathbf{0}$ and a relation $\Vdash_\bot \colon \Lambda_0 \to \mathbf{0} \to \mathsf{Set}$ given by the elimination rule for the empty type. The other two components $r_\bot$ and $t_\bot$ are trivial.

In addition, one can show that $\bot$ is even a *strict* $\mathscr{P}$-initial object. That is,

▶ **Proposition 3.11.** *Any trackable map from some assembly X to $\bot$ is a $\mathscr{P}$-isomorphism.*

From the strictness of the initial object, no morphism from $\top$ to $\bot$ could exist.

The construction of $\mathscr{P}$-exponential $X \Rightarrow Y$ is a bit laborious and, perhaps surprisingly, $X \Rightarrow Y$ has the type of *merely* trackable maps as its carrier.

▶ **Example 3.12** ($\mathscr{P}$-Exponential)**.** Given assemblies X and Y, define

$$|X \Rightarrow Y| :\equiv \sum_{f:|X|\to|Y|} \exists(F : \Lambda_1).\, \mathsf{Tracks}_{X,Y}(F, f) \equiv \sum_{f:|X|\to|Y|} \left\| \sum_{F:\Lambda_1} \mathsf{Tracks}_{X,Y}(F, f) \right\|$$

with $L \Vdash_{X\Rightarrow Y} (f, \star) :\equiv \prod_{(M:\Lambda_0)} \prod_{(x:|X|)} (M \Vdash_X x) \to (L\, M \Vdash_Y f\, x)$.

It remains to construct $r_{X\Rightarrow Y}$ and $t_{X\Rightarrow Y}$: We know that $L' \longrightarrow\!\!\!\!\!\rightarrow_\beta L$ implies $L'\, M \longrightarrow\!\!\!\!\!\rightarrow_\beta L\, M$, so L' realises $(f, \star)$ whenever L realises $(f, \star)$ and $L' \longrightarrow\!\!\!\!\!\rightarrow_\beta L$ by r_Y. For every $(f, \star) : |X \Rightarrow Y|$, there merely exists a tracker of f, say F. We see that $L :\equiv \lambda\mathtt{x}.\, F$ realises $(f, \star)$, since $(\lambda\mathtt{x}.\, F)\, M \longrightarrow_\beta F[M]$ for any M and $F[M] \Vdash_Y f\, x$ whenever $M \Vdash_X x$. By applying the recursion principle of the truncated type $\left\| \sum_{(F:\Lambda_1)} \mathsf{Tracks}_{X,Y}(F, f) \right\|$ to the second component of $(f, \star)$, there merely exists a realiser of $(f, \star)$ for the right totality.

The *evaluation map* $(X \Rightarrow Y) \times X \xrightarrow{ev_{X,Y}} Y$ natural in X and Y consists of a function $((f, \star), x) \mapsto f\,x$ and its tracker $(\mathtt{proj}_1\,\mathtt{x}_0)\,(\mathtt{proj}_2\,\mathtt{x}_0) : \Lambda_1$ where $\mathtt{x}_0$ is the free variable (thought of as a pair of realisers for a function and its argument) and $\mathtt{proj}_i$ the projection function between λ-terms.

The *curried map* $(f^*, F^*, \mathfrak{f}^*)$ of a trackable function $(f, F, \mathfrak{f})$ from $Z \times X$ to Y consists of a function $f^* :\equiv \lambda z.\,((\lambda x.\,f\,(z, x)), \star_z)$, where by the recursion principle on the mere existence of a realiser $L_z :\equiv t_Z z$ there is merely a tracker $F[\langle L_z, \mathtt{x}_0\rangle]$ of $\lambda x.\,f\,(z, x)$, and a term $F^* :\equiv \lambda \mathtt{x}_0.\,F[\langle \mathtt{x}_1, \mathtt{x}_0\rangle]$ with a witness $\mathfrak{f}^*$ of $\prod_{(L:\Lambda_0)} \prod_{(z:|Z|)} (L \Vdash_Z z) \to (F^*[L] \Vdash_{X \Rightarrow Y} f\,z)$ because of the reduction

$$(\lambda \mathtt{x}_0.\,F[\langle L, \mathtt{x}_0\rangle])\ M \longrightarrow_\beta F[\langle L, M\rangle]$$

and that F is indeed a tracker of $f\colon |Z \times X| \to |Y|$. It is routine to verify remaining details.

▶ **Corollary 3.13.** $\mathsf{Asm}(\Lambda)$ *is a cartesian closed* $\mathscr{P}$*-category with a strict* $\mathscr{P}$*-initial object.*

4 Realisability semantics for the S4 modality

We are now ready to introduce an exposure $\boxtimes\colon \mathsf{Asm}(\Lambda) \nrightarrow \mathsf{Asm}(\Lambda)$ modelling the **S4** modality (validating the **K** axiom $\boxtimes(A \to B) \to \boxtimes A \to \boxtimes B$, the **4** axiom $A \to \boxtimes\boxtimes A$, and the **T** axiom $\boxtimes A \to A$) and show that a generic quoting $X \to \boxtimes X$ cannot exist.

4.1 An exposure for the S4 modality

Given an assembly X, which describes a set of extensions merely realised by some intensions (i.e. terms), we can expose the intensions at the level of extensions by constructing an assembly $\boxtimes X$ where an inhabitant $(M, x, r) : |\boxtimes X|$ is an extension $x : |X|$ and a term $M : \Lambda_0$ that realises x, witnessed by $r : M \Vdash_X x$. This term M becomes the main part of the extension (with respect to $\boxtimes X$) and should be (merely) realised by some intensional representation of M; a natural choice of such representation is $\ulcorner M \urcorner$, or indeed any term β-reducible to $\ulcorner M \urcorner$. In short, the carrier and the realisability relation of $\boxtimes X$ are defined as

$$|\boxtimes X| :\equiv \sum_{(M:\Lambda_0)} \sum_{(x:|X|)} M \Vdash_X x \quad \text{and} \quad (N \Vdash_{\boxtimes X} (M, x, r)) :\equiv N \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner M \urcorner$$

respectively. It turns out that $\boxtimes X :\equiv (|\boxtimes X|, \Vdash_{\boxtimes X}, r_{\boxtimes X}, t_{\boxtimes X})$ is indeed an assembly where $r_{\boxtimes X}$ and $t_{\boxtimes X}$ are the transitivity and the reflexivity of $\longrightarrow\!\!\!\!\!\rightarrow_\beta$.

To make $\boxtimes$ an exposure, we should also define the mapping on morphisms. Consider any trackable map f from X to Y and define $\boxtimes f :\equiv (|f|^{\boxtimes}, F^{\boxtimes}, \mathfrak{f}^{\boxtimes})\colon \boxtimes X \to \boxtimes Y$ as follows. First define a function from $|\boxtimes X|$ to $|\boxtimes Y|$ by

$$|f|^{\boxtimes}\colon (M, x, r) \mapsto (F[M], |f|\,x, \mathfrak{f}\,M\,x\,r).$$

To give a tracker of $\boxtimes f$, recall that there is a term $\mathtt{subst}$ performing term substitution on codes (Section 2.2), and then the term $F^{\boxtimes} :\equiv \mathtt{subst}\ulcorner F \urcorner \mathtt{x}$ tracks $|f|^{\boxtimes}$ because

$$\mathtt{subst}\ulcorner F \urcorner N \longrightarrow\!\!\!\!\!\rightarrow_\beta \mathtt{subst}\ulcorner F \urcorner \ulcorner M \urcorner \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner F[M] \urcorner \Vdash_{\boxtimes Y} |f|^{\boxtimes}(M, x, r)$$

completing the definition of $\mathfrak{f}^{\boxtimes}$. In short, $\boxtimes f :\equiv (|f|^{\boxtimes}, F^{\boxtimes}, \mathfrak{f}^{\boxtimes})$ is a trackable map.

▶ **Definition 4.1.** By $\bigstar\colon \top \to \boxtimes\top$ we denote a closed element of $\boxtimes\top$ given by Lemma 3.8 with $(\mathtt{I}, \star, \mathsf{refl}_{\longrightarrow\!\!\!\!\!\rightarrow}) : |\boxtimes\top|$ and its realiser $\ulcorner \mathtt{I} \urcorner$.

▶ Remark 4.2. Given elements a, b of X with $a \sim b$ but with different trackers, it follows that by definition $\boxtimes a \circ \bigstar \not\sim \boxtimes b \circ \bigstar$ are two extensionally different elements. That is, $\boxtimes$ does not preserve extensional equality.

The assembly $\boxtimes\top$ cannot be $\mathscr{P}$-isomorphic to $\top$, since there are countably many inhabitants of $|\boxtimes\top|$ while there is exactly one inhabitant of $|\top| \equiv \mathbf{1}$. Similarly, there are trackable maps from $\boxtimes(X \times Y)$ to $\boxtimes X \times \boxtimes Y$ and vice versa, but they are not $\mathscr{P}$-isomorphic. It follows that the exposure $\boxtimes$ does not preserve finite $\mathscr{P}$-products.

▶ **Theorem 4.3.** $\boxtimes\colon \mathsf{Asm}(\Lambda) \nrightarrow \mathsf{Asm}(\Lambda)$ *is an exposure of assemblies. Moreover, there is an evaluator ϵ for $\boxtimes$, i.e. a natural transformation ϵ from $\boxtimes$ to $\mathcal{I}$.*

Proof sketch. It is routine to prove the preservation of identities and composition. For example, it follows by definition that $\mathsf{id}^{\boxtimes}_{|X|}(M, x, r) \equiv (\mathtt{x}[M], x, \mathsf{pr}_3\, M\, x\, r) \equiv (M, x, r)$.

Now we show that $\boxtimes$ reflects the extensional equality. Let f and g be trackable maps from X to Y. By assumptions that $\boxtimes f \sim \boxtimes g$ and that there is merely $M : \Lambda_0$ with $r : M \Vdash_X x$, we can apply the recursion principle of propositional truncation to derive

$$\boxtimes f(M, x, r) = (F[M], |f|\, x, \mathfrak{f}\, M\, x\, r) = (G[M], |g|\, x, \mathfrak{g}\, M\, x\, r) = \boxtimes g(M, x, r)$$

since the equality type on $\boxtimes Y$ is a proposition. Therefore, we have $\prod_{(x:|X|)} |f|\, x =_Y |g|\, x$. By function extensionality it then follows that $\big(|f| =_{|X|\to|Y|} |g|\big) \equiv f \sim g$.

As for the evaluator $\epsilon_X\colon \boxtimes X \to X$, recall the term $\mathtt{eval}$ which evaluates a code (Section 2.2). We simply define $|\epsilon_X|$ by $(M, x, r) \mapsto x$. Then, given $N : \Lambda_0$ with $N \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner M \urcorner$, we have $\mathtt{eval}[N] \longrightarrow\!\!\!\!\!\rightarrow_\beta \mathtt{eval}[\ulcorner M \urcorner] \longrightarrow\!\!\!\!\!\rightarrow_\beta M$ where $M \Vdash_X x$ is witnessed by r. That is, $|\epsilon_X|$ is tracked by $\mathtt{eval}$. The naturality of ϵ follows by definition. ◀

Given an element a of X, define its *quotation* as the element $\boxtimes a \circ \bigstar$ of $\boxtimes X$. The choice of $\bigstar$ does not matter if a is closed, since $\boxtimes a \circ \bigstar' \sim \boxtimes a \circ \bigstar$ for any element $\bigstar'$ of $\boxtimes\top$. We say that a trackable map $q\colon X \to \boxtimes X$ *quotes* an element a of X whenever $q \circ a \sim \boxtimes a \circ \bigstar$. The (**4**) axiom can be realised by a family of trackable maps which quote closed elements:

▶ **Proposition 4.4.** *There is a family of functions $|\delta_X|(M, x, r) :\equiv (\ulcorner M \urcorner, (M, x, r), \mathsf{refl}_{\twoheadrightarrow})$ indexed by objects X from $|\boxtimes X|$ to $|\boxtimes\boxtimes X|$ and tracked by* $\mathtt{quote}$*, which quote closed elements of $\boxtimes X$.*

The fact that δ_X quotes closed elements justifies the computational meaning categorically. Yet, δ_X may fail to quote an element a if a is not closed, since in general the intension part of $\boxtimes a$ is applied only verbatim. That is, $|\boxtimes a|(\mathtt{I}, \star, \mathsf{refl}_{\twoheadrightarrow})$ is $(F[\mathtt{I}], (M, x, r), s)$ where $F[\mathtt{I}]$ is not necessarily α-equivalent to $\ulcorner M \urcorner$. The subtlety goes on:

▶ Remark 4.5. One may expect that $(\boxtimes, \epsilon, \delta)$ is comonadic in the sense that δ is a natural transformation up to $\sim$ satisfying comonad laws, but these maps δ_X are *not* natural in X. In detail, for each trackable map $f\colon X \to Y$ the inhabitant

$$\delta_Y(\boxtimes f(M, x, r)) \equiv (\ulcorner F[M] \urcorner, (F[M], f\, x, \mathfrak{f}\, M\, x\, r), \mathsf{refl}_{\twoheadrightarrow}) : \boxtimes\boxtimes Y$$

is not equal to

$$\boxtimes\boxtimes f(\delta_X(M, x, r)) \equiv (\mathsf{subst}\ulcorner F \urcorner\ulcorner M \urcorner, (F[M], f\, x, \mathfrak{f}\, M\, x\, r), \mathsf{subst}_{\twoheadrightarrow}) : \boxtimes\boxtimes Y$$

despite that their extensions are the same, where $\mathsf{subst}_{\twoheadrightarrow}$ is the witness of the reduction sequence $\mathsf{subst}\ulcorner F \urcorner\ulcorner M \urcorner \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner F[M] \urcorner$. Let us define $(M, x, r) \leq (N, y, s)$ if $M \longrightarrow\!\!\!\!\!\rightarrow_\beta N$ and $x = y$ and $f \leq g$ if $|f|\, x \leq |g|\, x$ for all x. Then we only have $|\boxtimes\boxtimes f \circ \delta_X| \leq |\delta_Y \circ \boxtimes f|$. In general, the *lax naturality* appears more appropriate in the presence of intensionality.

The normality condition is realised exactly by $\mathtt{ap}$ without naturality:

▶ **Proposition 4.6.** *There is a family of trackable maps from* $\boxtimes(X \Rightarrow Y)$ *to* $\boxtimes X \Rightarrow \boxtimes Y$ *tracked by* $\lambda \mathtt{x}_0.\, \mathtt{ap}[\mathtt{x}_1]\ \mathtt{x}_0$.

On the other hand, it is impossible for the rule $A \to \boxtimes A$ to compute quotations for arbitrary A, since this is already impossible for the particular case of $\Lambda_0 \to \boxtimes\Lambda_0$ (where Λ_0 was given in Example 3.2).

▶ **Theorem 4.7.** *No trackable map from* Λ_0 *to* $\boxtimes\Lambda_0$ *quotes closed elements of* Λ_0.

Proof. Assume $\eta\colon \Lambda_0 \to \boxtimes\Lambda_0$ with $\eta \circ a \sim \boxtimes a \circ \bigstar$ for any $a\colon \top \to \Lambda_0$ given by Lemma 3.8. Every closed term M defines an element $\widehat{M} :\equiv (\lambda\_.\, M, M, \lambda\_.\, \lambda\_.\, \lambda\_.\, \mathsf{refl}_{\twoheadrightarrow})$ of Λ_0 and thus $\left|\boxtimes\widehat{M}\right|(N, y, s) = (M, M, \mathsf{refl}_{\twoheadrightarrow})$ for any $(N, y, s) : \boxtimes\top$ by definition. By assumption

$$|\eta|\ M \equiv |\eta|\ \left(\left|\widehat{M}\right| \star\right) = \left|\boxtimes\widehat{M}\right|\ (|\bigstar|\ \star) = (M, M, \mathsf{refl}_{\twoheadrightarrow}),$$

so the tracker $\mathtt{Q}$ of η should satisfy $\mathtt{Q}[N] \longrightarrow\!\!\!\!\!\twoheadrightarrow_\beta \ulcorner M \urcorner$ whenever $N \longrightarrow\!\!\!\!\!\twoheadrightarrow_\beta M$. In particular, it follows that $\mathtt{Q}[M] \longrightarrow\!\!\!\!\!\twoheadrightarrow_\beta \ulcorner M \urcorner$. By Proposition 2.1 such $\mathtt{Q}$ cannot exist. ◀

As the choice of $\bigstar$ does not matter for closed elements a, the above theorem shows that even a very limited form of naturality for any two morphisms η_{Λ_0} and $\eta_\top$ satisfying $\eta_{\Lambda_0} \circ a \sim \boxtimes a \circ \eta_\top$ for any closed element a remains impossible. It is unclear how to state "parametricity" for $\boxtimes$ so that any family of morphisms from A to $\boxtimes A$, satisfying a reasonable naturality, can be rejected.

5 Realisability semantics for the GL modality

Kavvos [15] advocated that the provability modality $\Box$ and the **GL** axiom $\Box(\Box A \to A) \to \Box A$ can also be understood as the type of code of type A and as intensional recursion respectively from the computational perspective. Since we already have an exposure $\boxtimes$ modelling typed code, a natural approach is to extend $\boxtimes$ to model the **GL** axiom in addition to **S4**. However, it is known that the **GL** axiom is incompatible with the reflection principle $\Box A \to A$. Indeed, let A be the falsity $\bot$ for both laws. Then, we have $\Box(\Box\bot \to \bot) \to \Box\bot$ and $\Box\bot \to \bot$. By the necessitation rule we can derive $\Box(\Box\bot \to \bot)$ and thus by modus ponens we can derive $\Box\bot$ and finally $\bot$. Therefore, by Proposition 3.11 and Theorem 4.3, we cannot expect the exposure $\boxtimes$ to model the **GL** axiom if we want the type system to be logically consistent.

To untie the knot and retain consistency and the understanding of $\Box A$ as code of type A, we observe that in general $\Box A$ and $\boxtimes A$ are types for different kinds of code and should be kept separate: code constructed with intensional recursion can only be expanded in stages (or otherwise may result in non-termination), whereas code supporting **S4** is only a special case where next stages include the current one, so that $\epsilon_X\colon \boxtimes X \to X$ is allowed. Therefore, separately from $\boxtimes$, we give an exposure $\Box\colon \mathsf{Asm}(\Lambda) \nrightarrow \mathsf{Asm}(\Lambda)$ modelling the **GL** modality in a staged setting (Section 5.2). The construction refutes both $X \to \Box X$ (by the same argument for $\boxtimes$) and the reflection principle $\Box X \to X$. We also derive the **GL** axiom as well as its deductive form. To express staged constructions more conveniently (without stage indexing), we work within guarded type theory.

5.1 Digression: Clocked cubical type theory

We use a particular version of guarded type theory – *clocked cubical type theory* [18], CCTT for short. It extends HoTT with a later modality $\triangleright^\kappa$, parametrised by a "clock" κ, and guarded recursion. Here we only intuitively introduce the constructs and properties of CCTT that are necessary for the informal presentation of the constructions in Section 5.2, but the formal details are all checked in Agda in the guarded cubical mode.

CCTT features a new type $\triangleright(\alpha : \kappa).\, A$ of suspended computations that take in a tick α on a clock κ to produce an inhabitant of A in a next stage. (Clocks will be discussed towards the end and can be ignored for now.) An inhabitant of $\triangleright(\alpha : \kappa).\, A$ therefore resembles a function computationally, and can be introduced as a λ-expression $\lambda(\alpha : \kappa).\, t$, often abbreviated to $\lambda\alpha.\, t$, where $t : A$, or eliminated by application to a tick, denoted by $f[\alpha] : A$ where $f : \triangleright(\alpha : \kappa).\, A$ and $\alpha : \kappa$. For brevity, $\triangleright(\alpha : \kappa).\, A$ is written as $\triangleright^\kappa A$ if α is not referred to in A. For example, the following term implements the normality axiom for $\triangleright^\kappa$:

$$\mathsf{ap}^\kappa :\equiv \lambda f.\, \lambda x.\, \lambda\alpha.\, f[\alpha]\,(x[\alpha]) : \triangleright^\kappa(A \to B) \to \triangleright^\kappa A \to \triangleright^\kappa B.$$

Viewed as a function, ap^κ takes $f : \triangleright^\kappa(A \to B)$ and $x : \triangleright^\kappa A$ as arguments, both of which are values that can be used in a next stage, and should produce a result of type $\triangleright^\kappa B$, that is, a value of type B in a next stage; this result is constructed by first taking in a tick α – after which the rest of the term describes a construction in the next stage – and then applying both f and x to α to produce $f[\alpha] : A \to B$, $x[\alpha] : A$, and eventually $f[\alpha]\,(x[\alpha]) : B$ in the next stage. Another important example is delaying a value to a next stage:

$$\mathsf{next}^\kappa :\equiv \lambda x.\, \lambda\alpha.\, x : A \to \triangleright^\kappa A.$$

In contrast to next^κ, there is no term of type $\triangleright^\kappa A \to A$, matching our intuition about a series of stages happening in order: in the current stage we should not be able to obtain a value that is only available in the next stage. Also there is no term of type $\triangleright^\kappa\triangleright^\kappa A \to \triangleright^\kappa A$ – it might be tempting to write $\lambda x.\, \lambda\alpha.\, x[\alpha][\alpha]$, but the two consecutive applications to α are prohibited in CCTT.

An important primitive is *guarded recursion*, also known as *Löb induction*: every function $f : \triangleright^\kappa A \to A$ has a delayed fixed point $\mathsf{dfix}^\kappa\, f : \triangleright^\kappa A$ with the fixed point equation $(\mathsf{dfix}^\kappa\, f)[\alpha] =_A f\,(\mathsf{dfix}^\kappa\, f)$ where the right-hand side can be seen as a fixed point of f without delay.

We now list some properties needed for Section 5.2. The first one helps to assure that we are still working with propositions and sets even when $\triangleright^\kappa$ is involved.

▶ **Lemma 5.1.** *If $A[\alpha]$ is a proposition/set for arbitrary $\alpha : \kappa$, then so is $\triangleright(\alpha : \kappa).\, A[\alpha]$.*

The later modality distributes over a Σ-type.

▶ **Lemma 5.2.** *Let $B(x)$ be a family of types indexed by $x : A$. Then there are functions from $\triangleright^\kappa \sum_{(x:A)} B(x)$ to $\sum_{(x:\triangleright^\kappa A)} \triangleright(\alpha : \kappa).\, B(x[\alpha])$ and vice versa. In fact, the two types are equivalent.*

Therefore guarded recursion has a specialised form for Σ-types.

▶ **Corollary 5.3.** *Let $B(x)$ be a family of types indexed by $x : A$. Then*

$$\sum_{x:\triangleright A} \triangleright(\alpha : \kappa).\, B(x[\alpha]) \to \sum_{x:A} B(x) \quad \textit{implies} \quad \sum_{x:A} B(x).$$

The final property states that if two values delayed to a next stage are equal, then they are equal in the current stage. It may be tempting to formulate the property as $\mathsf{next}^\kappa\, x =_{\triangleright^\kappa A} \mathsf{next}^\kappa\, y \to x =_A y$, but this is in fact invalid, since (analogously to function extensionality) the antecedent equality is equivalent to $\triangleright^\kappa(x =_A y)$ rather than $x =_A y$ [20]. The correct formulation is the following, where the antecedent includes a *clock quantification* "$\forall\kappa$".

▶ **Lemma 5.4.** *Let x and $y : A$. Then $x =_A y$ if $\forall\kappa.\, \mathsf{next}^\kappa\, x =_{\triangleright^\kappa A} \mathsf{next}^\kappa\, y$.*

This lemma holds because, with clock quantification, it is possible to write Atkey and McBride's [3] operator $\mathsf{force} : (\forall\kappa.\triangleright^\kappa A) \to (\forall\kappa.\, A)$, which can then be applied to the equality $\forall\kappa.\triangleright^\kappa(x =_A y)$ equivalent to the antecedent and yield $\forall\kappa.\, x =_A y$, which is equivalent to $x =_A y$. One way to think about (fully) clock-quantified types is that they are independent of the choice of clocks and can be viewed as the types of pure, completed descriptions of staged computation rather than ongoing computations that are taking effect in stages with respect to a particular clock in scope. We can manipulate such descriptions at will, irrespective of our current timeline – in particular, it is perfectly fine to take a description of a staged computation that produces results from the second stage onwards and make it produce the results right from the first stage instead, which is what force does.

5.2 An exposure for the GL modality

First we adapt the definition of exposures to the setting of CCTT.

▶ **Definition 5.5** (Clocked exposure)**.** Given $\mathscr{P}$-categories $\mathbf{C}$ and $\mathbf{D}$, a *clocked exposure* $Q\colon \mathbf{C} \nrightarrow \mathbf{D}$ consists of *(a)* a mapping Q^κ for each clock κ from objects X of $\mathbf{C}$ to objects $Q^\kappa X$ of $\mathbf{D}$ and *(b)* for each clock κ from well-defined morphisms $f\colon X \to Y$ to well-defined morphisms $Q^\kappa f\colon Q^\kappa X \to Q^\kappa Y$ satisfying following properties

(i) $Q^\kappa id_X \sim id_{QX}$,
(ii) $Q^\kappa(g \circ f) \sim Q^\kappa g \circ Q^\kappa f$, and
(iii) $f \sim g$ whenever $\forall\kappa.\, Q^\kappa f \sim Q^\kappa g$ for any two well-defined morphisms $f, g\colon X \to Y$.

Notably, the faithfulness of a clocked exposure mirrors the form of Lemma 5.4, and is the main reason that we need CCTT.

Now we introduce the clocked exposure $\Box\colon \mathsf{Asm}(\Lambda) \to \mathsf{Asm}(\Lambda)$ modelling **GL**. For an assembly X, the carrier $|\Box^\kappa X|$ and the realisability relation $\Vdash_{\Box^\kappa X}$ are defined as

$$|\Box^\kappa X| :\equiv \sum_{(M:\Lambda_0)} \sum_{(x:\triangleright^\kappa|X|)} \triangleright(\alpha:\kappa).\, M \Vdash_X x[\alpha] \quad \text{and} \quad (N \Vdash_{\Box^\kappa X} (M,x,r)) :\equiv N \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner M \urcorner$$

where $\Vdash_{\Box^\kappa X}$ is defined in the same way as the exposure $\boxtimes$. The main difference between the carriers $|\boxtimes X|$ and $|\Box^\kappa X|$ is that the extension part $|X|$ becomes $\triangleright^\kappa|X|$. That is, the extension x is available in a next stage but not earlier (with respect to the clock κ), but the intension M remains the same type. Similarly, $\Box^\kappa X :\equiv (|\Box^\kappa X|, \Vdash_{\Box^\kappa X}, r_{\Box^\kappa X}, t_{\Box^\kappa X})$ is an assembly where $r_{\Box^\kappa X}$ and $t_{\Box^\kappa X}$ are given by the transitivity and the reflexivity of $\longrightarrow\!\!\!\!\!\rightarrow_\beta$.

For any trackable map f from X to Y, also define $\Box^\kappa f$ in the same way as $\boxtimes f$ except that a later modality is involved:

$$|\Box^\kappa f|(M,x,r) :\equiv (F[M], \triangleright^\kappa|f|\, x, \lambda\alpha.\, \mathfrak{f}\, M\, (x[\alpha])\, (r[\alpha]))$$

where $\triangleright^\kappa|f| : \triangleright^\kappa|X| \to \triangleright^\kappa|Y|$ is given by the functoriality of the later modality $\triangleright^\kappa$. The very same argument for $\boxtimes f$ shows that $\Box^\kappa f$ is indeed a trackable map from $|\Box^\kappa X|$ to $|\Box^\kappa Y|$.

▶ Remark. By Lemma 5.2, the carrier of $\Box^\kappa X$ and the type $\sum_{(M:\Lambda_0)} \triangleright^\kappa \sum_{(x:|X|)} M \Vdash_X x$ are interchangeable. The latter form is more convenient when using guarded recursion.

It is straightforward to show that $\Box$ is a clocked exposure by Lemma 5.4.

▶ **Theorem 5.6.** $\Box\colon \mathsf{Asm}(\Lambda) \looparrowright \mathsf{Asm}(\Lambda)$ *is a clocked exposure.*

▶ **Proposition 5.7.** *There is a family of trackable maps from* $\Box^\kappa(X \Rightarrow Y)$ *to* $\Box^\kappa X \Rightarrow \Box^\kappa Y$ *tracked by* $\lambda \mathtt{x}_0.\, \mathtt{ap}[\mathtt{x}_1]\, \mathtt{x}_0$.

Similar to Theorem 4.7, no morphism $\Lambda_0 \to \Box^\kappa \Lambda_0$ can be quoting.

▶ **Theorem 5.8.** *There is no trackable map from* Λ_0 *to* $\Box^\kappa \Lambda_0$ *which quotes closed elements.*

It is also not possible to have a family of trackable maps ϵ_X from $\Box^\kappa X$ to X natural in X, since the extension of (M, x, r) can only be projected in a time step away from now.

▶ **Theorem 5.9.** *There is no function from* $|\Box^\kappa \bot|$ *to* $|\bot|$. *In particular, there is no natural transformation from* $\Box^\kappa$ *to* $\mathcal{I}$ *for any* κ.

Proof. Assume $\epsilon_\bot\colon |\Box^\kappa \bot| \to |\bot|$ exists. We show that there is $\mathsf{bang}\colon \triangleright \mathbf{0} \to \mathbf{0}$, so by guarded recursion a contradiction $\mathsf{fix}\, \mathsf{bang} : \mathbf{0}$ is derivable. Let x be an inhabitant of $\triangleright \mathbf{0}$. We construct an inhabitant (M, x, r) of $\Box^\kappa \bot$ so that the function $|\epsilon_\bot|$ from $|\Box^\kappa \bot|$ to $|\bot| \equiv \mathbf{0}$ can be applied. Choose an arbitrary closed term M, say $\lambda \mathtt{x}.\, \mathtt{x}$, and apply the recursion principle $\mathsf{rec}_\mathbf{0}$ of the empty type to x in a time step to get $r :\equiv \lambda \alpha.\, \mathsf{rec}_\mathbf{0}\, (M \Vdash_\bot x[\alpha])\, x[\alpha]$, which is an inhabitant of $\triangleright(\alpha : \kappa).\, M \Vdash_\bot x[\alpha]$, so (M, x, r) is of type $|\Box^\kappa \bot|$. ◀

The pay-off for disallowing evaluation is to be able to derive intensional recursion, which is logically the **GL** axiom and its deductive form(s).

▶ **Theorem 5.10** (Intensional recursion)**.** *For every trackable map* $f\colon \Box^\kappa X \to X$, *there are*
1. *an element* $f^\dagger$ *of* $\Box^\kappa X$ *realised by* $\ulcorner \mathsf{fix}\, F \urcorner$ *and*
2. *an element* $f^\ddagger$ *of* X *realised by* $\mathsf{fix}\, F$ *satisfying* $f^\ddagger \sim f \circ \Box^\kappa f^\ddagger \circ \bigstar$

where $\mathsf{fix}\, F : \Lambda_0$ *is a term that can be reduced to* $F[\ulcorner \mathsf{fix}\, F \urcorner]$.

Proof. Let f be a trackable map from $\Box^\kappa X$ to X tracked by $F : \Lambda_1$. Applying Theorem 2.2 to $\lambda \mathtt{x}_0.\, F : \Lambda_0$, we obtain a term $\mathsf{fix}\, F : \Lambda_0$ with $\mathsf{fix}\, F \longrightarrow\!\!\!\!\!\rightarrow_\beta (\lambda \mathtt{x}_0.\, F) \ulcorner \mathsf{fix}\, F \urcorner \longrightarrow_\beta F[\ulcorner \mathsf{fix}\, F \urcorner]$. Now we construct the first element by Löb induction on a Σ-type (Corollary 5.3): assuming

$$x : \triangleright |X| \quad \text{and} \quad r : \triangleright(\alpha : \kappa).\, F[\ulcorner \mathsf{fix}\, F \urcorner] \Vdash x[\alpha]$$

we show an inhabitant of type $|X|$ realised by $\ulcorner \mathsf{fix}\, F \urcorner$ as follows.
1. First, $\triangleright(\alpha : \kappa).\, (\mathsf{fix}\, F \Vdash x[\alpha])$ has an inhabitant, say r', since $\mathsf{fix}\, F$ reduces to $F[\ulcorner \mathsf{fix}\, F \urcorner]$.
2. Then, we derive an inhabitant of type $|X|$ realised by $F[\ulcorner \mathsf{fix}\, F \urcorner]$ as witnessed by

   $$\mathfrak{f}(\mathsf{refl}_{\twoheadrightarrow} \ulcorner \mathsf{fix}\, F \urcorner) : F[\ulcorner \mathsf{fix}\, F \urcorner] \Vdash |f|(\mathsf{fix}\, F, x, r')$$

   since F tracks $|f|$ and the set $\ulcorner \mathsf{fix}\, F \urcorner \Vdash_{\Box^\kappa X} (\mathsf{fix}\, F, x, r')$ is judgementally equal to $\ulcorner \mathsf{fix}\, F \urcorner \longrightarrow\!\!\!\!\!\rightarrow_\beta \ulcorner \mathsf{fix}\, F \urcorner$, which is inhabited by $\mathsf{refl}_{\twoheadrightarrow}$.
3. By Löb induction, $\sum_{(x:\triangleright|X|)} \triangleright(\alpha : \kappa).\, F[\ulcorner \mathsf{fix}\, F \urcorner] \Vdash x[\alpha]$ has an inhabitant (x_0, r_0).
4. By $\mathsf{fix}\, F \longrightarrow\!\!\!\!\!\rightarrow_\beta F[\ulcorner \mathsf{fix}\, F \urcorner]$, we have $(\mathsf{fix}\, F, x_0, r_0') : |\Box^\kappa X|$ where $r_0' : \triangleright(\alpha : \kappa).\, \mathsf{fix}\, F \Vdash x_0[\alpha]$.

Clearly $(\mathsf{fix}\, F, x_0, r_0')$ is realised by $\ulcorner \mathsf{fix}\, F \urcorner$, and by Lemma 3.8 it gives an element $f^\dagger$ of $\Box X$.

To construct the second element $f^\ddagger$, we follow the same steps except the third: we conclude $(x_0, r_0) : \sum_{(x:|X|)} F[\ulcorner \mathsf{fix}\, F \urcorner] \Vdash x[\alpha]$ without any delay and thus $x_0 : |X|$ is realised by $r_0' : \mathsf{fix}\, F \Vdash x_0$. The equation $f^\ddagger \sim f \circ \Box^\kappa(f^\ddagger) \circ \bigstar$ follows from the fixed point equation for guarded recursion. ◀

From the above proof, we can see that the intensional information available in the trackable map $\Box^\kappa X \to X$, i.e. the tracker F, does matter, since it is essential for constructing the fixpoint $\mathsf{fix}\, F$. To internalise the inductive form as the **GL** axiom, we also need the intension:

▶ **Theorem 5.11.** *There is a family of trackable maps from* $\Box^\kappa(\Box^\kappa X \Rightarrow X)$ *to* $\Box^\kappa X$.

The reader may wonder whether the strong Löb axiom, interpreted as a map from $\Box^\kappa X \Rightarrow X$ to X, can also be realised by the SRT in a similar way, but from $\Box^\kappa X \Rightarrow X$, which amounts to a merely trackable map, we do not get the tracker F explicitly needed by the SRT.

6 Related work

Kavvos introduced a comonadic exposure [15, Theorem 11], which we denote by $\boxtimes_K$ here, on $\mathscr{P}$-category of assemblies on a PCA (instead of λ-calculus) to model the intensional **S4** modality. For an assembly X on a PCA $(A, \cdot)$, the assembly $\boxtimes_K X$ is defined by

$$|\boxtimes_K X| :\equiv \{\, (a, x) \mid a \Vdash x \,\} \quad \text{and} \quad b \Vdash_{\boxtimes_K X} (a, x) :\equiv a = b,$$

without the use of Gödel encoding. The morphism mapping is similar to $\boxtimes$. The difference between $\boxtimes_K$ and $\boxtimes$ mainly comes from the chosen notion of realisability and the use of Gödel encoding. First, the exposure $\boxtimes_K$ preserves finite products, while $\boxtimes$ does not. Some β-equivalent intensions have to be identified to satisfy equations of PCA. For example $\star : |\top|$ has only one realiser $\mathtt{I}$, so $|\boxtimes_K \top|$ has only one element $(\star, \mathtt{I})$, too. Second, $\boxtimes_K$ is comonadic by a similar reason, while our δ is not even natural. Third, $\boxtimes_K$ is idempotent, i.e. δ_K are isomorphisms, which is impossible for $\boxtimes$ because of Gödel encoding. Finally, a generic quoting for $\boxtimes_K$ *can* be defined:

▶ **Observation 6.1.** *For the one-element PCA, there exists a natural transformation from the identity exposure to* $\boxtimes_K$.

Further, assuming the axiom of choice, there is (merely) a function q from $|X|$ to $|\boxtimes_K X|$ because of the right totality of $\Vdash$. It is likely that q could be realised in the sense of Krivine's classical realisability [19], which would establish a non-trivial example of generic quoting for $\boxtimes_K$ or alike. On the other hand, using λ-terms subject to α-equivalence as realisers and $\ulcorner M \urcorner$ as realisers for (M, x, r) allows the exposure $\boxtimes$ to distinguish β-equivalent intensions, so $\boxtimes$ – being more intensional than $\boxtimes_K$ – lacks well-behaved extensional properties, as expected.

As a reviewer pointed out, Remark 4.5 is reminiscent of *categorical simulation* studied by Cockett and Hofstra [6].

Artemov and Beklemishev [2] pointed out that Gödel attempted to use classical **S4** modal logic to capture provability for Peano Arithmetic (**PA**) but realised that $\mathsf{Prov}(A) :\equiv \exists x.\, \mathsf{Proof}(x, A)$ cannot be **S4**. Löb found the well-known **GL** axiom and Solovay showed that **GL** is complete with respect to **PA**. On the other hand, Artemov [1] proposed *logic of proofs* extending **S4** and argued that **S4** is for *explicit proofs*. Goris [13] discussed two modalities **GL** and **S4** over classical logic by presenting a bi-modal logic and provides a Kripke semantics for both. We took Goris' notation for the **S4** modality $\boxtimes$. Our work is partly inspired by Shamkanov's [25, 26] provability semantics for **GL** using circular proofs.

7 Conclusion

In this paper we follow the principle of modality-as-intension [10] and the $\mathscr{P}$-categorical semantics [15] to manifest the concepts of denotations, extensions, and intensions. We have studied the $\mathscr{P}$-category of assemblies on λ-calculus developed and formalised in HoTT as a semantic foundation for intensionality, on which we modelled **S4** and **GL** modalities.

Notably, our denotational semantics of the **S4** modality $\boxtimes$ is more intensional than that of Kavvos' $\boxtimes_K$. For the **GL** modality $\Box$, we have given the *first* denotational semantics, which is shown (Theorem 5.10 and Theorem 5.11) to satisfy the Gödel-Löb axiom and its deductive form – the intensional recursion – using guarded type theory.

As future work, an important issue we have not discussed yet is the connection between $\boxtimes$ and $\Box$ – for example, it is easy to construct a family of trackable maps from $\boxtimes X$ to $\Box^\kappa X$ natural in X by deferring the extension part of (M, x) to a later stage, and Goris' bi-modal logic [13] suggests that there are more rules to discover. In the long term, based on the $\mathscr{P}$-category of assemblies and the denotational semantics of modalities, we intend to design a type theory with two modes of `Code` and prove its meta-properties such as consistency, confluence, and decidability of type checking by interpreting judgements into $\mathsf{Asm}(\Lambda)$.

References

1. Sergei N. Artemov. Explicit provability and constructive semantics. *Bulletin of Symbolic Logic*, 7(1):1–36, 2001. doi:10.2307/2687821.
2. Sergei N. Artemov and Lev D. Beklemishev. Provability logic. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition*, volume 13 of *Handbook of Philosophical Logic*, pages 189–360. Springer, Dordrecht, 2005. doi:10.1007/1-4020-3521-7_3.
3. Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, page 197, New York, New York, USA, 2013. ACM Press. doi:10.1145/2500365.2500597.
4. Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic*. North Holland, 1984.
5. George S. Boolos. *The Logic of Provability*. Cambridge University Press, 1994. doi:10.1017/cbo9780511625183.
6. J.R.B. Cockett and Pieter J.W. Hofstra. Categorical simulations. *Journal of Pure and Applied Algebra*, 214(10):1835–1853, 2010. doi:10.1016/j.jpaa.2009.12.028.
7. Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–34, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2015.5.
8. Djordje Čubrić, Peter Dybjer, and Philip J. Scott. Normalization and the Yoneda embedding. *Mathematical Structures in Computer Science*, 8(2):153–192, 1998. doi:10.1017/S0960129597002508.
9. Rowan Davies. A temporal logic approach to binding-time analysis. *Journal of the ACM*, 64(1):1–45, 2017. doi:10.1145/3011069.
10. Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001. doi:10.1145/382780.382785.
11. Agda development team. Agda 2.6.2 documentation. Accessed: 2021-06-20. URL: https://agda.readthedocs.io/en/v2.6.2/.
12. Murdoch J. Gabbay and Aleksandar Nanevski. Denotation of contextual modal type theory (CMTT): Syntax and meta-programming. *Journal of Applied Logic*, 11(1):1–29, 2013. doi:10.1016/j.jal.2012.07.002.
13. Evan Goris. A modal provability logic of explicit and implicit proofs. *Annals of Pure and Applied Logic*, 161(3):388–403, 2009. doi:10.1016/j.apal.2009.07.020.
14. Pieter Hofstra and Jaap van Oosten. Ordered partial combinatory algebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 134(3):445–463, 2003. doi:10.1017/S0305004102006424.

15 G. A. Kavvos. On the semantics of intensionality. In Javier Esparza and Andrzej S. Murawski, editors, *Proceedings of the 20th International Conference the Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 10203 of *Lecture Notes in Computer Science*, pages 550–566. Springer, Berlin, Heidelberg, 2017. doi:10.1007/978-3-662-54458-7_32.

16 G. A. Kavvos. Intensionality, intensional recursion, and the Gödel-Löb axiom. *Journal of Applied Logics - The IfCoLog Journal of Logics and their Applications*, 8(8):2287–2311, 2021. Special Issue: Intuitionistic Modal Logic and Applications. URL: http://collegepublications.co.uk/ifcolog/?00050.

17 Oleg Kiselyov. The design and implementation of BER MetaOCaml. In Michael Codish and Eijiro Sumii, editors, *Proceedings of the 12th International Symposium on Functional and Logic Programming (FLOPS)*, volume 8475 of *Lecture Notes in Computer Science*, pages 86–102. Springer, Cham, 2014. doi:10.1007/978-3-319-07151-0_6.

18 Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, and Andrea Vezzosi. Greatest HITs: Higher inductive types in coinductives via induction under clocks. *ArXiv preprint*, 2021. arXiv:2102.01969.

19 Jean-Louis Krivine. A program for the full axiom of choice. *Logical Methods in Computer Science*, 17(3):21:1–22, 2021. doi:10.46298/lmcs-17(3:21)2021.

20 Rasmus Ejlers Møgelberg and Niccolò Veltri. Bisimulation as path type for guarded recursive types. *Proceedings of the ACM on Programming Languages*, 3(POPL):4:1–29, 2019. doi:10.1145/3290317.

21 Hiroshi Nakano. A modality for recursion. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2000. doi:10.1109/LICS.2000.855774.

22 Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008. doi:10.1145/1352582.1352591.

23 Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 221–230. IEEE Computer Society, 2002. doi:10.1109/LICS.2001.932499.

24 Andrew Polonsky. Axiomatizing the quote. In Marc Bezem, editor, *Computer Science Logic (CSL)—25th International Workshop/20th Annual Conference of the EACSL*, volume 12 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 458–469, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2011.458.

25 Daniyar S. Shamkanov. Circular proofs for the Gödel-Löb provability logic. *Mathematical Notes*, 96(3–4):575–585, 2014. doi:10.1134/S0001434614090326.

26 Daniyar S. Shamkanov. A realization theorem for the Gödel-Löb provability logic. *Sbornik: Mathematics*, 207(9):1344–1360, 2016. doi:10.1070/SM8667.

27 Tim Sheard and Simon Peyton Jones. Template meta-programming for Haskell. In *Proceedings of the 2002 ACM SIGPLAN Workshop on Haskell*, pages 1–16, New York, New York, USA, 2002. ACM Press. doi:10.1145/581690.581691.

28 Walid Taha and Tim Sheard. MetaML and multi-stage programming with explicit annotations. *Theoretical Computer Science*, 248(1–2):211–242, 2000. doi:10.1016/S0304-3975(00)00053-0.

29 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

30 Jaap van Oosten. *Realizability: An Introduction to its Categorical Side*. Studies in Logic and the Foundations of Mathematics. Elsevier, 2008. doi:10.1016/S0049-237X(08)80001-8.

31 Niccolò Veltri and Andrea Vezzosi. Formalizing π-calculus in guarded cubical Agda. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pages 270–283, New York, NY, USA, 2020. ACM. doi:10.1145/3372885.3373814.

32 Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: A dependently typed programming language with univalence and higher inductive types. *Journal of Functional Programming*, 31:e8:1–47, 2021. doi:10.1017/s0956796821000034.

Localisable Monads

Carmen Constantin
University of Edinburgh, UK

Nuiok Dicaire
University of Edinburgh, UK

Chris Heunen
University of Edinburgh, UK

Abstract

Monads govern computational side-effects in programming semantics. A collection of monads can be combined together in a local-to-global way to handle several instances of such effects. Indexed monads and graded monads do this in a modular way. Here, instead, we start with a single monad and equip it with a fine-grained structure by using techniques from tensor topology. This provides an intrinsic theory of local computational effects without needing to know how constituent effects interact beforehand.

Specifically, any monoidal category decomposes as a sheaf of local categories over a base space. We identify a notion of localisable monads which characterises when a monad decomposes as a sheaf of monads. Equivalently, localisable monads are formal monads in an appropriate presheaf 2-category, whose algebras we characterise. Three extended examples demonstrate how localisable monads can interpret the base space as locations in a computer memory, as sites in a network of interacting agents acting concurrently, and as time in stochastic processes.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases Monad, Monoidal category, Presheaf, Central idempotent, Graded monad, Indexed monad, Formal monad, Strong monad, Commutative monad

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.15

Related Version An extended version of this paper with deferred proofs is available.
Extended Version: `https://arxiv.org/abs/2108.01756`

Funding *Carmen Constantin*: Supported by EPSRC Fellowship EP/R044759/1.
Nuiok Dicaire: Supported by a PGSD Scholarship from the Natural Sciences and Engineering Research Council of Canada (NSERC) and an Enlightenment Scholarship.
Chris Heunen: Supported by EPSRC Fellowship EP/R044759/1.

Acknowledgements We thank Rui Soares Barbosa, Robert Furber, and Nesta van der Schaaf for useful discussions. We also thank the reviewers for their helpful feedback.

1 Introduction

The computation of some desired value may influence parts of the environment in which the computation occurs that are separate from the value itself. Rather than being accidental byproducts, several modern programming platforms harness such *computational side-effects* to structure computations in a modular way [31, 30]. The most well-known use is via *monads* [28, 29], which let one analyse a computational effect apart from the rest of the computation.

A computation may use more than one effect. The corresponding monads can then be combined using *distributive laws* into a single monad [17, 3, 38]. This combination uses the fact that the base category on which the monad lives is highly structured; usually it is a cartesian category of presheaves. It may involve other formalisms such as Lawvere theories [33, 32], but we focus on monads here. An especially interesting case is when many

instances of effects of the same kind are in play [34]. A related use of monads is to have several layers of granularity to an effect. Indexed monads and *graded monads* then model for example different levels of access to a computational effect [12, 26]. Here, as in the previous case, this is usually conceived of in a local-to-global fashion, where one specifies the behaviour at each level and then adds interplay between the levels.

In this article we take a dual approach and start with a single monad on a category with some structure. We then ask when and how that monad is the combination of constituent monads. This work is a first step towards an *intrinsic* theory of computational effects, one that doesn't need to specify in detail how constituent effects have to interact in advance. In particular, we do not postulate that the base category consists of presheaves, which is a consequence rather than an assumption.

To do so, we follow the programme of *tensor topology*, by observing that any monoidal category comes equipped with a notion of base space over which the category decomposes [10, 2, 9, 14]. This "spatial" aspect can be cleanly separated: any monoidal category embeds into a category of global sections of a sheaf of so-called local monoidal categories (see Theorems 10 and 11 below). This is recalled in Section 2.

Our main question is when and how a monad on a monoidal category respects this decomposition in the sense that it corresponds to a sheaf of monads on the local categories. The answer is a *localisable monad*, discussed in Section 3. To connect back to the local-to-global approach, we then characterise such monads as *formal monads* [35] in a (pre)sheaf category in Section 4. This opens a way to analyse the (Kleisli) algebras for localisable monads, which we do in Section 6. The breadth of this approach is demonstrated in Section 5, where we work out three extended examples. They show a range of how localisable monads may interpret the base space: as locations in a computer memory governed by a *local state* monad; as sites in a network of interacting agents governed by a monad inspired by the *pi calculus*; and as moments in time governed by a monad of *stochastic processes*. Section 7 concludes. Some proofs can be found in the extended version of this paper [5].

2 Tensor topology

This section summarises necessary notions from tensor topology. We have to be brief, and for more details we refer the reader to [10, 2, 14, 9]. To save space we will not use the graphical calculus for monoidal categories [16], but will not be careful in denoting coherence isomorphisms in this section. The following notions and results hold for arbitrary monoidal categories, but for simplicity we deal here with the symmetric monoidal case only.

▶ **Definition 1.** *A* central idempotent *in a symmetric monoidal category is a morphism* $u\colon U \to I$ *such that* $\rho_U \circ (U \otimes u) = \lambda_U \circ (u \otimes U)\colon U \otimes U \to U$ *and this map is invertible. We identify two central idempotents* $u\colon U \to I$ *and* $v\colon V \to I$ *when there is an isomorphism* $m\colon U \to V$ *satisfying* $u = v \circ m$*. Write* $\mathrm{ZI}(\mathbf{C})$ *for the collection of central idempotents of* $\mathbf{C}$*.*

A central idempotent $u\colon U \to I$ is completely determined by its domain U. The central idempotents always form a (meet-)semilattice. The order is defined by $u \leq v$ if and only if $u = v \circ m$ for some morphism $m\colon U \to V$. The meet is given $u \wedge v = \lambda_I \circ (u \otimes v)\colon U \otimes V \to I$. The largest central idempotent is the identity $1\colon I \to I$.

▶ **Example 2.** Consider a (meet-)semilattice $(L, \wedge, 1)$ as a symmetric monoidal category $\mathbf{C}$: objects of $\mathbf{C}$ are elements of L, there is a morphism $u \to v$ if and only if $u \leq v$, and $u \otimes v = u \wedge v$. Then $\mathrm{ZI}(\mathbf{C}) \simeq L$. In fact, ZI is a functor that is right adjoint to the inclusion of the category of semilattices into the category of symmetric monoidal categories.

▶ **Example 3.** If $\mathbf{C}$ is cartesian – that is, tensor products are in fact categorical products – then central idempotents are exactly subterminal objects: objects U whose unique morphism $!\colon U \to 1$ to the terminal object is monic. In particular, if X is any topological space, the category of sheaves over X has as central idempotent semilattice the collection of open sets $U \subseteq X$ under intersection.

▶ **Example 4.** If X is a locally compact Hausdorff topological space, the category of Hilbert modules over $C_0(X)$ is symmetric monoidal. It is equivalent to the category of fields of Hilbert spaces over X, and its central idempotents correspond to open subsets $U \subseteq X$.

Because of the previous examples, we can think of central idempotents as open subsets of a hidden base space that any symmetric monoidal category comes equipped with. Tensor topology develops general accompanying notions of locality, restriction, and support. For example, we can restrict attention to the "part of the category that lives over an open set", as follows.

▶ **Proposition 5.** *For every central idempotent u in a symmetric monoidal category $\mathbf{C}$, there is a symmetric monoidal category $\mathbf{C}\|_u$ where:*

- *objects are as in $\mathbf{C}$;*
- *morphisms $A \to B$ are morphisms $A \otimes U \to B$ in $\mathbf{C}$;*
- *composition of $f\colon A\otimes U \to B$ and $g\colon B\otimes U \to C$ is $g\circ(f\otimes U)\circ(A\otimes U\otimes u)^{-1}\colon A\otimes U \to C$;*
- *the identity on A is given by $A \otimes u$;*
- *tensor product of objects is as in $\mathbf{C}$;*
- *tensor product of morphisms $f\colon A \otimes U \to B$ and $f'\colon A' \otimes U \to B'$ is $(f \otimes f') \circ (A \otimes \sigma_{A',U} \otimes U) \circ (A \otimes A' \otimes U \otimes u)^{-1}\colon A \otimes A' \otimes U \to B \otimes B'$.* ◀

▶ Remark 6. In $\mathbf{C}\|_u$, any object A is isomorphic to $A \otimes U$: the isomorphism and its inverse are given by the identity $A \otimes U \to A \otimes U$ in $\mathbf{C}$ and $A \otimes u \otimes u\colon A \otimes U \otimes U \to A$.

▶ **Example 7.** In the category $\mathbf{C}$ of sheaves over a topological space X, central idempotents u correspond to open subsets $U \subseteq X$ as in Example 3. The category $\mathbf{C}\|_u$ is then equivalent to the category of sheaves over U.

The intuition of a category $\mathbf{C}$ "living over" open subsets is further strengthened by the following lemma, that says we can pass between the part of a category living over a larger open subset and the part living over a smaller open subset.

▶ **Lemma 8.** *If $u \leq v$ are central idempotents in $\mathbf{C}$, with $u = v \circ m$, there is an adjunction:*

$$\mathbf{C}\|_u \underset{\mathbf{C}\|_{u\leq v}}{\overset{\mathbf{C}\|^{u\leq v}}{\rightleftarrows}} \mathbf{C}\|_v \qquad (\perp)$$

The functor $\mathbf{C}\|_{u\leq v}$ is given by $A \mapsto A$ and $f \mapsto f \circ (A \otimes m)$ and is strict monoidal. The functor $\mathbf{C}\|^{u\leq v}$ is given by $A \mapsto A \otimes U$ and $f \mapsto (f \otimes U) \circ (A \otimes u \otimes U)^{-1} \circ (A \otimes U \otimes v)$ and is oplax monoidal. The unit of the adjunction is an isomorphism.

Proof. See [2, Lemmas 5.4 and 5.5]. ◀

To make the intuition built up so far completely rigorous, we now summarise a series of results saying that any symmetric monoidal category may be regarded as a sheaf of monoidal categories over a base topological space. To state them, we need to introduce mild conditions on the central idempotents being respected by tensor products.

▶ **Definition 9.** *A symmetric monoidal category* $\mathbf{C}$ *is called* stiff *when the diagram on the left below is a pullback for any object* A *and central idempotents* u *and* v*.*

$$\begin{array}{ccc} A \otimes U \otimes V & \longrightarrow & A \otimes V \\ \downarrow & & \downarrow A \otimes v \\ A \otimes U & \xrightarrow{A \otimes u} & A \end{array} \qquad \begin{array}{ccc} A \otimes U \otimes V & \rightarrowtail & A \otimes V \\ \downarrow & & \downarrow \\ A \otimes U & \rightarrowtail & A \otimes (U \vee V) \end{array}$$

We say $\mathbf{C}$ *has* finite universal joins *of central idempotents when it has an initial object* 0 *satisfying* $A \otimes 0 \simeq 0$ *for all objects* A*, and* $\mathrm{ZI}(\mathbf{C})$ *has binary joins such that the diagram on the right above is a pullback and a pushout for all objects* A *and central idempotents* u *and* v*.*

The following theorem says that any stiff monoidal category can be freely completed with universal finite joins of central idempotents [2, Theorem 12.8]. Finally, Theorem 11 [2, Theorem 8.6] says that any symmetric monoidal category $\mathbf{C}$ with universal finite joins has a particularly nice form. It considers the semilattice of central idempotents $\mathrm{ZI}(\mathbf{C})$ as the basic opens of a topological space X by taking its Zariski spectrum [2, Section 4].

▶ **Theorem 10.** *Any stiff symmetric monoidal category allows a strict monoidal full embedding into a symmetric monoidal category with finite universal joins of central idempotents.*

▶ **Theorem 11.** *Any symmetric monoidal category* $\mathbf{C}$ *with universal finite joins of central idempotents is monoidally equivalent to a category of global sections of a sheaf* $u \mapsto \mathbf{C}\|_u$ *of local monoidal categories over* $\mathrm{ZI}(\mathbf{C})$*.*

Here, a monoidal category $\mathbf{C}$ is called local when $u \vee v = 1$ implies $u = 1$ or $v = 1$ in $\mathrm{ZI}(\mathbf{C})$. When $\mathrm{ZI}(\mathbf{C})$ is the opens of a topological space, that means there is a single focal point that all nets in the topological space converge to – intuitively, $\mathbf{C}$ is local when it has no nontrivial central idempotents. Being a sheaf of local monoidal categories means that the stalks $\mathbf{C}\|_x = \mathrm{colim}_{x \in u} \mathbf{C}\|_u$ over points $x \in X$ are local monoidal categories.

It follows that any stiff symmetric monoidal category embeds into such a category of global sections. This makes precise the intuition that a symmetric monoidal category continuously varies over its base space of central idempotents.

3 Localisable monads

The previous section showed how any symmetric monoidal category $\mathbf{C}$ may be regarded as a sheaf $\mathbf{C}\|_u$ of local ones. In this section, we work out when a monad on $\mathbf{C}$ corresponds to a sheaf of monads on $\mathbf{C}\|_u$. The crucial definition is as follows.

▶ **Definition 12.** *A monad* T *on a monoidal category* $\mathbf{C}$ *is called* localisable *when there are morphisms* $\mathrm{st}_{A,U} \colon T(A) \otimes U \to T(A \otimes U)$ *for each object* A *and central idempotent* $u \colon U \to I$ *satisfying:*

$$T(\rho_A) \circ \mathrm{st}_{A,I} = \rho_{T(A)} \tag{1}$$

$$T(\alpha_{A,U,V}) \circ \mathrm{st}_{A,U \otimes V} = \mathrm{st}_{A \otimes U,V} \circ (\mathrm{st}_{A,U} \otimes V) \circ \alpha_{TA,U,V} \tag{2}$$

$$\eta_{A \otimes U} = \mathrm{st}_{A,U} \circ (\eta_A \otimes U) \tag{3}$$

$$\mu_{A \otimes U} \circ T(\mathrm{st}_{A,U}) \circ \mathrm{st}_{T(A),U} = \mathrm{st}_{A,U} \circ (\mu_A \otimes U) \tag{4}$$

$$\mathrm{st}_{A,V} \circ (T(A) \otimes m) = T(A \otimes m) \circ \mathrm{st}_{A,U} \tag{5}$$

$$\mathrm{st}_{B,U} \circ \big(T(f) \otimes U\big) = T(f \otimes U) \circ \mathrm{st}_{A,U} \tag{6}$$

for any morphism $f \colon A \to B$ *and central idempotents* $u \colon U \to I$ *and* $v \colon V \to I$*, and where* $m \colon U \to V$ *in* (5) *satisfies* $u = v \circ m$*.*

▶ **Example 13.** Consider a semilattice $(L, \wedge, 1)$ as a symmetric monoidal category $\mathbf{C}$ as in Example 2. A monad on $\mathbf{C}$ then is exactly a closure operator on L, that is, a function $\overline{(-)}\colon L \to L$ satisfying $u \leq \overline{u} = \overline{\overline{u}}$ and $u \leq v \implies \overline{u} \leq \overline{v}$. This monad is localisable if and only if $\overline{u} \wedge v \leq \overline{u \wedge v}$ for all $u, v \in L$. This is for example the case when L is the powerset of a set X, and $\overline{U}$ is the closure of $U \subseteq X$ in a fixed topology on X.

▶ **Example 14.** Strong monads [21, 18] are localisable: axioms (1)–(4) are a special case of the axioms for a strong monad; and axioms (5)–(6) follow from naturality of strength. Hence a monad T on a symmetric monoidal closed category is localisable if $T(U \multimap A) \simeq T(U) \multimap T(A)$, namely with $\mathrm{st}_{A,U}$ as follows (where coev denotes the curry of the identity on $A \otimes U$)

$$\begin{array}{ccc} T(A) \otimes U & \xrightarrow{T(\mathrm{coev})\otimes\eta} & T\big(U \multimap (A \otimes U)\big) \otimes T(U) \\ & & \| \\ T(A \otimes U) & \xleftarrow[\mathrm{ev}]{} & \big(T(U) \multimap T(A \otimes U)\big) \otimes T(U) \end{array}$$

▶ **Example 15.** It follows from Example 14 and [20] that a monad T on a cartesian closed category is localisable as soon as $T(A \times B) \simeq T(A) \times T(B)$. In particular, this applies for any monad on the category of sheaves over a topological space X as in Example 3.

We will work out more examples in Section 5 below. Next we consider the main consequence of a monad on $\mathbf{C}$ being localisable: it restricts to the categories $\mathbf{C}\|_u$.

▶ **Proposition 16.** *If T is a localisable monad on $\mathbf{C}$ and u a central idempotent, the following defines a monad $T\|_u$ on $\mathbf{C}\|_u$:*

$$\begin{aligned} T\|_u(A) &= T(A) & (\eta\|_u)_A &= \eta_A \otimes u \\ T\|_u\big(f\colon A \otimes U \to B\big) &= T(f) \circ \mathrm{st}_{A,U} & (\mu\|_u)_A &= \mu_A \otimes u \end{aligned}$$

Proof. This is mainly a matter of unwinding definitions and being careful in which category compositions are taken. For example, the unit law $(\mu\|_u)_A \circ (\eta\|_u)_{T\|_u(A)} = T(A)$ in $\mathbf{C}\|_u$ comes down to the following diagram commuting in $\mathbf{C}$:

$$\begin{array}{ccc} T(A) \otimes U \otimes U & \xrightarrow{\eta_{TA}\otimes u\otimes U} & T^2(A) \otimes U \\ {\scriptstyle T(A)\otimes(u\otimes U)^{-1}}\uparrow & \nearrow^{\eta_{TA}\otimes U} & \downarrow{\scriptstyle \mu_A\otimes u} \\ T(A) \otimes U & \xrightarrow[T(A)\otimes u]{} & T(A) \end{array}$$

Similarly, naturality of $\eta\|_u$, which is $T\|_u(f) \circ (\eta\|_u)_A = (\eta\|_u)_B \circ f$ in $\mathbf{C}\|_u$, comes down to commutativity of the following diagram in $\mathbf{C}$:

$$\begin{array}{ccccc} A \otimes U \otimes U & \xrightarrow{\eta_A\otimes U\otimes u} & T(A) \otimes U \otimes I & \xrightarrow{\rho_{T(A)\otimes U}} & T(A) \otimes U \\ \| & & {\scriptstyle \mathrm{st}_{A,U}\otimes I}\downarrow & & \downarrow{\scriptstyle \mathrm{st}_{A,U}} \\ A \otimes U \otimes U & \longrightarrow & T(A \otimes U) \otimes I & \xrightarrow[\rho_{T(A\otimes U)}]{} & T(A \otimes U) \\ {\scriptstyle f\otimes U}\downarrow & & {\scriptstyle T(f)\otimes I}\downarrow & & \downarrow{\scriptstyle T(f)} \\ B \otimes U & \xrightarrow[\eta_B\otimes u]{} & T(B) \otimes I & \xrightarrow[\rho_{T(B)}]{} & T(B) \end{array}$$

Here the upper left square follows from (3), the right squares are naturality of unitors, and the lower left square is naturality of η in $\mathbf{C}$. The other laws are verified similarly. ◀

▶ **Example 17.** Consider a closure operator $T(u) = \overline{u}$ on a semilattice $\mathbf{C} = L$ as in Example 13. Then $T_u(a)$ is simply $\overline{a}$. This is a well-defined closure operator on the pre-order $\mathbf{C}\|_u$: if $a \wedge u \leq b$, then $\overline{a} \wedge u \leq \overline{a \wedge u} \leq \overline{b}$ because T is localisable. Collapsing the pre-order $\mathbf{C}\|_u$ to a partially ordered semilattice as in Remark 6 simply gives the downset $\downarrow u = \{a \in L \mid a \leq u\}$ of u in L, and T_u just becomes the restriction of the closure operator to $\downarrow u$.

Recall that a *(lax) monad morphism* [35] from a monad (S, η^S, μ^S) on $\mathbf{C}$ to a monad (T, η^T, μ^T) on $\mathbf{D}$ consists of a functor $F\colon \mathbf{C} \to \mathbf{D}$ and a natural transformation $\varphi\colon T \circ F \Rightarrow F \circ S$ making the following two diagrams commute:

$$\begin{array}{ccc} F & \xRightarrow{\eta^T_F} & T \circ F \\ & {\scriptstyle F\eta^S}\searrow \quad \swarrow{\scriptstyle \varphi} & \\ & F \circ S & \end{array} \qquad \begin{array}{ccccc} T^2 \circ F & \xRightarrow{T\varphi} & T \circ F \circ S & \xRightarrow{\varphi S} & F \circ S^2 \\ {\scriptstyle \mu^T_F}\Downarrow & & & & \Downarrow{\scriptstyle F\mu^S} \\ T \circ F & & \xRightarrow[\varphi]{} & & F \circ S \end{array} \tag{7}$$

Monads on $\mathbf{C}$ and their (lax) morphisms form a category $\mathbf{Monad}(\mathbf{C})$. An *oplax monad morphism* has $\psi\colon F \circ S \Rightarrow T \circ F$ that respects units and multiplication instead of φ.

▶ **Lemma 18.** *Let T be a localisable monad on $\mathbf{C}$. If $u \leq v$ are central idempotents, then the functor $\mathbf{C}\|_{u \leq v}$ from Lemma 8 is a (lax) monad morphism $T\|_v \to T\|_u$ with $\varphi_A = T(A) \otimes u$.*

Proof. Here we need to show the naturality of φ and the commutativity of the diagrams (7). These directly follow from (5), bifunctoriality of the tensor product and a few commuting diagrams that can be found in the extended version of this paper [5]. ◀

If $F\colon \mathbf{C} \to \mathbf{D}$ with $\varphi\colon T \circ F \Rightarrow F \circ S$ is a (lax) monad morphism between localisable monads S and T, and F is a (lax) monoidal functor with $\theta_{A,B}\colon F(A) \otimes F(B) \to F(A \otimes B)$, we say (F, φ, θ) is a (lax) *morphism of localisable monads* when the following diagram commutes:

$$\begin{array}{ccccc} TF(A) \otimes F(U) & \xrightarrow{\mathrm{st}_{FA,FU}} & T\big(F(A) \otimes F(U)\big) & \xrightarrow{T(\theta_{A,U})} & TF(A \otimes U) \\ {\scriptstyle \varphi_A \otimes F(U)}\downarrow & & & & \downarrow{\scriptstyle \varphi_{A,U}} \\ FS(A) \otimes F(U) & \xrightarrow[\theta_{S(A),U}]{} & F\big(S(A) \otimes U\big) & \xrightarrow[\mathrm{st}_{A,U}]{} & FS(A \otimes U) \end{array}$$

In this sense, the monad morphism $T\|_v \to T\|_u$ of Lemma 18 is localisable.

▶ **Corollary 19.** *If T is a localisable monad on $\mathbf{C}$, and $u \leq v$ are central idempotents, then the functor $\mathbf{C}\|^{u \leq v}$ from Lemma 8 is an oplax monad morphism $T\|_u \to T\|_v$ with $\psi_A = \mathrm{st}_{A,U}$.*

Proof. Applying [35, Theorem 9] to Lemmas 8 and 18, we can compute ψ as follows. By the adjunction, $\varphi_A\colon T\|_u(\mathbf{C}\|_{u \leq v}(\mathbf{C}\|^{u \leq v}(A))) \to \mathbf{C}\|_{u \leq v}(T\|_v(\mathbf{C}\|^{u \leq v}(A)))$ corresponds to a morphism

$$\mathbf{C}\|^{u \leq v}(T\|_u(\mathbf{C}\|_{u \leq v}(\mathbf{C}\|^{u \leq v}(A)))) \to T\|_v(\mathbf{C}\|^{u \leq v}(A))$$

and $\psi_A\colon \mathbf{C}\|^{u \leq v}(T\|_u(A)) \to T\|_v(\mathbf{C}\|^{u \leq v}(A))$ is obtained by precomposing this morphism with the unit $A \to \mathbf{C}\|_{u \leq v}(\mathbf{C}\|^{u \leq v}(A))$ of the adjunction. Starting with $\varphi_A = T(A) \otimes u$, this gives exactly $\psi_A = \mathrm{st}_{A,U}$. ◀

▶ Remark 20. If T is a localisable monad on a stiff symmetric monoidal category $\mathbf{C}$, and x is a point of $\mathrm{ZI}(\mathbf{C})$ regarded as a topological space, we can go further and define a monad $T\|_x$ on the stalk $\mathbf{C}\|_x$. The stalk $\mathbf{C}\|_x$ is defined as the colimit of the diagram $\mathbf{C}\|_{u \leq v}\colon \mathbf{C}\|_v \to \mathbf{C}\|_u$

ranging over all central idempotents $u \leq v$ containing the point x, taken in the category of symmetric monoidal categories. Accordingly, $T\|_x$ is the colimit over the same diagram, but now taken in the category of localisable monads. Using the concrete description in [2, Definition 7.1] of these stalks, we can compute:

$$T\|_x(A) = T(A) \qquad (\eta\|_x)_A = [1, \eta_A \circ \rho_A]$$
$$T\|_x\big([u, f\colon A \otimes U \to B]\big) = [u, T(f) \circ \mathrm{st}_{A,U}] \qquad (\mu\|_x)_A = [1, \mu_A \circ \rho_{T^2(A)}]$$

If $x \in u$ there is a localisable monad morphism $T\|_u \to T\|_x$ formed by the functor $\mathbf{C}\|_{x\in u}\colon \mathbf{C}\|_u \to \mathbf{C}\|_x$ given by $\mathbf{C}\|_{x\in u}(A) = A$ and $\mathbf{C}\|_{x\in u}(f\colon A \to B) = [u, f]$ with the identity natural transformation $\varphi\colon T\|_u \circ \mathbf{C}\|_{x\in u} \Rightarrow \mathbf{C}\|_{x\in u} \circ T\|_x$.

The representation of Theorems 10 and 11 is in fact functorial [2, Section 11]: a (lax) monoidal functor $T\colon \mathbf{C} \to \mathbf{C}$ corresponds to a family of stalk functors $T\|_x\colon \mathbf{C}\|_x \to \mathbf{C}\|_x$ that are continuous in a certain sense. However, this notion of continuity is quite involved, and we will not pursue it further here.

4 Formal monads, graded monads, and indexed monads

This section characterises localisable monads as formal monads in a certain presheaf category, and connects to graded monads and indexed monads.

4.1 Formal monads

We will characterise localisable monads as formal monads in the 2-category $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$ with functors $\mathrm{ZI}(\mathbf{C})^{\mathrm{op}} \to \mathbf{Cat}$ as 0-cells, *natural* transformations as 1-cells, and modifications as 2-cells [35, 25]. More precisely, we will define a formal monad *on* the sheaf $\overline{\mathbf{C}}\colon \mathrm{ZI}(\mathbf{C})^{\mathrm{op}} \to \mathbf{Cat}$ that maps a central idempotent u to the category $\mathbf{C}\|_u$ and morphisms $u \leq v$ to the functors $\mathbf{C}\|_{u\leq v}\colon \mathbf{C}\|_v \to \mathbf{C}\|_u$ of Lemma 8. A formal monad then consists of a natural transformation $\overline{T}\colon \overline{\mathbf{C}} \Rightarrow \overline{\mathbf{C}}$ and two modifications $\mu\colon \overline{T}\,\overline{T} \Rrightarrow \overline{T}$ and $\eta\colon \mathrm{id}_{\overline{\mathbf{C}}} \Rrightarrow \overline{T}$ satisfying the usual monad laws. More precisely, the data of this formal monad consists of:

- monads $(T\|_u, \mu\|_u, \eta\|_u)$ on $\mathbf{C}\|_u$ for every central idempotent u in $\mathbf{C}$;
- functors $\mathbf{C}\|_{u\leq v}\colon \mathbf{C}\|_v \to \mathbf{C}\|_u$ for central idempotents $u \leq v$ in $\mathbf{C}$;

such that the following equations hold in $\mathbf{C}\|_u$:

$$\begin{array}{ccc} \mathbf{C}\|_{u\leq v}(A) & \xrightarrow{\mathbf{C}\|_{u\leq v}((\eta\|_v)_A)} & \mathbf{C}\|_{u\leq v}(T\|_v(A)) \\ & \searrow^{(\eta\|_v)_{\mathbf{C}\|_{u\leq v}(A)}} & \| \\ & & T\|_u(\mathbf{C}\|_{u\leq v}(A)) \end{array} \qquad \begin{array}{ccc} T_u^2(\mathbf{C}\|_{u\leq v}(A)) & \xrightarrow{(\mu\|_u)_{\mathbf{C}\|_{u\leq v}(A)}} & T_u(\mathbf{C}\|_{u\leq v}(A)) \\ \| & & \| \\ \mathbf{C}\|_{u\leq v}(T_v^2(A)) & \xrightarrow[\mathbf{C}\|_{u\leq v}((\mu\|_v)_A)]{} & \mathbf{C}\|_{u\leq v}(T_v(A)) \end{array} \tag{8}$$

Moreover $\overline{T}$ is natural, meaning that if $u = v \circ m$ then for any $f\colon A \to B$ in $\mathbf{C}\|_v$:

$$T\|_u(\mathbf{C}\|_{u\leq v}A) = \mathbf{C}\|_{u\leq v}T\|_v(A) \tag{9}$$
$$T\|_u\big(\mathbf{C}\|_{u\leq v}f\big) = \mathbf{C}\|_{u\leq v}T\|_v(f). \tag{10}$$

Given the definition of $\mathbf{C}\|_{u\leq v}$, the first equation simply reads $T\|_u(A) = T\|_v(A)$. The following two lemmas follow from the definition of the adjoint functors $\mathbf{C}\|^{u\leq v} \dashv \mathbf{C}\|_{u\leq v}$.

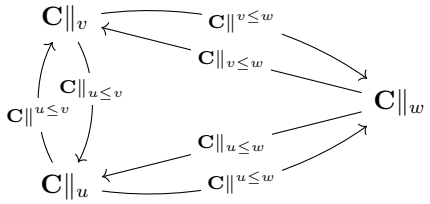
▶ **Lemma 21.** *There is a comonad $- \otimes U$ on $\mathbf{C}$ for any central idempotent u of $\mathbf{C}$. More generally, there is a comonad $- \otimes U$ on $\mathbf{C}\|_v$ for any central idempotents $u \leq v$ of $\mathbf{C}$.*

▶ **Lemma 22.** *The category $\mathbf{C}\|_u$ is the co-Kleisli category of the comonad $- \otimes U$ on $\mathbf{C}\|_v$.*

It follows from Lemma 22 that there is a canonical adjunction between the co-Kleisli category $\mathbf{C}\|_u$ and category $\mathbf{C}\|_v$ (or the base category $\mathbf{C}$ for $v = 1$) given by adjoint functors $\mathbf{C}\|^{u\leq v} \dashv \mathbf{C}\|_{u\leq v}$ such that $-\otimes U = \mathbf{C}\|^{u\leq v} \circ \mathbf{C}\|_{u\leq v}$. These correspond to the adjoint functors defined in Lemma 8. Further than Lemma 8, observe the following decomposition.

▶ **Lemma 23.** *If $u \leq v \leq w$ are central idempotents in $\mathbf{C}$, the functors of Lemma 8 satisfy:*

$$\mathbf{C}\|^{u\leq w} = \mathbf{C}\|^{v\leq w} \circ \mathbf{C}\|^{u\leq v}$$
$$\mathbf{C}\|_{u\leq w} = \mathbf{C}\|_{u\leq v} \circ \mathbf{C}\|_{v\leq w}$$
$$\mathbf{C}\|^{u\leq v} = \mathbf{C}\|_{v\leq w} \circ \mathbf{C}\|^{u\leq w}$$



Proof. This follows directly from the definition of the functors. ◀

▶ **Proposition 24.** *Let $\mathbf{C}$ be a stiff category. Let $(\overline{T}, \overline{\mu}, \overline{\eta})$ be a formal monad in $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$ above $\overline{\mathbf{C}}$ and let $u \leq v$ be central idempotents. Then the monad $T\|_v$ is a localisable monad with the strength $\mathrm{st}_{A,U} \colon T\|_v(A) \otimes U \to T\|_v(A \otimes U)$ defined as the following composition in $\mathbf{C}\|_v$ for any object A in $\mathbf{C}\|_v$:*

$$\begin{array}{c} T\|_v(A) \otimes U = \mathbf{C}\|^{u\leq v}\mathbf{C}\|_{u\leq v}T\|_v A = \mathbf{C}\|^{u\leq v}T\|_u\mathbf{C}\|_{u\leq v}A \\ \Big\downarrow {\scriptstyle \mathbf{C}\|^{u\leq v}T\|_u\eta^{u\leq v}_{\mathbf{C}\|_{u\leq v}A}} \\ \mathbf{C}\|^{u\leq v}T\|_u\mathbf{C}\|_{u\leq v}\mathbf{C}\|^{u\leq v}\mathbf{C}\|_{u\leq v}A = \mathbf{C}\|^{u\leq v}\mathbf{C}\|_{u\leq v}T\|_v\mathbf{C}\|^{u\leq v}\mathbf{C}\|_{u\leq v}A \\ \Big\downarrow {\scriptstyle \varepsilon^{u\leq v}_{T\|_v\mathbf{C}\|^{u\leq v}\mathbf{C}\|_{u\leq v}A}} \\ T\|_v\mathbf{C}\|^{u\leq v}\mathbf{C}\|_{u\leq v}A = T\|_v(A \otimes U) \end{array} \tag{11}$$

where $\eta^{u\leq v}$ and $\varepsilon^{u\leq v}$ are the unit and counit of adjunction $\mathbf{C}\|^{u\leq v} \dashv \mathbf{C}\|_{u\leq v}$.

Proof. We need to prove each of the axioms of Definition 12. This consist of many commutating diagrams. The complete proof can be found in the extended version of this paper [5]. For simplicity, the proof is laid out for the case $v = 1$, but the same arguments hold for any $T\|_v$ by using the relevant strength. ◀

▶ **Proposition 25.** *A localisable monad T on a stiff category $\mathbf{C}$ induces a formal monad on $\overline{\mathbf{C}}$ in $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$. The natural transformation $\overline{T} \colon \overline{\mathbf{C}} \Rightarrow \overline{\mathbf{C}}$ has components $T\|_u$, the modification $\overline{\eta} \colon \overline{\mathbf{C}} \Rrightarrow \overline{T}$ has components $\eta\|_u$, and the modification $\overline{\mu} \colon \overline{T}^2 \Rrightarrow \overline{T}$ has components $\mu\|_u$ as in Proposition 16.*

Proof. This proof consist in verifying the naturality of $\overline{T}$, in showing that $\overline{\eta}$ and $\overline{\mu}$ are modifications (which follows directly from Lemma 18) and natural, and in proving that $\overline{\eta}$ and $\overline{\mu}$ satisfy the monad laws (which pointwise follows from Proposition 16). The complete proof is included in the extended version of this paper [5]. ◀

▶ **Theorem 26.** *For a stiff monoidal category $\mathbf{C}$ there is a bijective correspondence between localisable monads on $\mathbf{C}$ and formal monads on $\overline{\mathbf{C}}$ in $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$ (via the constructions of Propositions 24 and 25).*

Proof. Start with a localisable monad T and follow Proposition 25 to get a formal monad $\overline{T}$. Then apply Proposition 24 to get a localisable monad T' which we claim equals the original monad T. It is clear that T' equals T as a functor. It remains to check that the strength

obtained this way on T' is the same as the original strength on T. To do this, note that the strength (11) from Proposition 24 can be rewritten as follows, where st denotes the original strength from the localisable monad:

$$\varepsilon_{T\|_1 FGA} \circ FT\|_u \eta^u_{GA} = (T(A \otimes U) \otimes u) \circ (\mathrm{st}_{A,U} \otimes U) \otimes (T(A) \otimes (U \otimes u)^{-1}) = \mathrm{st}_{A,U}$$

Here we use the naturality of the strength and the fact that $U \otimes u$ is an isomorphism. We prove similarly that using Proposition 25 then Proposition 24 gives us back the unit and the multiplication of the starting localisable monad. To simplify the notation we used F and G to denote $\mathbf{C}\|^{u\leq 1}$ and $\mathbf{C}\|_{u\leq 1}$.

Now start with a formal monad $\overline{T}$, turn it into a localisable monad $(\overline{T}\|_1, \mathrm{st})$, and then into a formal monad $\widetilde{T}$. Then $\widetilde{T}\|_u(A) = \overline{T}\|_u(A)$ and $\widetilde{T}\|_u$ sends a morphism $f : GA \to GB$ in $\mathbf{C}\|_u$ given by $f\colon A \otimes U \to B$ to the morphism $\overline{T}\|_u(A) \to \overline{T}\|_u(B)$ in $\mathbf{C}\|_u$ given by:

$$\overline{T}\|_1(A) \otimes U \xrightarrow{\mathrm{st}_{A,U}} \overline{T}\|_1(A \otimes U) \xrightarrow{\overline{T}\|_1(f)} \overline{T}\|_1(B)$$

We have to prove that this equals $\overline{T}\|_u(f)$. To see this, first note that by the properties of the adjunction, a map f in the coKleisi category $\mathbf{C}\|_u$ is defined in the base category as $\varepsilon \circ F(f)$, which we will denote $f^{\mathbf{C}}$. With this notation, and again using F and G to denote $\mathbf{C}\|^{u\leq 1}$ and $\mathbf{C}\|_{u\leq 1}$, we get:

$$\begin{aligned}
\overline{T}\|_1(f^{\mathbf{C}}) \circ \mathrm{st}_{A,U} &= \overline{T}\|_1 \varepsilon_B \circ \overline{T}\|_1 Ff \circ \varepsilon_{\overline{T}\|_1 FGA} \circ F\overline{T}\|_u \eta_{GA} && (12)\\
&= \varepsilon_{\overline{T}\|_1 B} \circ FG\overline{T}\|_1 \varepsilon_B \circ FG\overline{T}\|_1 Ff \circ F\overline{T}\|_u \eta_{GA} && (13)\\
&= \varepsilon_{\overline{T}\|_1 B} \circ F\overline{T}\|_u G\varepsilon_B \circ F\overline{T}\|_u GFf \circ F\overline{T}\|_u \eta_{GA} && (14)\\
&= \varepsilon_{\overline{T}\|_1 B} \circ F\overline{T}\|_u G\varepsilon_B \circ F\overline{T}\|_u \eta_{GB} \circ F\overline{T}\|_u f && (15)\\
&= \varepsilon_{\overline{T}\|_1 B} \circ F\overline{T}\|_u f && (16)\\
&= (\overline{T}\|_u(f))^{\mathbf{C}} && (17)
\end{aligned}$$

Line (12) follows from the definition of the strength given in Equation (11) and the definition of $f^{\mathbf{C}}$. The next three lines follow from naturality of ε used twice, Equation (10), and naturality of η respectively. Line (16) uses the property of the adjunction and the last line uses the definition of $(\overline{T}\|_u(f))^{\mathbf{C}}$.

Similarly, using Proposition 24 and then Proposition 25 gives back the unit and the multiplication of the original formal monad. ◀

4.2 Graded monads and indexed monads

We now connect these notions to the pre-existing notions of $\mathbf{E}$-indexed monads and $\mathbf{E}$-graded monads for a monoidal category $\mathbf{E}$. Recall that an $\mathbf{E}$-*graded* monad is a lax monoidal functor $\mathbf{E} \to [\mathbf{C}, \mathbf{C}]$. It consists of functors $T_u\colon \mathbf{C} \to \mathbf{C}$, a natural transformation $\eta_A\colon A \to T_I(A)$, and a transformation $\mu_{u,v,A}\colon T_u(T_v(A)) \to T_{u\otimes v}(A)$ natural in u, v, and A, satisfying some coherence diagrams [12].

On the other hand, an $\mathbf{E}$-*indexed* monad is a functor $\mathbf{E} \to \mathbf{Monad}(\mathbf{C})$. It also consists of functors $T_u\colon \mathbf{C} \to \mathbf{C}$, but now with transformations $\eta_{u,A}\colon A \to T_u(A)$ and transformations $\mu_{u,A}\colon T_u^2(A) \to T_u(A)$ natural in u and A, such that each (T_u, η_u, μ_u) forms a monad. The formal monads on $\overline{\mathbf{C}}$ as defined in Section 4 are $\mathrm{ZI}(\mathbf{C})$-indexed monads. The next lemma provides conditions under which indexed monads induce graded monads and vice versa.

Recall that a monoidal category *has codiagonals* when there is a natural transformation $A \otimes A \to A$ that respects the coherence isomorphisms [18].

▶ **Lemma 27.** *Let* $\mathbf{E}$ *be a monoidal category. If the tensor unit is initial, then an* $\mathbf{E}$*-indexed monad induces a* $\mathbf{E}$*-graded monad. If the tensor product has codiagonals, then an* $\mathbf{E}$*-graded monad induces an* $\mathbf{E}$*-indexed monad. If* $\mathbf{E}$ *is cocartesian, there is a bijective correspondence between* $\mathbf{E}$*-graded monads and* $\mathbf{E}$*-indexed monads.*

Proof. Suppose the tensor unit 0 in $\mathbf{E}$ is initial. An $\mathbf{E}$-indexed monad (T_u, η_u, μ_u) then induces an $\mathbf{E}$-graded monad with the same T_u but $\overline{\eta}_A = \eta_{0,A}$ and $\overline{\mu}_{u,v,A}$ given by:

$$T_u(T_v(A)) \xrightarrow{T_\rho{}^{-1}(T_\lambda{}^{-1}(A))} T_{u\otimes 0}(T_{0\otimes v}(A)) \xrightarrow{T_{u\otimes !}(T_{!\otimes v}(A))} T^2_{u\otimes v}(A) \xrightarrow{\mu_{u\otimes v,A}} T_{u\otimes v}(A)$$

Now suppose that $\mathbf{E}$ has codiagonals. An $\mathbf{E}$-graded monad $(T_u, \eta, \mu_{u,v})$ then induces an $\mathbf{E}$-indexed monad with the same T_u but $\overline{\eta}_{u,A} = \eta_A$ and $\overline{\mu}_{u,A}$ given by:

$$T^2_u(A) \xrightarrow{\mu_{u,u,A}} T_{u\otimes u}(A) \xrightarrow{T_{\nabla_u}(A)} T_u(A)$$

If $\mathbf{E}$ is cocartesian, these two constructions are each other's inverse. For example, $\overline{\overline{\mu}}_{u,A} = \mu_{u,A}$ because:

$$\begin{array}{ccccc} T_{u+0}(T_{0+u}(A)) & \xrightarrow{T_{u+!}(T_{!+u}(A))} & T^2_{u+u}(A) & \xrightarrow{\mu_{u+u,A}} & T_{u+u}(A) \\ & \searrow^{T_\rho(T_\lambda(A))} & \downarrow{\scriptstyle T^2_{\nabla_u}(A)} & & \downarrow{\scriptstyle T_{\nabla_u}(A)} \\ & & T^2_u(A) & \xrightarrow[\mu_{u,A}]{} & T_u(A) \end{array}$$

Also $\overline{\overline{\eta}}_A = \eta_A$ because $!\colon 0 \to 0$ is the identity. The other properties follow from naturality in u and v. ◀

In particular, it follows that there is no difference between graded monads and indexed monads over (join-)semilattices.

5 Examples

In this section we discuss three extended examples, showing that localisable monads may interpret central idempotents as locations in a computer memory (Subsection 5.1), physical locations in a network of interacting agents (Subsection 5.2), or time in extended processes (Subsection 5.3). These examples use the following characterisation of central idempotents in functor categories.

▶ **Lemma 28.** *If* $\mathbf{C}$ *is a category and* $\mathbf{D}$ *is a symmetric monoidal category, then the functor category* $[\mathbf{C}, \mathbf{D}]$ *is again symmetric monoidal under pointwise tensor products. Regarding* $\mathrm{ZI}(\mathbf{D})$ *as a full subcategory of the slice category* $\mathbf{D}/I$*, there is an isomorphism of categories:*

$$\mathrm{ZI}[\mathbf{C}, \mathbf{D}] \simeq [\mathbf{C}, \mathrm{ZI}(\mathbf{D})]$$

Proof. Let $u\colon U \to I$ be a central idempotent in $[\mathbf{C}, \mathbf{D}]$. The functor $[\mathbf{C}, \mathbf{D}] \to \mathbf{D}$ that evaluates at a fixed object $C \in \mathbf{C}$ is strong monoidal and so preserves central idempotents. Hence each component $u_C\colon U(C) \to I$ represents a central idempotent in $\mathbf{D}$. This is functorial and gives one direction of the isomorphism.

Conversely, let $F\colon \mathbf{C} \to \mathrm{ZI}(\mathbf{D})$ be a functor. Define $U\colon \mathbf{C} \to \mathbf{D}$ by $U(C) = \mathrm{dom}(F(C))$ and $u\colon U \Rightarrow I$ by $u_C = F(C)$. This is functorial and gives the other direction of the isomorphism. It is clear that these two assignments are inverses. ◀

5.1 Quantum buffer

The (global) state monad on **Set** is a well-known monad that combines the properties of the reader and writer monads to implement computational side-effect in functional programming. It is defined as $T(-) = S \multimap (- \times S)$ for a state object $S \in \mathbf{Set}$. For example, to store one bit, take $S = \{0, 1\}$. The central idempotents of **Set** are (represented by) the empty set $\emptyset$ and the singleton set 1. It follows that the (global) state monad is trivially localisable. This example is trivial but can be expanded in several ways:

1. Expanded to the category $\mathbf{Set}^n$, whose objects are n-tuples of sets and morphisms are n-tuples of functions. The state monad on some object $A = (A_1, \ldots, A_n)$ in $\mathbf{Set}^n$ is

$$T(A_1, \ldots, A_n) = (S_1, \ldots, S_n) \multimap \big((A_1, \ldots, A_n) \times (S_1, \ldots, S_n)\big)$$

   for a chosen state object $S = (S_1, \ldots, S_n) \in \mathbf{Set}^\mathbf{n}$. For example, to store n bits, take $S_1 = \cdots = S_n = \{0, 1\}$. It follows from Lemma 28 that $\mathrm{ZI}(\mathbf{Set}^n) \simeq 2^n$. While $\mathbf{Set}^n$ is symmetric monoidal closed, the state monad does not satisfy $T(A \multimap U) = T(A) \multimap T(U)$ as in Example 14. There is still a strength, by currying the evaluation:

$$T(A_1, \ldots, A_n) \times (U_1, \ldots, U_n) \times (S_1, \ldots, S_n) \to (S_1, \ldots, S_n) \times (A_1, \ldots, A_n) \times (U_1, \ldots, U_n)$$

   We have not discussed commutativity yet, but note that this strength is commutative in a sense made clear in Definition 34 below. Conceptually, this means that the computational side-effects modelled by a state monad "over" a region $(U_1, \ldots, U_n)$ are independent of those modelled by $(V_1, \ldots, V_n)$, assuming that $(U_1, \ldots, U_n) \times (V_1, \ldots, V_n) = 0$.
2. The localisable state monad of the previous point does not just work for cartesian closed categories such as $\mathbf{Set}^n$, but also for exponentiable objects in a symmetric monoidal category. For example, we can replicate it in the category **Hilb** of Hilbert spaces and completely positive linear maps used in quantum computation [16]. To store one qubit, take $S = \mathbb{C}^2$. The monad then becomes $T(-) = S^* \otimes - \otimes S$, where $S^* = \mathbf{Hilb}(S, \mathbb{C})$ is the dual Hilbert space, which is isomorphic to $T(A) = A \otimes \mathbb{M}_2$, where $\mathbb{M}_2$ is the Hilbert space of complex 2-by-2 matrices. Similarly, to store n qubits, move to $\mathbf{Hilb}^n$. We can now see a phenomenon that didn't occur for cartesian categories: rather than a quantum memory, this monad models a quantum buffer of n qubits, because there is no entanglement between the different qubits. Because $\mathrm{ZI}(\mathbf{Hilb}) = \{0, \mathbb{C}\}$, again $\mathrm{ZI}(\mathbf{Hilb}^n) \simeq 2^n$. The strength map is yet again given by the curry of the evaluation map, which makes $T(-)$ a commutative localisable monad in the sense of Definition 34 below.
3. We can also promote the (global) state monad on **Set** in another direction, namely from $n = 1$ or finite n to an arbitrary topological space X indexing the bits to be stored. Consider the category $\mathrm{Sh}(X)$ of (**Set**-valued) sheaves on X, take S to be the constant sheaf $S(U) = \{0, 1\}$, and define $T(-) = S \multimap (- \otimes S)$. As in Example 3, the central idempotents correspond to open subsets $U \subseteq X$, and this monad is still localisable. Its stalks (as discussed in Remark 20) are the simple (global) state monads on **Set** storing a single bit each.
4. Points 2 and 3 combine to model a quantum buffer over an arbitrary locally compact Hausdorff topological space X. Consider the category $\mathbf{Hilb}_{C_0(X)}$ of Hilbert modules over $C_0(X)$, take S to be Hilbert module $C_0(X, \mathbb{C}^2)$ of continuous functions $X \to \mathbb{C}^2$ that vanish at infinity, and define $T(-) = S^* \otimes - \otimes S$. As in Example 4, central idempotents are open subsets $U \subseteq X$. Again, this monad is localisable, with $T\|_U = S_u^* \otimes - \otimes S_u$ for $S_u = C_0(U, \mathbb{C}^2)$. In fact, this example is related to the one in point 3, as Hilbert modules over $C_0(X)$ correspond to a Hilbert space internal to the topos $\mathrm{Sh}(X)$ by Takahashi's Theorem [2, 15].

5.2 Concurrent processes

Suppose M_1 is a monoid of actions that some agent 1 can perform, and M_2 is a monoid of actions that an agent 2 can perform. They could, for example, be free monoids over sets of atomic actions. Then we can form the coproduct $M_1 + M_2$ of monoids, and quotient out a congruence that specifies $ab = ba$ for $a \in M_1$ and $b \in M_2$ when actions a and b are independent, to get the monoid M of Mazurkiewicz traces [8, 37]. Now M localises to M_1 by projections $M \to M_i$ that disregard actions of the other agent.

The following lemma engineers a single category with two central idempotents and a monoid, that localises to the given ones. The idea is to take a product of categories, but to add *silent actions*, that enforce the order in which both agents' actions occur, as in the pi calculus [27].

▶ **Lemma 29.** *Let M_1 and M_2 be monoids in symmetric monoidal categories $\mathbf{C}_1$ and $\mathbf{C}_2$ that have an initial object 0 satisfying $A \otimes 0 \simeq 0$ for all objects A. There is a symmetric monoidal category $\mathbf{C}$ with a monoid M and central idempotents u_1, u_2, that allows an isomorphism $\mathbf{C}\|_{u_i} \simeq \mathbf{C}_i$ of monoidal categories under which M_i corresponds with $\mathbf{C}\|_{u_i \leq 1}(M)$.*

If $\mathbf{C}_i$ does not yet have an initial object 0 satisfying $A \otimes 0 \simeq 0$, we may freely adjoin one to obtain a well-defined symmetric monoidal category.

Proof. First construct a new category $\mathbf{C}'$. Objects are pairs (A, B) of $A \in \mathbf{C}_1$ and $B \in \mathbf{C}_2$. Morphisms $(A, B) \to (A', B')$ include pairs (f, g) of $f \in \mathbf{C}_1(A, A')$ and $g \in \mathbf{C}_2(B, B')$, to which we freely adjoin morphisms $\tau_{A,B} \colon (A, B) \to (A, B)$ for each object (A, B). Thus morphisms are finite lists $\big((f_1, g_1), \tau_1, \ldots, \tau_{n-1}, (f_n, g_n)\big)$ where the domain of τ_n is the codomain of $f_n \otimes g_n$. Composition concatenates and then contracts:

$$\begin{aligned}&\big((f_1', g_1'), \tau_1', \ldots, (f_n', g_n')\big) \circ \big((f_1, g_1), \tau_1, \ldots, (f_m, g_m)\big)\\ &= \big((f_1, g_1), \tau_1, \ldots, (f_1' \circ f_m, g_1' \circ g_m), \tau, \ldots, (f_n', g_n')\big)\end{aligned}$$

Defining identity to be the trivial list $(\mathrm{id}[A], \mathrm{id}[B])$ makes $\mathbf{C}'$ into a well-defined category.

Next, take the free symmetric monoidal category $\mathbf{C}''$ on $\mathbf{C}'$. Objects of $\mathbf{C}''$ are finite lists of objects of $\mathbf{C}'$, and morphisms are pairs $(\pi, h_1, \ldots, h_n)$ of a permutation π of list indices and a list of morphisms in $\mathbf{C}'$; see for example [1]. Finally, consider the generalised equivalence relation [4] on $\mathbf{C}''$ generated by

$$\begin{aligned}(I, 0) \otimes \tau_{A,B} &\sim (A, 0)\\ (0, I) \otimes \tau_{A,B} &\sim (0, B)\\ \big(\pi, (f_1, g_1), (f_2, g_2)\big) &\sim (\sigma, (f_1 \otimes f_2, g_1 \otimes g_2))\end{aligned}$$

where π is the bijection $1 \mapsto 2$ and $2 \mapsto 1$ on $\{1, 2\}$. This is a symmetric monoidal congruence, so $\mathbf{C} = \mathbf{C}''/\!\sim$ is a well-defined symmetric monoidal category.

Because 0 is initial and $A \otimes 0 = 0$ in $\mathbf{C}_i$, the objects $(I, 0)$ and $(0, I)$ in $\mathbf{C}'$ become central idempotents u_1, u_2 in $\mathbf{C}$, and moreover $(A, B) \mapsto [A]_\sim$ is an isomorphism $\mathbf{C}\|_{u_1} \simeq \mathbf{C}_1$ and similarly for u_2. Finally, $M = (M_1, M_2)$ is a monoid in $\mathbf{C}$, that localises to M_i by construction. ◀

In the proof of the previous lemma, we could alternatively have described $\mathbf{C}$ as consisting of formal string diagrams generated by $\mathbf{C}_1 \times \mathbf{C}_2$ and the silent actions $\tau_{A,B}$ [6], or as terms in a formal syntactic language [19].

▶ **Example 30.** Let M_i be monoids in $\mathbf{C}_i = \mathbf{Set}$. They induce writer monads $T_i(A) = M_i \otimes A$ on $\mathbf{C}_i$. Now the monoid M in the category $\mathbf{C}$ of the previous lemma induces a writer monad T on $\mathbf{C}$. The monad T is localisable by Example 14, and $T\|_i$ corresponds to T_i under the isomorphism $\mathbf{C}\|_i \simeq \mathbf{C}_i$. Thus T tracks the agents' actions as side effects during a (distributed) computation.

It seems possible to extend this example to a network where the communicating agents form the points of an arbitrary topological space.

5.3 Stochastic processes

Write **Meas** for the category of measurable spaces and measurable functions. This is a symmetric monoidal category, where the tensor unit is the singleton set with its unique σ-algebra, and the tensor product of two measurable spaces is the cartesian product of the sets with the tensor product of the σ-algebras. The monoidal category **Meas** has only two central idempotents: the empty set $\emptyset$, and the tensor unit 1 itself.

Instead, consider the functor category $[\mathbb{N}, \mathbf{Meas}]$, where the partially ordered set $\mathbb{N}$ is considered as a category by having a morphism $m \to n$ if and only if $m \leq n$. Its objects are sequences $X_1, X_2, X_3, \ldots$ of measurable spaces. Lemma 28 shows that this category has many more central idempotents. It follows that central idempotents $u \colon U \Rightarrow 1$ in $[\mathbb{N}, \mathbf{Meas}]$ correspond to upward-closed subsets of $\mathbb{N} \cup \{\infty\}$, or more succinctly, to elements of $n \in \mathbb{N} \cup \{\infty\}$, by

$$U(m) = \begin{cases} \emptyset & \text{if } m < n \\ 1 & \text{if } m \geq n \end{cases}$$

The *Giry monad* $G \colon \mathbf{Meas} \to \mathbf{Meas}$ takes a measurable space to the set of probability measures on it [13]. It extends to a monad on $[\mathbb{N}, \mathbf{Meas}]$.

▶ **Example 31.** The monad $\widehat{G} = G \circ (-)$ on $[\mathbb{N}, \mathbf{Meas}]$ is localisable, where the maps $\widehat{G}(X) \otimes U \Rightarrow \widehat{G}(X \otimes U)$ can simply be taken to be identities (because $G(\emptyset) = \emptyset$). The restricted category $[\mathbb{N}, \mathbf{Meas}]\|_n$ is $[\{n, n+1, \ldots\}, \mathbf{Meas}]$, and the monad $\widehat{G}\|_n$ is simply the restriction of $\widehat{G}$ to $\{n, n+1, \ldots\}$.

The adjunction between **Meas** and the Kleisli category $\mathrm{Kl}(G)$ lifts to an adjunction between $[\mathbb{N}, \mathbf{Meas}]$ and $[\mathbb{N}, \mathrm{Kl}(G)]$. The latter is not equivalent to the Kleisli category of $\widehat{\mathbf{G}}$ because the functor $[\mathbb{N}, \mathbf{Meas}] \to [\mathbb{N}, \mathrm{Kl}(G)]$ that turns a sequence of elements of measurable spaces into a sequence of Dirac measures is not essentially surjective [36, Theorem 9].

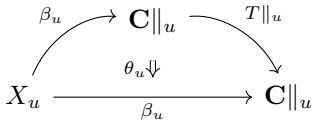
The objects of $[\mathbb{N}, \mathbf{Meas}]$ are *stochastic processes* [24, 13, 11]. Instead of $(\mathbb{N}, \leq)$, we could equally well have taken continuous time $(\mathbb{R}^{\geq 0}, \leq)$. In fact, we could also have regarded the monoid $(\mathbb{N}, +, 0)$ or $(\mathbb{R}^{\geq 0}, +, 0)$ as a one-object category. Then $[\mathbb{N}, \mathrm{Kl}(G)]$ would consist of stationary processes, but the central idempotents would remain the same by Lemma 28: ideals of $\mathbb{N}$ or $\mathbb{R}^{\geq 0}$ under $+$ are also upward-closed subsets.

Rather than stochastic (Markov) processes, that depend on the history thus far (one time step ago only), we could have taken more interesting partially ordered sets than the totally ordered ones $\mathbb{N}$ and $\mathbb{R}^{\geq 0}$.

6 Algebras

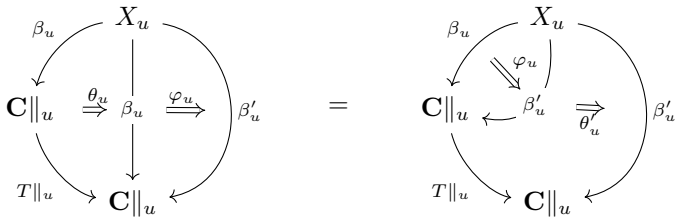
Let $\mathbf{C}$ be a symmetric monoidal category. As we have seen in Section 4, a localisable monad $T \colon \mathbf{C} \to \mathbf{C}$ is equivalently described as a formal monad $T\|\_$ in the 2-category $\mathbf{K} = [\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$. What are its formal (Eilenberg-Moore) algebras?

The general answer is described in [23, 35]. The formal algebra category is an object of $\mathbf{K}$ satisfying the following. For any object $X \in \mathbf{K}$, the formal monad $T\|_-$ induces a (concrete) monad $K(X, T\|_-)$ on the category $\mathbf{K}(X, \mathbf{C}\|_-)$; this monad sends a natural transformation $\beta\colon X \Rightarrow \mathbf{C}\|_-$ to the natural transformation with components $T\|_u \circ \beta_u\colon X_u \to \mathbf{C}\|_u$. This (concrete) monad has a (concrete) Eilenberg-Moore category of algebras. Objects are pairs of a natural transformation β and a modification θ of type



(18)

satisfying the algebra laws. Morphisms are modifications $\varphi\colon \beta \Rrightarrow \beta'$ satisfying:



(19)

This defines the object-part of a 2-functor $\mathbf{K}^{\mathrm{op}} \to \mathbf{Cat}$. Now $A \in \mathbf{K}$ is the *formal algebra* object of the formal monad $\mathbf{T}\|_-$ when this 2-functor is naturally isomorphic to $\mathbf{K}(-, A)$.

▶ **Proposition 32.** *Let T be a localisable monad on a symmetric monoidal category $\mathbf{C}$. The formal monad $T\|_-$ in $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$ has a formal algebra object $A\|_-$ where $A\|_u = \mathrm{Alg}(T\|_u)$ is the category of algebras of $T\|_u$.*

Proof. If $u \leq v$ then the monad morphism $\mathbf{C}\|_{u \leq v}$ of Lemma 18 induces a functor $A\|_v \to A\|_u$, so A is a well-defined object of $\mathbf{K} = [\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$. Now, for an object $X \in \mathbf{K}$, the hom-category $\mathbf{K}(X, A)$ has as objects natural transformations $\beta_u\colon X_u \to \mathrm{Alg}(T\|_u)$. But the objects of $\mathrm{Alg}(T\|_u)$ are themselves morphisms $\theta_u\colon T\|_u(B) \to B$ in $\mathbf{C}\|_u$, that furthermore satisfy the algebra laws. These assemble into a modification satisfying (18). It is labour-intensive but straightforward to verify that the morphisms of $\mathrm{Alg}(T\|_u)$ similarly match modifications satisfying (19), and that this in fact gives a 2-natural isomorphism to $A\|_-$. Thus $A\|_-$ is a formal algebra object. ◀

Similarly, a *formal Kleisli algebra* object of the formal monad $T\|_-$ is characterised in [23, 35] as a formal algebra object in the 2-category $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}^{\mathrm{op}}]$, where $\mathbf{Cat}^{\mathrm{op}}$ has reversed the 1-cells but not the 2-cells of $\mathbf{Cat}$.

▶ **Corollary 33.** *Let T be a localisable monad on a symmetric monoidal category $\mathbf{C}$. The formal monad $T\|_-$ in $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$ has a formal Kleisli object $K\|_-$ where $K\|_u = \mathrm{Kl}(T\|_u)$ is the Kleisli category of $T\|_u$.* ◀

A Kleisli category of a commutative monad on a symmetric monoidal category is again symmetric monoidal [7]. It would be interesting to see if there is a notion that stands to localisability as commutativity stands to strength, that guarantees that the formal Kleisli algebra object of the previous corollary is a monoid in $\mathbf{K} = [\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$. We leave this for future work, but give a tentative (re)definition now.

▶ **Definition 34.** *A localisable monad T on a symmetric monoidal category* $\mathbf{C}$ *is* commutative *when:*

$$\begin{array}{ccccc} T(A)\otimes U\otimes V & \xrightarrow{\mathrm{st}_{A,U}\otimes V} & T(A\otimes U)\otimes V & \xrightarrow{\mathrm{st}_{A\otimes U,V}} & T(A\otimes U\otimes V) \\ {\scriptstyle T(A)\otimes\sigma_{U,V}}\downarrow & & & & \uparrow{\scriptstyle T(A\otimes\sigma_{V,U})} \\ T(A)\otimes V\otimes U & \xrightarrow[\mathrm{st}_{A,V}\otimes U]{} & T(A\otimes V)\otimes U & \xrightarrow[\mathrm{st}_{A\otimes V,U}]{} & T(A\otimes V\otimes U) \end{array} \tag{20}$$

It follows from this definition that if $u \wedge v = 0$, then the computational side-effects modeled by T_u and T_v do not influence each other. Intuitively, side-effects T_u and T_v that act in disjoint areas must be independent of each other.

7 Further work

There are several interesting directions for further research.

- We have decomposed a localisable monad into monads on local monoidal categories, but can a monad on a local monoidal category be decomposed further? For example, the local state monad [29] is based on the presheaf category $[\mathbf{Inj}, \mathbf{Set}]$. Its central idempotents correspond to natural numbers, topologised by saying that a subset is open when it is upward-closed under the usual ordering of natural numbers. This topological space is already local: every net converges to the focal point 0. The "decomposition" using coends of [29] relies on the base category $[\mathbf{Inj}, \mathbf{Set}]$ having much more structure rather than just a monoidal category. The successor function of natural numbers there affords the possibility to allocate fresh locations. Our example of local states in Section 5.1 completely ignored this possibility. Can this extra structure be axiomatised – using open sets rather than points – and used for a further decomposition?
- Two monads on the same base category can be composed as soon as there is a distributive law between them [3, 38]. When does a distributive law respect the localisable nature of the monads, and how does it interact with their decomposition into monads on local monoidal categories?
- More generally than monads, when is a PROP localisable, and how does a localisable PROP decompose into local ones [22, 33]?
- Formal monads form a bridge between localisable monads and the local-to-global approach to providing a fine-grained structure over monads. Can this relationship be made more constructive? Given monads T_i on possibly different monoidal base categories $\mathbf{C}_i$, can we construct a monad T on a monoidal category $\mathbf{C}$ with central idempotents i such that $\mathbf{C}\|_i \simeq \mathbf{C}_i$ and $T\|_i \simeq T_i$? The free construction of Lemma 29 is an initial step in this direction; can it be given a more elegant concrete description, and extended to arbitrary topogical spaces?
- Is there a notion that stands to localisability as commutativity stands to strength, that guarantees that the formal Kleisli object of Corollary 33 is a monoid in $[\mathrm{ZI}(\mathbf{C})^{\mathrm{op}}, \mathbf{Cat}]$? Does it connect to partial commutativity as in the Mazurkiewicz traces of Section 5.2?

References

1 S. Abramsky. Abstract scalars, loops, and free traced and strongly compact closed categories. In *Conference on Algebra and Coalgebra*, volume 3629 of *Lecture Notes in Computer Science*, pages 1–31. Springer, 2005. doi:10.1007/11548133_1.

2 R. Soares Barbosa and C. Heunen. Sheaf representation of monoidal categories. arxiv:2106.08896, 2021.

3 J. Beck. Distributive laws. In *Seminar on Triples and Categorical Homology Theory*, pages 119–140. Springer, 1969. doi:10.1007/BFb0083084.

4 M. A. Bednarczyk, A. M. Borzyszkowski, and W. Pawlowski. Generalized congruences – epimorphisms in Cat. *Theory and Applications of Categories*, 5(11):266–280, 1999.

5 C. Constantin, N. Dicaire, and C. Heunen. Localisable monads. *arXiv preprint*, 2021. arXiv:2108.01756.

6 P.-L. Curien and S. Mimram. Coherent presentations of monoidal categories. *Logical Methods in Computer Science*, 13(3):1–38, 2017. doi:10.23638/LMCS-13(3:31)2017.

7 B. Day. On closed category of functors II. In *Sydney Category Theory Seminar*, number 420 in Lecture Notes in Mathematics, pages 20–54, 1974.

8 V. Diekert and Y. Métivier. *Handbook of formal languages*, chapter Partial commutation and traces, pages 457–533. Springer, 1997. doi:10.1007/978-3-642-59126-6_8.

9 P. Enrique Moliner, C. Heunen, and S. Tull. Space in monoidal categories. In *Electronic Proceedings in Theoretical Computer Science*, volume 266, pages 399–410, 2017. doi:10.4204/EPTCS.266.25.

10 P. Enrique Moliner, C. Heunen, and S. Tull. Tensor topology. *Journal of Pure and Applied Algebra*, 224(10):106378, 2020. doi:10.1016/j.jpaa.2020.106378.

11 T. Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020. doi:10.1016/j.aim.2020.107239.

12 S. Fujii, S. Katsumata, and P.-A. Melliès. Towards a formal theory of graded monads. In *Foundations of Software Science and Computation Structures*, pages 513–530. Springer, 2015. doi:10.1007/978-3-662-49630-5_30.

13 M. Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, volume 915 of *Lecture Notes in Mathematics*, pages 68–85. Springer, 1981. doi:10.1007/BFb0092872.

14 C. Heunen and J. P. Lemay. Tensor-restriction categories. *Theory and Applications of Categories*, 2021.

15 C. Heunen and M. L. Reyes. Frobenius structures over Hilbert C*-modules. *Communications in Mathematical Physics*, 361(2):787–824, 2018. doi:10.1007/s00220-018-3166-0.

16 C. Heunen and J. Vicary. *Categories for quantum theory: an introduction*. Oxford University Press, 2019. doi:10.1093/oso/9780198739623.001.0001.

17 M. Hyland, G. Plotkin, and J. Power. Combining effects: sum and tensor. *Theoretical Computer Science*, 357(1–3):70–99, 2006. doi:10.1016/j.tcs.2006.03.013.

18 B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69:73–106, 1994. doi:10.1016/0168-0072(94)90020-5.

19 C. B. Jay. Languages for monoidal categories. *Journal of Pure and Applied Algebra*, 59:61–85, 1989. doi:10.1016/0022-4049(89)90163-1.

20 A. Kock. Bilinearity and cartesian closed monads. *Mathematica Scandinavica*, 29:161–174, 1971. doi:10.7146/math.scand.a-11042.

21 A. Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23:113–120, 1972. doi:10.1007/BF01304852.

22 S. Lack. Composing PROPs. *Theory and Applications of Categories*, 13(13):147–163, 2004.

23 S. Lack and R. Street. The formal theory of monads II. *Journal of Pure and Applied Algebra*, 175(1–3):243–265, 2002. doi:10.1016/S0022-4049(02)00137-8.

24 F. W. Lawvere. The category of probabilistic mappings. 1962. URL: https://ncatlab.org/nlab/files/lawvereprobability1962.pdf.

25 T. Leinster. *Higher operads, higher categories*. Cambridge University Press, 2004. doi:10.1017/CBO9780511525896.

26 S. Milius, D. Pattinson, and L. Schröder. Generic trace semantics and graded monads. In *Conference on Algebra and Coalgebra in Computer Science*, volume 35 of *Leibniz International Proceedings in Informatics*, pages 253–269, 2015. doi:10.4230/LIPIcs.CALCO.2015.253.

27 R. Milner. *Communicating and mobile systems: the pi calculus*. Cambridge University Press, 1999.

28 E. Moggi. Computational lambda-calculus and monads. *Logic in Computer Science*, 1989. doi:10.1109/LICS.1989.39155.

29 G. Plotkin and J. Power. Notions of computation determine monads. *FoSSaCS*, pages 342–356, 2002. doi:10.1007/3-540-45931-6_24.

30 G. Plotkin and M. Pretnar. Handlers of algebraic effects. In *European Symposium on Programming*, volume 5502 of *Lecture Notes in Computer Science*, pages 80–94, 2009. doi:10.1007/978-3-642-00590-9_7.

31 G. D. Plotkin and A. J. Power. Computational effects and operations: an overview. In *Domains VI*, volume 73 of *Electronic Notes in Theoretical Computer Science*, pages 149–16, 2004. doi:10.1016/j.entcs.2004.08.008.

32 J. Power. Semantics for local computational effects. In *Mathematical Foundations of Programming Semantics*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 355–371, 2006. doi:10.1016/j.entcs.2006.04.018.

33 J. Power. *Models, Logics and Higher-Dimensional Categories: A Tribute to the Work of Mihály Makkai*, chapter Indexed Lawvere theories for local state, pages 213–229. American Mathematical Society, 2011.

34 S. Staton. Instances of computational effects: an algebraic perspective. In *Logic in Computer Science*, pages 519–528, 2013. doi:10.1109/LICS.2013.58.

35 R. Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2(2):149–168, 1972. doi:10.1016/0022-4049(72)90019-9.

36 A. Westerbaan. Quatum programs as Kleisli maps. In *Quantum Physics and Logic*, volume 237 of *Electronic Proceedings in Theoretical Computer Science*, pages 215–228, 2016. doi:10.4204/EPTCS.236.14.

37 G. Winskel and M. Nielsen. *Handbook of Logic in Computer Science*, volume 4, chapter Models for concurrency, pages 1–148. Oxford University Press, 1995.

38 M. Zwart. *On the Non-Compositionality of Monads via Distributive Laws*. PhD thesis, University of Oxford, 2020.

An Internal Language for Categories Enriched over Generalised Metric Spaces

Fredrik Dahlqvist
University College London, UK

Renato Neves
University of Minho, Braga, Portugal
INESC-TEC, Porto, Portugal

Abstract

Programs with a *continuous* state space or that interact with physical processes often require notions of equivalence going beyond the standard binary setting in which equivalence either holds or does not hold. In this paper we explore the idea of equivalence taking values in a quantale $\mathcal{V}$, which covers the cases of (in)equations and (ultra)metric equations among others.

Our main result is the introduction of a $\mathcal{V}$-*equational deductive system* for linear λ-calculus together with a proof that it is sound and complete (in fact, an internal language) for a class of enriched autonomous categories. In the case of inequations, we get an internal language for autonomous categories enriched over partial orders. In the case of (ultra)metric equations, we get an internal language for autonomous categories enriched over (ultra)metric spaces.

We use our results to obtain examples of inequational and metric equational systems for higher-order programs that contain real-time and probabilistic behaviour.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases λ-calculus, enriched category theory, quantale, equational theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.16

Related Version *Full Version*: `https://arxiv.org/abs/2105.08473` [10]

Funding This work was financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project POCI-01-0145-FEDER-030947. This work was also financed by the UK Research Institute in Verified Trustworthy Software Systems, via project "Quantitative Algebraic Reasoning for Hybrid Programs", and by the Leverhulme Project Grant "Verification of Machine Learning Algorithms".

Acknowledgements The authors are very grateful for the reviewer's incisive feedback.

1 Introduction

Programs frequently act over a *continuous* state space or interact with physical processes like time progression or the movement of a vehicle. Such features naturally call for notions of approximation and refinement integrated in different aspects of program equivalence. Our paper falls in this line of research. Specifically, our aim is to integrate notions of approximation and refinement into the *equational system* of linear λ-calculus [4, 28, 29].

The core idea that we explore in this paper is to have equations $t =_q s$ labelled by elements q of a quantale $\mathcal{V}$. This covers a wide range of situations, among which the cases of (in)equations [23, 2] and metric equations [30, 31]. The latter case is perhaps less known: it consists of equations $t =_\epsilon s$ labelled by a non-negative rational number ϵ which represents the 'maximum distance' that the two terms t and s can be from each other. In order to illustrate metric equations, consider a programming language with a (ground) type X and a signature

of operations $\Sigma = \{\mathtt{wait}_\mathtt{n} : X \to X \mid n \in \mathbb{N}\}$ that model time progression over computations of type X. Specifically, $\mathtt{wait}_\mathtt{n}(x)$ reads as "add a latency of n seconds to the computation x". In this context, the following axioms involving metric equations arise naturally:

$$\mathtt{wait}_\mathtt{0}(x) =_0 x \qquad \mathtt{wait}_\mathtt{n}(\mathtt{wait}_\mathtt{m}(x)) =_0 \mathtt{wait}_{\mathtt{n}+\mathtt{m}}(x) \qquad \frac{\epsilon = |m-n|}{\mathtt{wait}_\mathtt{n}(x) =_\epsilon \mathtt{wait}_\mathtt{m}(x)} \qquad (1)$$

An equation $t =_0 s$ states that the terms t and s are exactly the same and equations $t =_\epsilon s$ state that t and s differ by *at most* ϵ seconds in their execution time.

Contributions. In this paper we introduce an equational deductive system for linear λ-calculus in which equations are labelled by elements of a quantale $\mathcal{V}$. By using key features of a quantale's structure, we show that this deductive system is *sound and complete* for a class of enriched symmetric monoidal closed categories (*i.e.* enriched autonomous categories). In particular, if we fix $\mathcal{V}$ to be the Boolean quantale this class of categories consists of autonomous categories enriched over partial orders. If we fix $\mathcal{V}$ to be the (ultra)metric quantale, this class of categories consists of autonomous categories enriched over (ultra)metric spaces. The aforementioned example of wait calls fits in the setting in which $\mathcal{V}$ is the metric quantale. Our result provides this example with a sound and complete metric equational system, where the models are all those autonomous categories enriched over metric spaces that can soundly interpret the axioms of wait calls (1).

The next contribution of our paper falls in one of the major topics of categorical logic: to establish *logical descriptions* of certain classes of categories. A famous result of this kind is the correspondence between λ-calculus and Cartesian closed categories which states that the former is the *internal language* of the latter [24] – such a correspondence allows to study Cartesian closed categories by means of logical tools. An analogous result is presented in [28, 29] for linear λ-calculus and symmetric monoidal closed (*i.e.* autonomous) categories. We show that linear λ-calculus equipped with a $\mathcal{V}$-equational system is the internal language of autonomous categories enriched over 'generalised metric spaces'.

Outline. Section 2 recalls linear λ-calculus, its equational system, and the famous correspondence to autonomous categories, via soundness, completeness, and internal language theorems. The contents of this section are slight adaptations of results presented in [28, 4], the main difference being that we forbid the exchange rule to be explicitly part of linear λ-calculus (instead it is only admissible). This choice is important to ensure that judgements in the calculus have *unique* derivations, which allows us to refer to their interpretations unambiguously [37]. Section 3 presents the main contributions of this paper. It walks a path analogous to Section 2, but now in the setting of $\mathcal{V}$-equations (*i.e.* equations labelled by elements of a quantale $\mathcal{V}$). As we will see, the semantic counterpart of moving from equations to $\mathcal{V}$-equations is to move from categories to categories enriched over $\mathcal{V}$-*categories*. The latter, often regarded as generalised metric spaces, are central entities in a fruitful area of enriched category theory that aims to treat uniformly different kinds of "structured sets", such as partial orders, fuzzy partial orders, and (ultra)metric spaces [25, 38, 39]. Our results are applicable to all these examples. Section 4 presents some examples of $\mathcal{V}$-equational axioms and corresponding models. Specifically, we will revisit the axioms of wait calls (1) and consider an inequational variant. Then we will study a metric axiom for probabilistic programs and show that the category of Banach spaces and short linear maps is a model for the resulting metric theory. We will additionally use this example to illustrate how our deductive system allows to compute an approximate distance between two probabilistic

programs easily as opposed to computing an exact distance "semantically" which tends to involve quite complex operators. Finally, Section 5 establishes a functorial connection between our results and previous well-known semantics for linear logic [11, 29], and concludes with a brief exposition of future work. We assume knowledge of λ-calculus and category theory [28, 29, 24, 27]. Proofs omitted in the text are available in the extended version of this paper [10].

Related work. Several approaches to incorporating quantitative information to programming languages have been explored in the literature. Closest to this work are various approaches targeted at λ-calculi. In [7, 8] a notion of distance called *context distance* is developed, first for an affine, then for a more general λ-calculus, with probabilistic programs as the main motivation. [14] considers a notion of quantale-valued *applicative (bi)similarity*, an operational *coinductive* technique used for showing contextual equivalence between two programs. Recently, [34] presented several Cartesian closed categories of generalised metric spaces that provide a quantitative semantics to simply-typed λ-calculus based on a generalisation of logical relations. None of these examples reason about distances in a quantitative equational system, and in this respect our work is closer to the metric universal algebra developed in [30, 31].

A different approach consists in encoding quantitative information via a type system. In particular, graded (modal) types [16, 13, 32] have found applications in *e.g.* differential privacy [35] and information flow [1]. This approach is to some extent orthogonal to ours as it mainly aims to model coeffects, whilst we aim to reason about the intrinsic quantitative nature of λ-terms acting *e.g.* on continuous or ordered spaces.

Quantum programs provide an interesting example of intrinsically quantitative programs, by which we mean that the metric structure on quantum states does not arise from (co)effects. Recently, [19] showed how the issue of noise in a quantum while-language can be handled by developing a *deductive system* to determine how similar a quantum program is from its idealised, noise-free version; an approach very much in the spirit of this work.

2 An internal language for autonomous categories

In this section we briefly recall linear λ-calculus, which can be regarded as a term assignment system for the exponential free, multiplicative fragment of *intuitionistic linear logic*. Then we recall that it is sound and complete w.r.t. autonomous categories, and also that it is an internal language for such categories. We mention only what is needed to present our results, the interested reader will find a more detailed exposition in [28, 4, 29]. Let us start by fixing a *class* G of ground types. The grammar of types for linear λ-calculus is given by:

$$\mathbb{A} ::= X \in G \mid \mathbb{I} \mid \mathbb{A} \otimes \mathbb{A} \mid \mathbb{A} \multimap \mathbb{A}$$

We also fix a class Σ of sorted operation symbols $f : \mathbb{A}_1, \ldots, \mathbb{A}_n \to \mathbb{A}$ with $n \geq 1$. As usual, we use Greek letters $\Gamma, \Delta, E, \ldots$ to denote *typing contexts*, *i.e.* lists $x_1 : \mathbb{A}_1, \ldots, x_n : \mathbb{A}_n$ of typed variables such that each variable x_i occurs at most once in $x_1, \ldots, x_n$.

We will use the notion of a shuffle for building a linear typing system such that the *exchange rule* is admissible and each judgement $\Gamma \rhd v : \mathbb{A}$ (details about these below) has a *unique derivation* – this will allow us to refer to a judgement's denotation $[\![\Gamma \rhd v : \mathbb{A}]\!]$ unambiguously. By shuffle we mean a permutation of typed variables in a context sequence $\Gamma_1, \ldots, \Gamma_n$ such that for all $i \leq n$ the relative order of the variables in Γ_i is preserved [37]. For example, if $\Gamma_1 = x : \mathbb{A}, y : \mathbb{B}$ and $\Gamma_2 = z : \mathbb{C}$ then $z : \mathbb{C}, x : \mathbb{A}, y : \mathbb{B}$ is a shuffle but

$y : \mathbb{B}, x : \mathbb{A}, z : \mathbb{C}$ is *not*, because we changed the order in which x and y appear in Γ_1. As explained in [37] (and also in the proof of Lemma 1), such a restriction on relative orders is crucial for judgements having unique derivations. We denote by $\mathrm{Sf}(\Gamma_1; \ldots; \Gamma_n)$ the set of shuffles on $\Gamma_1, \ldots, \Gamma_n$.

The term formation rules of linear λ-calculus are listed in Fig. 1. They correspond to the natural deduction rules of the exponential free, multiplicative fragment of intuitionistic linear logic.

▶ **Lemma 1.** *All judgements $\Gamma \rhd v : \mathbb{A}$ have a unique derivation.*

Substitution is defined in the expected way, and the following result is standard.

▶ **Lemma 2** (Exchange and Substitution)**.** *For every judgement $\Gamma, x : \mathbb{A}, y : \mathbb{B}, \Delta \rhd v : \mathbb{C}$ we can derive $\Gamma, y : \mathbb{B}, x : \mathbb{A}, \Delta \rhd v : \mathbb{C}$. For all judgements $\Gamma, x : \mathbb{A} \rhd v : \mathbb{B}$ and $\Delta \rhd w : \mathbb{A}$ we can derive $\Gamma, \Delta \rhd v[w/x] : \mathbb{B}$.*

We now recall the interpretation of judgements $\Gamma \rhd v : \mathbb{A}$ in a symmetric monoidal closed (autonomous) category C. But before proceeding with this description, let us fix notation for some of the constructions available in autonomous categories. For all C-objects X, Y, Z, $\mathsf{sw} : X \otimes Y \to Y \otimes X$ denotes the symmetry morphism, $\lambda : \mathbb{I} \otimes X \to X$ the left unitor, $\mathrm{app} : (X \multimap Y) \otimes X \to Y$ the application, and $\alpha : X \otimes (Y \otimes Z) \to (X \otimes Y) \otimes Z$ the left associator. Moreover for all C-morphisms $f : X \otimes Y \to Z$ we denote the corresponding curried version (right transpose) by $\overline{f} : X \to (Y \multimap Z)$.

For all ground types $X \in G$ we postulate an interpretation $[\![X]\!]$ as a C-object. Types are then interpreted by induction over the type structure of linear λ-calculus, using the tensor $\otimes$ and exponential $\multimap$ constructs of autonomous categories. Given a non-empty context $\Gamma = \Gamma', x : \mathbb{A}$, its interpretation is defined by $[\![\Gamma', x : \mathbb{A}]\!] = [\![\Gamma']\!] \otimes [\![\mathbb{A}]\!]$ if Γ' is non-empty and $[\![\Gamma', x : \mathbb{A}]\!] = [\![\mathbb{A}]\!]$ otherwise. The empty context $-$ is interpreted as $[\![-]\!] = \mathbb{I}$ where $\mathbb{I}$ is the unit of $\otimes$ in C. To keep notation simple, given $X_1, \ldots, X_n \in \mathsf{C}$ we write $X_1 \otimes \cdots \otimes X_n$ for the n-tensor $(\ldots (X_1 \otimes X_2) \otimes \ldots) \otimes X_n$, and similarly for C-morphisms.

We will also need some 'housekeeping' morphisms to handle interactions between context interpretation and the autonomous structure of C. Specifically, given contexts $\Gamma_1, \ldots, \Gamma_n$ we denote by $\mathsf{sp}_{\Gamma_1;\ldots;\Gamma_n} : [\![\Gamma_1, \ldots, \Gamma_n]\!] \to [\![\Gamma_1]\!] \otimes \cdots \otimes [\![\Gamma_n]\!]$ the morphism that splits $[\![\Gamma_1, \ldots, \Gamma_n]\!]$ into $[\![\Gamma_1]\!] \otimes \cdots \otimes [\![\Gamma_n]\!]$, and by $\mathsf{jn}_{\Gamma_1;\ldots;\Gamma_n}$ the corresponding inverse. Given a context $\Gamma, x : \mathbb{A}, y : \mathbb{B}, \Delta$ we denote by $\mathsf{exch}_{\Gamma, \underline{x:\mathbb{A}, y:\mathbb{B}}, \Delta} : [\![\Gamma, x : \mathbb{A}, y : \mathbb{B}, \Delta]\!] \to [\![\Gamma, y : \mathbb{B}, x : \mathbb{A}, \Delta]\!]$ the morphism corresponding to the permutation of the variable $x : \mathbb{A}$ with $y : \mathbb{B}$. Whenever convenient we will drop variable names in the subscripts of sp, jn, and exch. For a context $E \in \mathrm{Sf}(\Gamma_1, \ldots, \Gamma_n)$ the morphism $\mathsf{sh}_E : [\![E]\!] \to [\![\Gamma_1, \ldots, \Gamma_n]\!]$ denotes the corresponding shuffling morphism.

For every operation symbol $f : \mathbb{A}_1, \ldots, \mathbb{A}_n \to \mathbb{A}$ in Σ we postulate an interpretation $[\![f]\!] : [\![\mathbb{A}_1]\!] \otimes \cdots \otimes [\![\mathbb{A}_n]\!] \to [\![\mathbb{A}]\!]$ as a C-morphism. The interpretation of judgements is defined by induction over the structure of judgement derivation according to the rules in Fig. 2.

As detailed in [4, 28, 29], linear λ-calculus comes equipped with a class of equations (Fig. 3), specifically *equations-in-context* $\Gamma \rhd v = w : \mathbb{A}$, that corresponds to the axiomatics of autonomous categories. As usual, we omit the context and typing information of the equations in Fig. 3, which can be reconstructed in the usual way.

▶ **Theorem 3.** *The equations presented in Fig. 3 are sound w.r.t. judgement interpretation. Specifically if $\Gamma \rhd v = w : \mathbb{A}$ is one of the equations in Fig. 3 then $[\![\Gamma \rhd v : \mathbb{A}]\!] = [\![\Gamma \rhd w : \mathbb{A}]\!]$.*

▶ **Definition 4** (Linear λ-theories)**.** *Consider a tuple (G, Σ) consisting of a class G of ground types and a class Σ of sorted operation symbols. A linear λ-theory $((G, \Sigma), Ax)$ is a triple such that Ax is a class of equations-in-context over linear λ-terms built from (G, Σ).*

$$\frac{\Gamma_i \rhd v_i : \mathbb{A}_i \quad f : \mathbb{A}_1, \ldots, \mathbb{A}_n \to \mathbb{A} \in \Sigma \quad E \in \mathrm{Sf}(\Gamma_1; \ldots; \Gamma_n)}{E \rhd f(v_1, \ldots, v_n) : \mathbb{A}}\ (\mathbf{ax}) \qquad \frac{}{x : \mathbb{A} \rhd x : \mathbb{A}}\ (\mathbf{hyp})$$

$$\frac{}{- \rhd * : \mathbb{I}}\ (\mathbb{I}_\mathbf{i}) \qquad \frac{\Gamma \rhd v : \mathbb{I} \quad \Delta \rhd w : \mathbb{A} \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{E \rhd v \ \mathtt{to}\ * .\ w : \mathbb{A}}\ (\mathbb{I}_\mathbf{e})$$

$$\frac{\Gamma \rhd v : \mathbb{A} \quad \Delta \rhd w : \mathbb{B} \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{E \rhd v \otimes w : \mathbb{A} \otimes \mathbb{B}}\ (\otimes_\mathbf{i})$$

$$\frac{\Gamma \rhd v : \mathbb{A} \otimes \mathbb{B} \quad \Delta, x : \mathbb{A}, y : \mathbb{B} \rhd w : \mathbb{C} \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{E \rhd \mathtt{pm}\ v\ \mathtt{to}\ x \otimes y .\ w : \mathbb{C}}\ (\otimes_\mathbf{e})$$

$$\frac{\Gamma, x : \mathbb{A} \rhd v : \mathbb{B}}{\Gamma \rhd \lambda x : \mathbb{A}. v : \mathbb{A} \multimap \mathbb{B}}\ (\multimap_\mathbf{i}) \qquad \frac{\Gamma \rhd v : \mathbb{A} \multimap \mathbb{B} \quad \Delta \rhd w : \mathbb{A} \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{E \rhd v\, w : \mathbb{B}}\ (\multimap_\mathbf{e})$$

Figure 1 Term formation rules for linear λ-calculus.

$$\frac{[\![\Gamma_i \rhd v_i : \mathbb{A}_i]\!] = m_i \quad f : \mathbb{A}_1, \ldots, \mathbb{A}_n \to \mathbb{A} \in \Sigma \quad E \in \mathrm{Sf}(\Gamma_1 \ldots \Gamma_n)}{[\![E \rhd f(v_1, \ldots, v_n) : \mathbb{A}]\!] = [\![f]\!] \cdot (m_1 \otimes \cdots \otimes m_n) \cdot \mathsf{sp}_{\Gamma_1;\ldots;\Gamma_n} \cdot \mathsf{sh}_E} \qquad \frac{}{[\![x : \mathbb{A} \rhd x : \mathbb{A}]\!] = \mathsf{id}_{[\![\mathbb{A}]\!]}}$$

$$\frac{}{[\![- \rhd * : \mathbb{I}]\!] = \mathsf{id}_{[\![\mathbb{I}]\!]}} \qquad \frac{[\![\Gamma \rhd v : \mathbb{I}]\!] = m \quad [\![\Delta \rhd w : \mathbb{A}]\!] = n \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{[\![E \rhd v \ \mathtt{to}\ * .\ w : \mathbb{A}]\!] = n \cdot \lambda \cdot (m \otimes \mathsf{id}) \cdot \mathsf{sp}_{\Gamma;\Delta} \cdot \mathsf{sh}_E}$$

$$\frac{[\![\Gamma \rhd v : \mathbb{A}]\!] = m \quad [\![\Delta \rhd w : \mathbb{B}]\!] = n \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{[\![E \rhd v \otimes w : \mathbb{A} \otimes \mathbb{B}]\!] = (m \otimes n) \cdot \mathsf{sp}_{\Gamma;\Delta} \cdot \mathsf{sh}_E} \qquad \frac{[\![\Gamma, x : \mathbb{A} \rhd v : \mathbb{B}]\!] = m}{[\![\Gamma \rhd \lambda x : \mathbb{A}. v : \mathbb{A} \multimap \mathbb{B}]\!] = \overline{(m \cdot \mathsf{jn}_{\Gamma;\mathbb{A}})}}$$

$$\frac{[\![\Gamma \rhd v : \mathbb{A} \otimes \mathbb{B}]\!] = m \quad [\![\Delta, x : \mathbb{A}, y : \mathbb{B} \rhd w : \mathbb{C}]\!] = n \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{[\![E \rhd \mathtt{pm}\ v\ \mathtt{to}\ x \otimes y .\ w : \mathbb{C}]\!] = n \cdot \mathsf{jn}_{\Delta;\mathbb{A};\mathbb{B}} \cdot \alpha \cdot \mathsf{sw} \cdot (m \otimes \mathsf{id}) \cdot \mathsf{sp}_{\Gamma;\Delta} \cdot \mathsf{sh}_E}$$

$$\frac{[\![\Gamma \rhd v : \mathbb{A} \multimap \mathbb{B}]\!] = m \quad [\![\Delta \rhd w : \mathbb{A}]\!] = n \quad E \in \mathrm{Sf}(\Gamma; \Delta)}{[\![E \rhd v\, w : \mathbb{B}]\!] = \mathrm{app} \cdot (m \otimes n) \cdot \mathsf{sp}_{\Gamma;\Delta} \cdot \mathsf{sh}_E}$$

Figure 2 Judgement interpretation on an autonomous category C.

$$\begin{aligned} \mathtt{pm}\ v \otimes w\ \mathtt{to}\ x \otimes y .\ u &= u[v/x, w/y] \\ \mathtt{pm}\ v\ \mathtt{to}\ x \otimes y .\ u[x \otimes y/z] &= u[v/z] \\ * \ \mathtt{to}\ * .\ v &= v \\ v \ \mathtt{to}\ * .\ w[*/z] &= w[v/z] \end{aligned}$$

(a) Monoidal structure.

$$\begin{aligned} (\lambda x : \mathbb{A}.\ v)\ w &= v[w/x] \\ \lambda x : \mathbb{A}.(v\ x) &= v \end{aligned}$$

(b) Higher-order structure.

$$\begin{aligned} u[v \ \mathtt{to} * .\ w/z] &= v \ \mathtt{to}\ * .\ u[w/z] \\ u[\mathtt{pm}\ v\ \mathtt{to}\ x \otimes y .\ w/z] &= \mathtt{pm}\ v\ \mathtt{to}\ x \otimes y .\ u[w/z] \end{aligned}$$

(c) Commuting conversions.

Figure 3 Equations corresponding to the axiomatics of autonomous categories.

The elements of Ax are called axioms (of the theory). Let $Th(Ax)$ be the smallest congruence that contains Ax, the equations listed in Fig. 3, and that is closed under the exchange and substitution rules. We call the elements of $Th(Ax)$ theorems (of the theory).

▶ **Definition 5** (Models of linear λ-theories). *Consider a linear λ-theory $((G, \Sigma), Ax)$ and an autonomous category C. Suppose that for each $X \in G$ we have an interpretation $[\![X]\!]$ that is a C-object and analogously for the operation symbols. This interpretation structure is a* model *of the theory if all axioms are satisfied by the interpretation.*

Next let us turn our attention to the correspondence between linear λ-calculus and autonomous categories, established via soundness, completeness, and internal language theorems. Despite the proofs of such theorems already being detailed in [28, 4, 29], we decided to briefly sketch them below to render the presentation of some of our own results self-contained.

▶ **Theorem 6** (Soundness & Completeness). *Consider a linear λ-theory T. An equation $\Gamma \rhd v = w : \mathbb{A}$ is a theorem of T iff it is satisfied by all models of the theory.*

Proof sketch. Soundness follows by induction over the rules that define $Th(Ax)$ (Definition 4) and by Theorem 3. Completeness is based on the idea of a Lindenbaum-Tarski algebra: it follows from building the *syntactic category* $\mathrm{Syn}(T)$ of T (also known as term model), showing that it possesses an autonomous structure and also that equality $[\![\Gamma \rhd v : \mathbb{A}]\!] = [\![\Gamma \rhd w : \mathbb{A}]\!]$ in the syntactic category is equivalent to provability $\Gamma \rhd v = w : \mathbb{A}$ in the theory.

The syntactic category of T has as objects the types of T and as morphisms $\mathbb{A} \to \mathbb{B}$ the equivalence classes (w.r.t. provability) of terms v for which we can derive $x : \mathbb{A} \rhd v : \mathbb{B}$. ◀

Next let us focus on the topic of internal languages, for which the following result is quite useful.

▶ **Theorem 7.** *Consider a linear λ-theory T and a model of T on an autonomous category C. The model induces a functor $F : \mathrm{Syn}(T) \to \mathsf{C}$ that (strictly) preserves the autonomous structure.*

Proof sketch. Consider a model of T on a category C. Then for any judgement $x : \mathbb{A} \rhd v : \mathbb{B}$, the induced functor F sends the equivalence class $[v]$ into $[\![x : \mathbb{A} \rhd v : \mathbb{B}]\!]$. ◀

An autonomous category C induces a linear λ-theory $\mathrm{Lang}(\mathsf{C})$ whose ground types $X \in G$ are the objects of C and whose signature Σ of operation symbols consists of all the morphisms in C plus certain isomorphisms that we describe in (2). The axioms of $\mathrm{Lang}(\mathsf{C})$ are all the equations satisfied by the obvious interpretation in C. In order to explicitly distinguish the autonomous structure of C from the type structure of $\mathrm{Lang}(\mathsf{C})$ let us denote the tensor of C by $\hat{\otimes}$, the unit by $\hat{\mathbb{I}}$, and the exponential by $\widehat{\multimap}$. Consider then the following map on types:

$$i(\mathbb{I}) = \hat{\mathbb{I}} \qquad i(X) = X \qquad i(\mathbb{A} \otimes \mathbb{B}) = i(\mathbb{A}) \mathbin{\hat{\otimes}} i(\mathbb{B}) \qquad i(\mathbb{A} \multimap \mathbb{B}) = i(\mathbb{A}) \mathbin{\widehat{\multimap}} i(\mathbb{B}) \tag{2}$$

For each type $\mathbb{A}$ we add an isomorphism $\mathbb{A} \simeq i(\mathbb{A})$ to the theory $\mathrm{Lang}(\mathsf{C})$.

▶ **Theorem 8** (Internal language). *For every autonomous category C there exists an equivalence of categories $\mathrm{Syn}(\mathrm{Lang}(\mathsf{C})) \simeq \mathsf{C}$.*

Proof sketch. By construction, we have an interpretation of $\mathrm{Lang}(\mathsf{C})$ in C which behaves as the identity for operation symbols and ground types. This interpretation is a model of $\mathrm{Lang}(\mathsf{C})$ on C and by Theorem 7 we obtain a functor $\mathrm{Syn}(\mathrm{Lang}(\mathsf{C})) \to \mathsf{C}$. The functor in the opposite direction behaves as the identity on objects and sends a C-morphism f into $[f(x)]$. The equivalence of categories is then shown by using the aforementioned isomorphisms which connect the type constructors of $\mathrm{Lang}(\mathsf{C})$ with the autonomous structure of C. ◀

3 From equations to $\mathcal{V}$-equations

We now extend the results of the previous section to the setting of $\mathcal{V}$-equations.

3.1 A $\mathcal{V}$-equational deductive system

Let $\mathcal{V}$ denote a commutative and unital quantale, $\otimes : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ the corresponding binary operation, and k the corresponding unit [33]. As mentioned in the introduction, $\mathcal{V}$ induces the notion of a $\mathcal{V}$-equation, *i.e.* an equation $t =_q s$ labelled by an element q of $\mathcal{V}$. This subsection explores this concept by introducing a $\mathcal{V}$-equational deductive system for linear λ-calculus and a notion of a linear $\mathcal{V}\lambda$-theory.

Let us start by recalling two definitions concerning ordered structures [15, 17] and then explain their relevance to our work.

▶ **Definition 9.** *Consider a complete lattice L. For every $x, y \in L$ we say that y is* way-below *x (in symbols, $y \ll x$) if for every subset $X \subseteq L$ whenever $x \leq \bigvee X$ there exists a* finite *subset $A \subseteq X$ such that $y \leq \bigvee A$. The lattice L is called* continuous *iff for every $x \in L$,*

$$x = \bigvee \{y \mid y \in L \text{ and } y \ll x\}$$

▶ **Definition 10.** *Let L be a complete lattice. A* basis *B of L is a subset $B \subseteq L$ such that for every $x \in L$ the set $B \cap \{y \mid y \in L \text{ and } y \ll x\}$ is directed and has x as the least upper bound.*

From now on we assume that the underlying lattice of $\mathcal{V}$ is continuous and has a basis B which is closed under finite joins, the multiplication of the quantale $\otimes$ and contains the unit k. These assumptions will allow us to work *only* with a specified subset of $\mathcal{V}$-equations chosen *e.g.* for computational reasons, such as the *finite* representation of values $q \in \mathcal{V}$.

▶ **Example 11.** The Boolean quantale $((\{0 \leq 1\}, \vee), \otimes := \wedge)$ is *finite* and thus continuous [15]. Since it is continuous, $\{0, 1\}$ itself is a basis for the quantale that satisfies the conditions above. For the Gödel t-norm [12] $(([0, 1], \vee), \otimes := \wedge)$, the way-below relation is the strictly-less relation $<$ with the exception that $0 < 0$. A basis for the underlying lattice that satisfies the conditions above is the set $\mathbb{Q} \cap [0, 1]$. Note that, unlike real numbers, rationals numbers always have a finite representation. For the metric quantale (also known as Lawvere quantale) $(([0, \infty], \wedge), \otimes := +)$, the way-below relation corresponds to the *strictly greater* relation with $\infty > \infty$, and a basis for the underlying lattice that satisfies the conditions above is the set of extended non-negative rational numbers. The latter also serves as basis for the ultrametric quantale $(([0, \infty], \wedge), \otimes := \max)$.

We also assume that $\mathcal{V}$ is *integral*, *i.e.* that the unit k is the top element of $\mathcal{V}$. This will allow us to establish a smoother theory of $\mathcal{V}$-equations, whilst still covering *e.g.* all the examples above. This assumption is common in quantale theory [39].

Recall the term formation rules of linear λ-calculus from Fig. 1. A $\mathcal{V}$-equation-in-context is an expression $\Gamma \rhd v =_q w : \mathbb{A}$ with $q \in B$ (the basis of $\mathcal{V}$), $\Gamma \rhd v : \mathbb{A}$ and $\Gamma \rhd w : \mathbb{A}$. Let $\top$ be the top element in $\mathcal{V}$. An equation-in-context $\Gamma \rhd v = w : \mathbb{A}$ now denotes the particular case in which both $\Gamma \rhd v =_\top w : \mathbb{A}$ and $\Gamma \rhd w =_\top v : \mathbb{A}$. For the case of the Boolean quantale, $\mathcal{V}$-equations are labelled by $\{0, 1\}$. We will see that $\Gamma \rhd v =_1 w : \mathbb{A}$ can be treated as an inequation $\Gamma \rhd v \leq w : \mathbb{A}$, whilst $\Gamma \rhd v =_0 w : \mathbb{A}$ corresponds to a trivial $\mathcal{V}$-equation, *i.e.* a $\mathcal{V}$-equation that always holds. For the Gödel t-norm, we can choose $\mathbb{Q} \cap [0, 1]$ as basis and then obtain what we call *fuzzy inequations*. For the metric quantale, we can choose the set of extended non-negative rational numbers as basis and then obtain *metric equations* in

the spirit of [30, 31]. Similarly, by choosing the ultrametric quantale $(([0,\infty],\wedge),\otimes := \max)$ with the set of extended non-negative rational numbers as basis we obtain what we call *ultrametric equations*.

▶ **Definition 12** (Linear $\mathcal{V}\lambda$-theories). *Consider a tuple (G,Σ) consisting of a class G of ground types and a class of sorted operation symbols $f : \mathbb{A}_1,\dots,\mathbb{A}_n \to \mathbb{A}$ with $n \geq 1$. A linear $\mathcal{V}\lambda$-theory $((G,\Sigma),Ax)$ is a tuple such that Ax is a class of $\mathcal{V}$-equations-in-context over linear λ-terms built from (G,Σ).*

$$\frac{}{v =_\top v}\ \textbf{(refl)} \qquad \frac{v =_q w \qquad w =_r u}{v =_{q\otimes r} u}\ \textbf{(trans)} \qquad \frac{v =_q w \qquad r \leq q}{v =_r w}\ \textbf{(weak)}$$

$$\frac{\forall r \ll q.\ v =_r w}{v =_q w}\ \textbf{(arch)} \qquad \frac{\forall i \leq n.\ v =_{q_i} w}{v =_{\vee q_i} w}\ \textbf{(join)}$$

$$\frac{\forall i \leq n.\ v_i =_{q_i} w_i}{f(v_1,\dots,v_n) =_{\otimes q_i} f(w_1,\dots,w_n)} \qquad \frac{v =_q w \quad v' =_r w'}{v \otimes v' =_{q\otimes r} w \otimes w'}$$

$$\frac{v =_q w \qquad v' =_r w'}{\texttt{pm}\ v\ \texttt{to}\ x\otimes y.\ v' =_{q\otimes r} \texttt{pm}\ w\ \texttt{to}\ x\otimes y.\ w'}$$

$$\frac{v =_q w \qquad v' =_r w'}{v\ \texttt{to}\ *.\ v' =_{q\otimes r} w\ \texttt{to}\ *.\ w'} \qquad \frac{v =_q w}{\lambda x:\mathbb{A}.\ v =_q \lambda x:\mathbb{A}.\ w} \qquad \frac{v =_q w \quad v' =_r w'}{v\,v' =_{q\otimes r} w\,w'}$$

$$\frac{\Gamma \rhd v =_q w : \mathbb{A} \qquad \Delta \in perm(\Gamma)}{\Delta \rhd v =_q w : \mathbb{A}} \qquad \frac{v =_q w \qquad v' =_r w'}{v[v'/x] =_{q\otimes r} w[w'/x]}$$

Figure 4 $\mathcal{V}$-congruence rules.

The elements of Ax are the axioms of the theory. Let $Th(Ax)$ be the smallest class that contains Ax and that is closed under the rules of Fig. 3 and of Fig. 4 (as usual we omit the context and typing information). The elements of $Th(Ax)$ are the theorems of the theory.

Let us examine the rules in Fig. 4 in more detail. They can be seen as a generalisation of the notion of a congruence. The rules **(refl)** and **(trans)** are a generalisation of equality's reflexivity and transitivity. Rule **(weak)** encodes the principle that the higher the label in the $\mathcal{V}$-equation, the "tighter" is the relation between the two terms in the $\mathcal{V}$-equation. In other words, $v =_r w$ is subsumed by $v =_q w$, for $r \leq q$. This can be seen clearly *e.g.* with the metric quantale by reading $v =_q w$ as "the terms v and w are *at most* at distance q from each other" (recall that in the metric quantale the usual order is reversed, *i.e.* $\leq := \geq_{[0,\infty]}$). **(arch)** is essentially a generalisation of the Archimedean rule in [30, 31]. It says that if $v =_r w$ for all *approximations* r of q then it is also the case that $v =_q w$. **(join)** says that deductions are closed under finite joins, and in particular it is always the case that $v =_\bot w$. All other rules correspond to a generalisation of *compatibility* to a $\mathcal{V}$-equational setting.

The reader may have noticed that the rules in Fig. 4 do not contain a $\mathcal{V}$-generalisation of symmetry w.r.t. standard equality. Such a generalisation would be:

$$\frac{v =_q w}{w =_q v}$$

This rule is not present in Fig. 4 because in some quantales $\mathcal{V}$ it forces too many $\mathcal{V}$-equations. For example, in the Boolean quantale the condition $v \leq w$ would automatically entail $w \leq v$

(due to symmetry); in fact, for this particular case symmetry forces the notion of inequation to collapse into the classical notion of equation. On the other hand, symmetry is desirable in the (ultra)metric case because (ultra)metrics need to respect the symmetry equation [17].

▶ **Definition 13** (Symmetric linear $\mathcal{V}\lambda$-theories). *A symmetric linear $\mathcal{V}\lambda$-theory is a linear $\mathcal{V}\lambda$-theory whose set of theorems is closed under symmetry.*

In the paper's extended version [10] we further explore how specific families of quantales are reflected in the $\mathcal{V}$-equational system here introduced, and briefly compare the latter to metric algebra [30, 31].

3.2 Semantics of $\mathcal{V}$-equations

In this subsection we set the necessary background for presenting a sound and complete class of models for (symmetric) linear $\mathcal{V}\lambda$-theories. We start by recalling basics concepts of $\mathcal{V}$-categories, which are central in a field initiated by Lawvere in [25] and can be intuitively seen as generalised metric spaces [38, 18, 39]. As we will see, $\mathcal{V}$-categories provide structure to suitably interpret $\mathcal{V}$-equations.

▶ **Definition 14.** *A (small) $\mathcal{V}$-category is a pair (X, a) where X is a class (set) and $a : X \times X \to \mathcal{V}$ is a function that satisfies:*

$$k \leq a(x,x) \qquad \text{and} \qquad a(x,y) \otimes a(y,z) \leq a(x,z) \qquad\qquad (x,y,z \in X)$$

For two $\mathcal{V}$-categories (X, a) and (Y, b), a $\mathcal{V}$-functor $f : (X, a) \to (Y, b)$ is a function $f : X \to Y$ that satisfies the inequality $a(x, y) \leq b(f(x), f(y))$ for all $x, y \in X$.

Small $\mathcal{V}$-categories and $\mathcal{V}$-functors form a category which we denote by $\mathcal{V}$-Cat. A $\mathcal{V}$-category (X, a) is called *symmetric* if $a(x, y) = a(y, x)$ for all $x, y \in X$. We denote by $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$ the full subcategory of $\mathcal{V}$-Cat whose objects are symmetric. Every $\mathcal{V}$-category carries a natural order defined by $x \leq y$ whenever $k \leq a(x, y)$. A $\mathcal{V}$-category is called *separated* if its natural order is anti-symmetric. We denote by $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$ the full subcategory of $\mathcal{V}$-Cat whose objects are separated.

▶ **Example 15.** For $\mathcal{V}$ the Boolean quantale, $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$ is the category Pos of partially ordered sets and monotone maps; $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ is simply the category Set of sets and functions. For $\mathcal{V}$ the metric quantale, $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ is the category Met of extended metric spaces and non-expansive maps. In what follows we omit the qualifier "extended" in "extended (ultra)metric spaces". For $\mathcal{V}$ the ultrametric quantale, $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ is the category of ultrametric spaces and non-expansive maps.

The inclusion functor $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}} \hookrightarrow \mathcal{V}$-$\mathsf{Cat}$ has a left adjoint [18]. It is constructed first by defining the equivalence relation $x \sim y$ whenever $x \leq y$ and $y \leq x$ (for $\leq$ the natural order introduced earlier). Then this relation induces the separated $\mathcal{V}$-category $(X/_\sim, \tilde{a})$ where $\tilde{a}$ is defined as $\tilde{a}([x], [y]) = a(x, y)$ for every $[x], [y] \in X/_\sim$. The left adjoint of the inclusion functor $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}} \hookrightarrow \mathcal{V}$-$\mathsf{Cat}$ sends every $\mathcal{V}$-category (X, a) to $(X/_\sim, \tilde{a})$. This quotienting construct preserves symmetry, and therefore we automatically obtain the following result.

▶ **Theorem 16.** *The inclusion functor $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym,sep}} \hookrightarrow \mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$ has a left adjoint.*

Next, we recall notions of enriched category theory [20] instantiated into the setting of *autonomous categories enriched over $\mathcal{V}$-categories*. We will use the enriched structure to give semantics to $\mathcal{V}$-equations between linear λ-terms. First, note that every category $\mathcal{V}$-Cat

is autonomous with the tensor $(X,a) \otimes (Y,b) := (X \times Y, a \otimes b)$ where $a \otimes b$ is defined as $(a \otimes b)((x,y),(x',y')) = a(x,x') \otimes b(y,y')$ and the set of $\mathcal{V}$-functors $\mathcal{V}\text{-}\mathsf{Cat}((X,a),(Y,b))$ equipped with the map $(f,g) \mapsto \bigwedge_{x \in X} b(f(x),g(x))$.

▶ **Theorem 17.** *The categories $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$, $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sep}}$, and $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym,sep}}$ inherit the autonomous structure of $\mathcal{V}\text{-}\mathsf{Cat}$ whenever $\mathcal{V}$ is integral.*

Since we assume that $\mathcal{V}$ is integral, this last theorem allows us to formally define the notion of categories enriched over $\mathcal{V}$-categories using [20].

▶ **Definition 18.** *A category C is $\mathcal{V}\text{-}\mathsf{Cat}$-enriched (or simply, a $\mathcal{V}\text{-}\mathsf{Cat}$-category) if for all C-objects X and Y the hom-set $\mathsf{C}(X,Y)$ is a $\mathcal{V}$-category and if the composition of C-morphisms,*

$$(\ \cdot\ ) : \mathsf{C}(X,Y) \otimes \mathsf{C}(Y,Z) \longrightarrow \mathsf{C}(X,Z)$$

is a $\mathcal{V}$-functor. Given two $\mathcal{V}\text{-}\mathsf{Cat}$-categories C and D and a functor $F : \mathsf{C} \to \mathsf{D}$, we call F a $\mathcal{V}\text{-}\mathsf{Cat}$-functor if for all C-objects X and Y the map $F_{X,Y} : \mathsf{C}(X,Y) \to \mathsf{D}(FX,F,Y)$ is a $\mathcal{V}$-functor. An adjunction $\mathsf{C} : F \dashv G : \mathsf{D}$ is called $\mathcal{V}\text{-}\mathsf{Cat}$-enriched if the underlying functors F and G are $\mathcal{V}\text{-}\mathsf{Cat}$-functors and if for all objects $X \in |\mathsf{C}|$ and $Y \in |\mathsf{D}|$ there exists a $\mathcal{V}$-isomorphism $\mathsf{D}(FX,Y) \simeq \mathsf{C}(X,GY)$ natural in X and Y.

If C is a $\mathcal{V}\text{-}\mathsf{Cat}$-category then $\mathsf{C} \times \mathsf{C}$ is also a $\mathcal{V}\text{-}\mathsf{Cat}$-category via the tensor operation $\otimes$ in $\mathcal{V}\text{-}\mathsf{Cat}$. We take advantage of this fact in the following definition.

▶ **Definition 19.** *A $\mathcal{V}\text{-}\mathsf{Cat}$-enriched autonomous category C is an autonomous and $\mathcal{V}\text{-}\mathsf{Cat}$-category C such that the bifunctor $\otimes : \mathsf{C} \times \mathsf{C} \to \mathsf{C}$ is a $\mathcal{V}\text{-}\mathsf{Cat}$-functor and the adjunction $(- \otimes X) \dashv (X \multimap -)$ is a $\mathcal{V}\text{-}\mathsf{Cat}$-adjunction.*

▶ **Example 20.** Recall that $\mathsf{Pos} \simeq \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sep}}$ when $\mathcal{V}$ is the Boolean quantale. According to Theorem 17 the category Pos is autonomous. It follows by general results that the category is Pos-enriched [6]. It is also easy to see that its tensor is Pos-enriched and that the adjunction $(- \otimes X) \dashv (X \multimap -)$ is Pos-enriched. Therefore, Pos is an instance of Definition 19. Note also that $\mathsf{Set} \simeq \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym,sep}}$ for $\mathcal{V}$ the Boolean quantale and that Set is an instance of Definition 19.

Recall that $\mathsf{Met} \simeq \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym,sep}}$ when $\mathcal{V}$ is the metric quantale. Thus, the category Met is autonomous (Theorem 17) and Met-enriched [6]. It follows as well from routine calculations that its tensor is Met-enriched and that the adjunction $(- \otimes X) \dashv (X \multimap -)$ is Met-enriched. Therefore Met is an instance of Definition 19. An analogous reasoning tells that the category of ultrametric spaces (enriched over itself) is also an instance of Definition 19.

Finally, recall the interpretation of linear λ-terms on an autonomous category C (Section 2) and assume that C is $\mathcal{V}\text{-}\mathsf{Cat}$-enriched. Then we say that a $\mathcal{V}$-equation $\Gamma \rhd v =_q w : \mathbb{A}$ is *satisfied* by this interpretation if $a(\llbracket \Gamma \rhd v : \mathbb{A} \rrbracket, \llbracket \Gamma \rhd w : \mathbb{A} \rrbracket) \geq q$ where $a : \mathsf{C}(\llbracket \Gamma \rrbracket, \llbracket \mathbb{A} \rrbracket) \times \mathsf{C}(\llbracket \Gamma \rrbracket, \llbracket \mathbb{A} \rrbracket) \to \mathcal{V}$ is the underlying function of the $\mathcal{V}$-category $\mathsf{C}(\llbracket \Gamma \rrbracket, \llbracket \mathbb{A} \rrbracket)$.

▶ **Theorem 21.** *The rules listed in Fig. 3 and Fig. 4 are sound for $\mathcal{V}\text{-}\mathsf{Cat}$-enriched autonomous categories C. Specifically, if $\Gamma \rhd v =_q w : \mathbb{A}$ results from the rules in Fig. 3 and Fig. 4 then $a(\llbracket \Gamma \rhd v : \mathbb{A} \rrbracket, \llbracket \Gamma \rhd w : \mathbb{A} \rrbracket) \geq q$.*

Proof. Let us focus first on the equations listed in Fig. 3. Recall that an equation $\Gamma \rhd v = w : \mathbb{A}$ abbreviates the $\mathcal{V}$-equations $\Gamma \rhd v =_\top w : \mathbb{A}$ and $\Gamma \rhd w =_\top v : \mathbb{A}$. Moreover, we already know that the equations listed in Fig. 3 are sound for autonomous categories, specifically if $v = w$ is an equation of Fig. 3 then $\llbracket v \rrbracket = \llbracket w \rrbracket$ in C (Theorem 3). Thus, by the definition of a $\mathcal{V}$-category and by the assumption of $\mathcal{V}$ being integral ($k = \top$) we obtain $a(\llbracket v \rrbracket, \llbracket w \rrbracket) \geq k = \top$ and $a(\llbracket w \rrbracket, \llbracket v \rrbracket) \geq k = \top$.

Let us now focus on the rules listed in Fig. 4. The first three rules follow from the definition of a $\mathcal{V}$-category and the transitivity property of $\leq$. Rule **(arch)** follows from the continuity of $\mathcal{V}$, specifically from the fact that q is the *least* upper bound of all elements r that are way-below q. Rule **(join)** follows from the definition of least upper bound. The remaining rules follow from the definition of the tensor functor $\otimes$ in $\mathcal{V}$-Cat, the fact that C is $\mathcal{V}$-Cat-enriched, $\otimes : \mathsf{C} \times \mathsf{C} \to \mathsf{C}$ is a $\mathcal{V}$-Cat-functor, and the fact that $(- \otimes X) \dashv (X \multimap -)$ is a $\mathcal{V}$-Cat-adjunction. For example, for the sixth rule we reason as follows:

$$
\begin{aligned}
& a([\![f(v_1,\dots,v_n)]\!],[\![f(w_1,\dots,w_n)]\!]) \\
&= a([\![f]\!]\cdot([\![v_1]\!]\otimes\cdots\otimes[\![v_n]\!])\cdot \mathsf{sp}_{\Gamma_1;\dots;\Gamma_n}\cdot \mathsf{sh}_E, [\![f]\!]\cdot([\![w_1]\!]\otimes\cdots\otimes[\![w_n]\!])\cdot \mathsf{sp}_{\Gamma_1;\dots;\Gamma_n}\cdot \mathsf{sh}_E) \\
&\geq a([\![f]\!]\cdot([\![v_1]\!]\otimes\cdots\otimes[\![v_n]\!]), [\![f]\!]\cdot([\![w_1]\!]\otimes\cdots\otimes[\![w_n]\!])) \\
&\geq a([\![v_1]\!]\otimes\cdots\otimes[\![v_n]\!]), ([\![w_1]\!]\otimes\cdots\otimes[\![w_n]\!]) \\
&\geq a([\![v_1]\!],[\![w_1]\!])\otimes\cdots\otimes a([\![v_n]\!],[\![w_n]\!]) \\
&\geq q_1\otimes\cdots\otimes q_n
\end{aligned}
$$

where the second step follows from the fact that $\mathsf{sp}_{\Gamma_1;\dots;\Gamma_n}\cdot\mathsf{sh}_E$ is a morphism in C and that C is $\mathcal{V}$-Cat-enriched. The third step follows from an analogous reasoning. The fourth step follows from the fact that $\otimes : \mathsf{C}\times\mathsf{C}\to\mathsf{C}$ is a $\mathcal{V}$-Cat-functor. The last step follows from the premise of the rule in question. As another example, the proof for the substitution rule proceeds similarly:

$$
\begin{aligned}
& a([\![v[v'/x]]\!],[\![w[w'/x]]\!]) \\
&= a([\![v]\!]\cdot \mathsf{jn}_{\Gamma,\mathbb{A}}\cdot(\mathsf{id}\otimes[\![v']\!])\cdot \mathsf{sp}_{\Gamma;\Delta}, [\![w]\!]\cdot \mathsf{jn}_{\Gamma,\mathbb{A}}\cdot(\mathsf{id}\otimes[\![w']\!])\cdot \mathsf{sp}_{\Gamma;\Delta}) \\
&\geq a([\![v]\!]\cdot \mathsf{jn}_{\Gamma,\mathbb{A}}\cdot(\mathsf{id}\otimes[\![v']\!]), [\![w]\!]\cdot \mathsf{jn}_{\Gamma,\mathbb{A}}\cdot(\mathsf{id}\otimes[\![w']\!])) \\
&\geq a(\mathsf{id}\otimes[\![v']\!],\mathsf{id}\otimes[\![w']\!])\otimes a([\![v]\!]\cdot \mathsf{jn}_{\Gamma,\mathbb{A}}, [\![w]\!]\cdot \mathsf{jn}_{\Gamma,\mathbb{A}}) \\
&\geq a(\mathsf{id}\otimes[\![v']\!],\mathsf{id}\otimes[\![w']\!])\otimes a([\![v]\!],[\![w]\!]) \\
&\geq a(\mathsf{id},\mathsf{id})\otimes a([\![v']\!],[\![w']\!])\otimes a([\![v]\!],[\![w]\!]) \\
&= a([\![v']\!],[\![w']\!])\otimes a([\![v]\!],[\![w]\!]) \\
&\geq q\otimes r
\end{aligned}
$$

The proof for the rule concerning $(\multimap_i)$ additionally requires the following two facts: if a $\mathcal{V}$-functor $f : (X,a)\to(Y,b)$ is an isomorphism then $a(x,x') = b(f(x),f(x'))$ for all $x,x'\in X$. For a context Γ, the morphism $\mathsf{jn}_{\Gamma;x:\mathbb{A}} : [\![\Gamma]\!]\otimes[\![\mathbb{A}]\!]\to[\![\Gamma, x:\mathbb{A}]\!]$ is an isomorphism in C. The proof for the rule concerning the permutation of variables (exchange) also makes use of the fact that $[\![\Delta]\!]\to[\![\Gamma]\!]$ is an isomorphism. ◀

3.3 Soundness, completeness, and internal language

In this subsection we establish a formal correspondence between linear $\mathcal{V}\lambda$-theories and $\mathcal{V}$-Cat-enriched autonomous categories, via soundness, completeness, and internal language theorems. A key construct in this correspondence is the quotienting of a $\mathcal{V}$-category into a *separated* $\mathcal{V}$-category: we will use it to identify linear λ-terms when generating a syntactic category (from a linear $\mathcal{V}\lambda$-theory) that satisfies the axioms of autonomous categories. This naturally leads to the following notion of a model for linear $\mathcal{V}\lambda$-theories.

▶ **Definition 22** (Models of linear $\mathcal{V}\lambda$-theories). *Consider a linear $\mathcal{V}\lambda$-theory $((G,\Sigma),Ax)$ and a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched autonomous category C. Suppose that for each $X\in G$ we have an interpretation $[\![X]\!]$ as a C-object and analogously for the operation symbols. This interpretation structure is a model of the theory if all axioms in Ax are satisfied by the interpretation.*

Another thing that we need to take into account is the size of categories. In Section 2 we did not assume that autonomous categories should be locally small. In particular linear λ-theories are able to generate non-(locally small) categories. Now we need to be stricter because $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched autonomous categories are always locally small (recall the definition of $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$). Thus for two types $\mathbb{A}$ and $\mathbb{B}$ of a $\mathcal{V}\lambda$-theory T, consider the class $\mathsf{Values}(\mathbb{A},\mathbb{B})$ of values v such that $x : \mathbb{A} \rhd v : \mathbb{B}$. We equip $\mathsf{Values}(\mathbb{A},\mathbb{B})$ with the function $a : \mathsf{Values}(\mathbb{A},\mathbb{B}) \times \mathsf{Values}(\mathbb{A},\mathbb{B}) \to \mathcal{V}$ defined by,

$$a(v,w) = \bigvee\{q \mid v =_q w \text{ is a theorem of } T\}$$

It is easy to see that $(\mathsf{Values}(\mathbb{A},\mathbb{B}), a)$ is a (possibly large) $\mathcal{V}$-category. We then quotient this $\mathcal{V}$-category into a *separated* $\mathcal{V}$-category which we suggestively denote by $\mathsf{C}(\mathbb{A},\mathbb{B})$ (as detailed in the proof of the next theorem, $\mathsf{C}(\mathbb{A},\mathbb{B})$ will serve as a hom-object of a syntactic category C generated from a linear $\mathcal{V}\lambda$-theory). Following the nomenclature of [26], we call T *varietal* if $\mathsf{C}(\mathbb{A},\mathbb{B})$ is a *small* $\mathcal{V}$-category. In the rest of the paper we will only work with varietal theories and locally small categories.

▶ **Theorem 23** (Soundness & Completeness)**.** *Consider a varietal $\mathcal{V}\lambda$-theory. A $\mathcal{V}$-equation-in-context $\Gamma \rhd v =_q w : \mathbb{A}$ is a theorem iff it holds in all models of the theory.*

Proof sketch. Soundness follows by induction over the rules that define the class $Th(Ax)$ (Definition 12) and by Theorem 21. For completeness, we use a strategy similar to the proof of Theorem 6, and take advantage of the quotienting of a $\mathcal{V}$-category into a separated $\mathcal{V}$-category. Recall that we assume that the theory is *varietal* and therefore can safely take $\mathsf{C}(\mathbb{A},\mathbb{B})$ to be a small $\mathcal{V}$-category. Note that the quotienting process identifies all terms $x : \mathbb{A} \rhd v : \mathbb{B}$ and $x : \mathbb{A} \rhd w : \mathbb{B}$ such that $v =_\top w$ and $w =_\top v$. Such a relation contains the equations-in-context from Fig. 3 and moreover it is straighforward to show that it is compatible with the term formation rules of linear λ-calculus (Fig. 1). So, analogously to Theorem 6 we obtain an autonomous category C whose objects are the types of the language and whose hom-sets are the underlying sets of the $\mathcal{V}$-categories $\mathsf{C}(\mathbb{A},\mathbb{B})$.

Our next step is to show that the category C has a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched autonomous structure. We start by showing that the composition map $\mathsf{C}(\mathbb{A},\mathbb{B}) \otimes \mathsf{C}(\mathbb{B},\mathbb{C}) \to \mathsf{C}(\mathbb{A},\mathbb{C})$ is a $\mathcal{V}$-functor:

$$\begin{aligned}
a(([v'],[v]),([w'],[w])) &= a([v],[w]) \otimes a([v'],[w']) \\
&= a(v,w) \otimes a(v',w') \\
&= \bigvee\{q \mid v =_q w\} \otimes \bigvee\{r \mid v' =_r w'\} \\
&= \bigvee\{q \otimes r \mid v =_q w, v' =_r w'\} \\
&\leq \bigvee\{q \mid v[v'/x] =_q w[w'/x]\} \qquad (A \subseteq B \Rightarrow \textstyle\bigvee A \leq \bigvee B) \\
&= a(v[v'/x], w[w'/x]) \\
&= a([v[v'/x]],[w[w'/x]]) \\
&= a([v]\cdot[v'],[w]\cdot[w'])
\end{aligned}$$

The fact that $\otimes : \mathsf{C} \times \mathsf{C} \to \mathsf{C}$ is a $\mathcal{V}$-Cat-functor follows by an analogous reasoning. Next, we need to show that $(- \otimes X) \dashv (X \multimap -)$ is a $\mathcal{V}$-Cat-adjunction. It is straightforward to show that both functors are $\mathcal{V}$-Cat-functors, and from a similar reasoning it follows that the isomorphism $\mathsf{C}(\mathbb{B}, \mathbb{A} \multimap \mathbb{C}) \simeq \mathsf{C}(\mathbb{B} \otimes \mathbb{A}, \mathbb{C})$ is a $\mathcal{V}$-isomorphism.

The final step is to show that if an equation $\Gamma \rhd v =_q w : \mathbb{A}$ with $q \in B$ is satisfied by C then it is a theorem of the linear $\mathcal{V}\lambda$-theory. By assumption $a([v],[w]) = a(v,w) = \bigvee\{r \mid v =_r w\} \geq q$. It follows from the definition of the way-below relation that for all $x \in B$ with

$x \ll q$ there exists a *finite* set $A \subseteq \{r \mid v =_r w\}$ such that $x \leq \bigvee A$. Then by an application of rule **(join)** in Fig. 4 we obtain $v =_{\bigvee A} w$, and consequently rule **(weak)** in Fig. 4 provides $v =_x w$ for all $x \ll q$. Finally, by an application of rule **(arch)** in Fig. 4 we deduce that $v =_q w$ is part of the theory. ◀

Next we establish results that will be key in the proof of the internal language theorem. Let $\mathrm{Syn}(T)$ be syntactic category of a linear $\mathcal{V}\lambda$-theory T, as described in Theorem 23.

▶ **Theorem 24.** *Consider a linear $\mathcal{V}\lambda$-theory T and a model of T on a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched autonomous category C. The model induces a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-functor $\mathrm{Syn}(T) \to \mathsf{C}$ that (strictly) preserves the autonomous structure of $\mathrm{Syn}(T)$.*

Proof. Consider a model of T over C. Let a denote the underlying function of the hom-($\mathcal{V}$-categories) in $\mathrm{Syn}(T)$ and b the underlying function of the hom-($\mathcal{V}$-categories) in C. Then note that if $[v] = [w]$ then, by completeness, the equations $v =_\top w$ and $w =_\top v$ are theorems, which means that $[\![v]\!] = [\![w]\!]$ by the definition of a model and *separability*. This allows us to define a mapping $F : \mathrm{Syn}(T) \to \mathsf{C}$ that sends each type $\mathbb{A}$ to $[\![\mathbb{A}]\!]$ and each morphism $[v]$ to $[\![v]\!]$. The fact that this mapping is an autonomous functor follows from an analogous reasoning to the one used in the proof of Theorem 7. We now need to show that this functor is $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched. Recall that $a([v],[w]) = \bigvee\{q \mid v =_q w\}$ and observe that for every $v =_q w$ in the previous quantification we have $b([\![v]\!],[\![w]\!]) \geq q$ (by the definition of a model), which establishes, by the definition of a least upper bound, $a([v],[w]) = \bigvee\{q \mid v =_q w\} \leq b([\![v]\!],[\![w]\!])$. ◀

Consider now a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched autonomous category C. It induces a linear $\mathcal{V}\lambda$-theory $\mathrm{Lang}(\mathsf{C})$ whose ground types and operations symbols are defined as in the case of linear λ-theories (recall Section 2). The axioms of $\mathrm{Lang}(\mathsf{C})$ are all the $\mathcal{V}$-equations-in-context that are satisfied by the obvious interpretation on C.

▶ **Theorem 25.** *The linear $\mathcal{V}\lambda$-theory $\mathrm{Lang}(\mathsf{C})$ is varietal.*

In conjunction with the proof of Theorem 23, a consequence of this last theorem is that $\mathrm{Syn}(\mathrm{Lang}(\mathsf{C}))$ is a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched category. Then we state,

▶ **Theorem 26** (Internal language)**.** *For every $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched autonomous category C there exists a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-equivalence of categories $\mathrm{Syn}(\mathrm{Lang}(\mathsf{C})) \simeq \mathsf{C}$.*

Proof. Let a denote the underlying function of the hom-($\mathcal{V}$-categories) in $\mathrm{Syn}(\mathrm{Lang}(\mathsf{C}))$ and b the underlying function of the hom-($\mathcal{V}$-categories) in C. We have, by construction, a model of $\mathrm{Lang}(\mathsf{C})$ on C which acts as the identity in the interpretation of ground types and operation symbols. We can then appeal to Theorem 24 to establish a $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-functor $\mathrm{Syn}(\mathrm{Lang}(\mathsf{C})) \to \mathsf{C}$. Next, the functor working on the inverse direction behaves as the identity on objects and sends a morphism f into $[f(x)]$. Let us show that it is $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched. First, observe that if $q \ll b(f,g)$ in C and $q \in B$ then $f(x) =_q g(x)$ is a theorem of $\mathrm{Lang}(\mathsf{C})$, due to the fact that $\ll$ entails $\leq$ and by the definition of $\mathrm{Lang}(\mathsf{C})$. Using the definition of a basis, we thus obtain $b(f,g) = \bigvee\{q \in B \mid q \ll b(f,g)\} \leq \bigvee\{q \in B \mid f(x) =_q g(x)\} = a([f(x)],[g(x)])$. The equivalence of categories is then shown as in the proof of Theorem 8. ◀

All the results in this section can be extended straightforwardly to the case of *symmetric* linear $\mathcal{V}\lambda$-theories and $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym},\mathsf{sep}}$-enriched autonomous categories.

4 Examples of linear $\mathcal{V}\lambda$-theories and their models

▶ **Example 27** (Wait calls)**.** We now return to the example of wait calls and the corresponding metric axioms (1) sketched in the Introduction. Let us build a model over Met for this theory: fix a metric space A, interpret the ground type X as $\mathbb{N} \otimes A$ and the operation symbol $\mathtt{wait_n} : X \to X$ as the non-expansive map, $[\![\mathtt{wait_n}]\!] : \mathbb{N} \otimes A \to \mathbb{N} \otimes A$, $(i, a) \mapsto (i + n, a)$. Since we already know that Met is enriched over itself (recall Definition 19 and Example 20) we only need to show that the axioms in (1) are satisfied by the proposed interpretation. This can be shown via a few routine calculations.

Now, it may be the case that is unnecessary to know the *distance* between the execution time of two programs – instead it suffices to know whether a program finishes its execution *before* another one. This leads us to linear $\mathcal{V}\lambda$-theories where $\mathcal{V}$ is the Boolean quantale. We call such theories *linear ordered λ-theories*. Recall the language from the Introduction with a single ground type X and the signature of wait calls $\Sigma = \{\mathtt{wait_n} : X \to X \mid n \in \mathbb{N}\}$. Then we adapt the metric axioms (1) to the case of the Boolean quantale by considering instead:

$$\mathtt{wait_0}(x) = x \qquad \mathtt{wait_n}(\mathtt{wait_m}(x)) = \mathtt{wait_{n+m}}(x) \qquad \frac{n \leq m}{\mathtt{wait_n}(x) \leq \mathtt{wait_m}(x)}$$

where a classical equation $v = w$ is shorthand for $v \leq w$ (*i.e.* $v =_1 w$) and $w \leq v$ (*i.e.* $w =_1 v$). In the resulting theory we can consider for instance (and omitting types for simplicity) the λ-term that defines the composition of two functions $\lambda f.\ \lambda g.\ g\ (f\ x)$, which we denote by v, and show that $v\ (\lambda x.\ \mathtt{wait_1}(x)) \leq v\ (\lambda x.\ \mathtt{wait_1}(\mathtt{wait_1}(x)))$. This inequation between higher-order programs arises from the argument $\lambda x.\ \mathtt{wait_1}(\mathtt{wait_1}(x))$ being costlier than the argument $\lambda x.\ \mathtt{wait_1}(x)$ – specifically, the former will invoke one more wait call ($\mathtt{wait_1}$) than the latter. Moreover, the inequation entails that for every argument g the execution time of computation $v\ (\lambda x.\ \mathtt{wait_1}(x))\ g$ will always be smaller than that of computation $v\ (\lambda x.\ \mathtt{wait_1}(\mathtt{wait_1}(x)))\ g$ since it invokes one more wait call. Thus in general the inequation tells that costlier programs fed as input to v will result in longer execution times when performing the corresponding computation. In order to build a model for the ordered theory of wait calls, we consider a poset A and define a model over Pos by sending X into $\mathbb{N} \otimes A$ and $\mathtt{wait_n} : X \to X$ to the monotone map $[\![\mathtt{wait_n}]\!] : \mathbb{N} \otimes A \to \mathbb{N} \otimes A$, $(i, a) \mapsto (i + n, a)$. Since we already know that Pos is enriched over itself (recall Definition 19 and Example 20) we only need to show that the ordered axioms are satisfied by the proposed interpretation. But again, this can be shown via a few routine calculations.

▶ **Example 28** (Probabilistic programs)**.** We consider ground types $\mathtt{Real}, \mathtt{Real}^+, \mathtt{unit}$ and a signature consisting of $\{r : \mathbb{I} \to \mathtt{Real} \mid r \in \mathbb{Q}\} \cup \{r^+ : \mathbb{I} \to \mathtt{Real}^+ \mid r \in \mathbb{Q}_{\geq 0}\} \cup \{r^u : \mathbb{I} \to \mathtt{unit} \mid r \in [0, 1] \cap \mathbb{Q}\}$, an operation $+$ of type $\mathtt{Real}, \mathtt{Real} \to \mathtt{Real}$, and *sampling* functions $\mathtt{bernoulli} : \mathtt{Real}, \mathtt{Real}, \mathtt{unit} \to \mathtt{Real}$ and $\mathtt{normal} : \mathtt{Real}, \mathtt{Real}^+ \to \mathtt{Real}$. Whenever no ambiguities arise, we drop the superscripts in r^u and r^+. Operationally, $\mathtt{bernoulli}(x, y, p)$ generates a sample from the Bernoulli distribution with parameter p on the set $\{x, y\}$, whilst $\mathtt{normal}(x, y)$ generates a normal deviate with mean x and standard deviation y. We then postulate the metric axiom,

$$\frac{p, q \in [0, 1] \cap \mathbb{Q}}{\mathtt{bernoulli}(x_1, x_2, p(*)) =_{|p-q|} \mathtt{bernoulli}(x_1, x_2, q(*))} \tag{3}$$

We interpret the resulting linear metric λ-theory in the category Ban of Banach spaces and short operators, *i.e.* the semantics of [9, 21] without the order structure needed to interpret `while` loops. This is the usual representation of Markov chains/kernels as matrices/operators.

▶ **Theorem 29.** *The category* Ban *is a* Met*-enriched autonomous category, and thus an instance of Definition 19.*

In particular, Ban forms a model for the theory of our small probabilistic language via the following interpretation. We define $[\![\texttt{Real}]\!] = \mathcal{M}\mathbb{R}$, the Banach space of finite Borel measures on $\mathbb{R}$ equipped with the total variation norm, and similarly $[\![\texttt{Real}^+]\!] = \mathcal{M}\mathbb{R}^+$ and $[\![\texttt{unit}]\!] = \mathcal{M}[0,1]$. We have $[\![\mathbb{I}]\!] = \mathbb{R} \ni 1$, and for every $r \in \mathbb{Q}$ we put $[\![r]\!] : \mathbb{R} \to \mathcal{M}\mathbb{R}, x \mapsto x\delta_r$, where δ_r is the Dirac delta over r; thus $[\![r]\!](1) = \delta_r$. We define an analogous interpretation for the operation symbols r^+ and r^u. For $\mu, \upsilon \in \mathcal{M}\mathbb{R}$ we define $[\![+]\!](\mu \otimes \upsilon) \triangleq +_*(\mu \otimes \upsilon)$ the pushforward under $+$ of the product measure $\mu \otimes \upsilon$ (seen as an element of $\mathcal{M}\mathbb{R} \otimes \mathcal{M}\mathbb{R}$, see [9]). For $\mu, \upsilon, \xi \in \mathcal{M}\mathbb{R}$ we define $[\![\texttt{bernoulli}]\!](\mu \otimes \upsilon \otimes \xi) \triangleq \mathrm{bern}_*(\mu \otimes \upsilon \otimes \xi)$, the pushforward of the product measure $\mu \otimes \upsilon \otimes \xi$ under the Markov kernel $\mathrm{bern} : \mathbb{R}^3 \to \mathbb{R}, (u, v, p) \mapsto p\delta_u + (1-p)\delta_v$, and similarly for $[\![\texttt{normal}]\!]$ (see [9] for the definition of pushforward by a Markov kernel).

This interpretation is sound (a proof is given in [10]) because the norm on $\mathcal{M}\mathbb{R}$ is the total variation norm, and the metric axiom (3) describes the total variation distance between the corresponding Bernoulli distributions. Consider now the following λ-terms (where we abbreviate the constants $0(*), 1(*), p(*), q(*)$ to $0, 1, p, q$, respectively),

$$\texttt{walk1} \triangleq \lambda x : \texttt{Real}.\texttt{bernoulli}(0, x + \texttt{normal}(0,1), p)$$
$$\texttt{walk2} \triangleq \lambda x : \texttt{Real}.\texttt{bernoulli}(0, x + \texttt{normal}(0,1), q), \qquad p, q \in [0,1] \cap \mathbb{Q}.$$

As the names suggest, these two terms of type $\texttt{Real} \multimap \texttt{Real}$ are denoted by random walks on $\mathbb{R}$. At each call, `walk1` (resp. `walk2`) performs a jump drawn randomly from a standard normal distribution, or is forced to return to the origin with probability p (resp. q). These are non-standard random walks whose semantics are concretely given by complicated operators $\mathcal{M}\mathbb{R} \to \mathcal{M}\mathbb{R}$, but the simple quantitative equational system of Fig. 4 and the axiom (3) allow us to easily derive $\texttt{walk1} =_{|p-q|} \texttt{walk2}$ without having to compute the semantics of these terms. In other words, the soundness of (3) is enough to tightly bound the distance between two non-trivial random walks represented as higher-order terms in a probabilistic programming language. Furthermore, the tensor in the λ-calculus allows us to easily scale up this reasoning to random walks in higher dimensions such as $\texttt{walk1} \otimes \texttt{walk2}$ on $\mathbb{R}^2$.

5 Conclusions and future work

We introduced the notion of a $\mathcal{V}$-equation which generalises the well-established notions of equation, inequation [23, 2], and metric equation [30, 31]. We then presented a sound and complete $\mathcal{V}$-equational system for linear λ-calculus, illustrated with different examples of programs containing real-time and probabilistic behaviour.

Functorial connection to previous work. As a concluding note, let us introduce a simple yet instructive functorial connection between (1) the categorical semantics of linear λ-calculus with the $\mathcal{V}$-equational system, (2) the categorical semantics of linear λ-calculus with the equational system of Section 2, and (3) the algebraic semantics of the exponential free, multiplicative fragment of linear logic. First we need to recall some well-known facts. As detailed before, typical categorical models of linear λ-calculus and its equational system are locally small autonomous categories. The latter form a quasicategory Aut whose morphisms are autonomous functors. The usual algebraic models of the exponential free, multiplicative fragment of linear logic are the so-called *lineales* [11]. In a nutshell, a lineale is a poset $(X, \leq)$ paired with a commutative, monoid operation $\otimes : X \times X \to X$ that satisfies certain

conditions. Lineales are almost quantales: the only difference is that they do not require X to be cocomplete. The key idea in algebraic semantics is that the order $\leq$ in the lineale encodes the logic's entailment relation. A functorial connection between autonomous categories and lineales (*i.e.* between (2) and (3)) is stated in [11] and is based on the following two observations. First, (possibly large) lineales can be seen as *thin* autonomous categories, *i.e.* as elements of the enriched quasicategory $\{0,1\}$-Aut. Second, the inclusion $\{0,1\}$-Aut $\hookrightarrow$ Aut has a left adjoint which *collapses all morphisms* of a given autonomous category C (intuitively, it eliminates the ability of C to differentiate different terms between two types). This provides an adjoint situation between (2) and (3). We can now expand this connection to our categorical semantics of linear λ-calculus and corresponding $\mathcal{V}$-equational system (*i.e.* (1)) in the following way. The forgetful functor $\mathcal{V}$-$\mathsf{Cat} \to \mathsf{Set}$ has a left adjoint $D : \mathsf{Set} \to \mathcal{V}$-$\mathsf{Cat}$ which sends a set X to $DX = (X, d)$, with $d(x_1, x_2) = k$ if $x_1 = x_2$ and $d(x_1, x_2) = \bot$ otherwise. This left adjoint is strong monoidal, specifically we have $D(X_1 \times X_2) = DX_1 \otimes DX_2$ and $\mathbb{I} = (1, (*, *) \mapsto k) = D1$. This gives rise to the functors,

$$(\mathcal{V}\text{-}\mathsf{Cat})\text{-}\mathsf{Aut} \underset{\bot}{\overset{\hat{D}}{\leftarrow\!\!\rightarrow}} \mathsf{Aut} \underset{\bot}{\overset{c}{\rightarrow\!\!\leftarrow}} \{0,1\}\text{-}\mathsf{Aut}$$

where $\hat{D}$ equips the hom-sets of an autonomous category with the corresponding discrete $\mathcal{V}$-category and c collapses all morphisms of an autonomous category as described earlier. The right adjoint of $\hat{D}$ forgets the $\mathcal{V}$-categorical structure between terms (*i.e.* morphisms) and the right adjoint of c is the inclusion functor mentioned earlier. Note that $\hat{D}$ restricts to $(\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sep}})$-$\mathsf{Aut}$ and $(\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym,sep}})$-$\mathsf{Aut}$, and thus we obtain a functorial connection between the categorical semantics of linear λ-calculus with the $\mathcal{V}$-equational system (*i.e.* (1)), (2), and (3). In essence, the connection formalises the fact that our categorical models admit a richer structure over terms (*i.e.* morphisms) than the categorical models of linear λ-calculus and its classical equational system. The latter in turn permits the existence of different terms between two types as opposed to the algebraic semantics of the exponential free, multiplicative fragment of linear logic. The connection also shows that models for (2) and (3) can be mapped into models of our categorical semantics by equipping the respective hom-sets with a trivial, discrete structure.

Future work. Recall that linear λ-calculus is at the root of different ramifications of λ-calculus that relax resource-based conditions in different ways. Currently, we are studying analogous ramifications of linear λ-calculus in the $\mathcal{V}$-equational setting, particularly affine and Cartesian versions. We are also studying the possibility of adding an exponential modality in order to obtain a *mixed linear-non-linear* calculus [3]. We also started to explore different definitions of a morphism between $\mathcal{V}\lambda$-theories and respective categories. This is the basis to establish a categorical equivalence between a (quasi)category of $\mathcal{V}\lambda$-theories and a (quasi)category of $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sep}}$-enriched autonomous categories.

Next, our main examples of $\mathcal{V}\lambda$-theories (see Section 4) used either the Boolean or the metric quantale. We would like to study linear $\mathcal{V}\lambda$-theories whose underlying quantales are neither the Boolean nor the metric one, for example the ultrametric quantale which is (tacitly) used to interpret Nakano's guarded λ-calculus [5] and also to interpret a higher-order language for functional reactive programming [22]. Another interesting quantale is the Gödel one which is a basis for fuzzy logic [12] and whose $\mathcal{V}$-equations give rise to what we call fuzzy inequations.

Finally we plan to further explore the connections between our work and different results on metric universal algebra [30, 31, 36] and inequational universal algebra [23, 2, 36]. For example, an interesting connection is that the monad construction presented in [30] crucially relies on quotienting a pseudometric space into a metric space – this is a particular case of quotienting a $\mathcal{V}$-category into a separated $\mathcal{V}$-category (which we crucially use in our work).

References

1 Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G Riecke. A core calculus of dependency. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 147–160, 1999.

2 Jiří Adámek, Chase Ford, Stefan Milius, and Lutz Schröder. Finitary monads on the category of posets, 2020. `arXiv:2011.14796`.

3 Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *International Workshop on Computer Science Logic*, pages 121–135. Springer, 1994.

4 Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. *Term assignment for intuitionistic linear logic (preliminary report)*. Citeseer, 1992.

5 Lars Birkedal, Jan Schwinghammer, and Kristian Støvring. A metric model of lambda calculus with guarded recursion. In *FICS*, pages 19–25, 2010.

6 Francis Borceux. *Handbook of categorical algebra*, volume 2 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994.

7 Raphaëlle Crubillé and Ugo Dal Lago. Metric reasoning about λ-terms: The affine case. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 633–644. IEEE, 2015.

8 Raphaëlle Crubillé and Ugo Dal Lago. Metric Reasoning About *lambda*-Terms: The General Case. In *European Symposium on Programming*, pages 341–367. Springer, 2017.

9 Fredrik Dahlqvist and Dexter Kozen. Semantics of higher-order probabilistic programs with conditioning. In *Proc. 47th ACM SIGPLAN Symp. Principles of Programming Languages (POPL'20)*, pages 57:1–29, New Orleans, January 2020. ACM.

10 Fredrik Dahlqvist and Renato Neves. An internal language for categories enriched over generalised metric spaces, 2021. `arXiv:2105.08473`.

11 Valeria De Paiva. Lineales: algebraic models of linear logic from a categorical perspective. In *Proceedings of LLC8*, 1999.

12 Klaus Denecke, Marcel Erné, and Shelly L Wismath. *Galois connections and applications*, volume 565. Springer Science & Business Media, 2013.

13 Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart, and Tarmo Uustalu. Combining effects and coeffects via grading. *ACM SIGPLAN Notices*, 51(9):476–489, 2016.

14 Francesco Gavazzo. Quantitative behavioural reasoning for higher-order effectful programs: Applicative distances. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 452–461, 2018.

15 Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael W. Mislove, and Dana S. Scott. *Continuous lattices and domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2003.

16 Jean-Yves Girard, Andre Scedrov, and Philip J Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical computer science*, 97(1):1–66, 1992.

17 Jean Goubault-Larrecq. *Non-Hausdorff Topology and Domain Theory—Selected Topics in Point-Set Topology*, volume 22 of *New Mathematical Monographs*. Cambridge University Press, March 2013.

18 Dirk Hofmann and Pedro Nora. Hausdorff coalgebras. *Applied Categorical Structures*, 28(5):773–806, 2020.

19 Shih-Han Hung, Kesha Hietala, Shaopeng Zhu, Mingsheng Ying, Michael Hicks, and Xiaodi Wu. Quantitative robustness analysis of quantum programs. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019.

20 Gregory Maxwell Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982.

21 Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, June 1981. doi:10.1016/0022-0000(81)90036-2.

22 Neelakantan R Krishnaswami and Nick Benton. Ultrametric semantics of reactive programs. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*, pages 257–266. IEEE, 2011.

23 Alexander Kurz and Jiří Velebil. Quasivarieties and varieties of ordered algebras: regularity and exactness. *Mathematical Structures in Computer Science*, 27(7):1153–1194, 2017.

24 Joachim Lambek and Philip J Scott. *Introduction to higher-order categorical logic*, volume 7. Cambridge University Press, 1988.

25 F William Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del seminario matématico e fisico di Milano*, 43(1):135–166, 1973.

26 Fred E. J. Linton. Some aspects of equational categories. In *Proceedings of the Conference on Categorical Algebra*, pages 84–94. Springer, 1966.

27 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. springer, 1998.

28 Ian Mackie, Leopoldo Román, and Samson Abramsky. An internal language for autonomous categories. *Applied Categorical Structures*, 1(3):311–343, 1993.

29 Maria Emilia Maietti, Paola Maneggia, Valeria De Paiva, and Eike Ritter. Relating categorical semantics for intuitionistic linear logic. *Applied categorical structures*, 13(1):1–36, 2005.

30 Radu Mardare, Prakash Panangaden, and Gordon Plotkin. Quantitative algebraic reasoning. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 700–709, 2016.

31 Radu Mardare, Prakash Panangaden, and Gordon Plotkin. On the axiomatizability of quantitative algebras. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.

32 Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–30, 2019.

33 Jan Paseka and Jiří Rosickỳ. Quantales. In *Current research in operational quantum logic*, pages 245–262. Springer, 2000.

34 Paolo Pistone. On Generalized Metric Spaces for the Simply Typed λ-Calculus. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2021.

35 Jason Reed and Benjamin C Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 157–168, 2010.

36 Jiří Rosickỳ. Metric monads, 2020. arXiv:2012.14641.

37 Michael Shulman. A practical type theory for symmetric monoidal categories. *arXiv preprint*, 2019. arXiv:1911.00818.

38 Isar Stubbe. An introduction to quantaloid-enriched categories. *Fuzzy Sets and Systems*, 256:95–116, 2014.

39 Jiří Velebil, Alexander Kurz, and Adriana Balan. Extending set functors to generalised metric spaces. *Logical Methods in Computer Science*, 15, 2019.

MSO Undecidability for Hereditary Classes of Unbounded Clique Width

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, UK

Abhisekh Sankaran
Department of Computer Science and Technology, University of Cambridge, UK

Abstract

Seese's conjecture for finite graphs states that monadic second-order logic (MSO) is undecidable on all graph classes of unbounded clique-width. We show that to establish this it would suffice to show that grids of unbounded size can be interpreted in two families of graph classes: minimal hereditary classes of unbounded clique-width; and antichains of unbounded clique-width under the induced subgraph relation. We explore all the currently known classes of the former category and establish that grids of unbounded size can indeed be interpreted in them.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases clique width, Seese's conjecture, antichain, MSO interpretation, grid

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.17

Related Version *Full Version*: https://arxiv.org/abs/2011.02894

Funding Research supported by the Leverhulme Trust through a Research Project Grant on "Logical Fractals".

1 Introduction

The monadic second-order logic (MSO) of graphs has been an object of intensive research for many years now. It is a logic that is highly expressive and yet very well behaved on many interesting classes of graphs. It has enabled the extension of many automata-theoretic and algebraic techniques to the construction of algorithms on graphs (see the comprehensive treatment in [7]). It has become a reference logic against which many others are compared. A key area of investigation is determining on which classes of graphs MSO is algorithmically well-behaved.

The good algorithmic behaviour of MSO on a class $\mathscr{C}$ of graphs is usually taken to mean one of two things: the evaluation (or model-checking) problem for MSO sentences on $\mathscr{C}$ is tractable; or the satisfiability problem of MSO sentences on $\mathscr{C}$ is decidable. Usually, these two are linked. Broadly speaking, the only way we know to show that the MSO theory of a class $\mathscr{C}$ is decidable is to show that $\mathscr{C}$ can be obtained by means of an MSO interpretation from a class of trees, which itself has a decidable theory and this also yields efficient evaluation algorithms for MSO sentences on $\mathscr{C}$. And the only way we know to show that the MSO theory of $\mathscr{C}$ is undecidable is to show that there is an MSO interpretation that yields arbitrarily large grids on $\mathscr{C}$ and this also yields an obstacle to the tractability of MSO evaluation on $\mathscr{C}$.

Seese [20] formalizes the first of these observations into a conjecture: if the MSO theory of a class $\mathscr{C}$ is decidable, there is an MSO interpretation Ψ and a class $\mathscr{T}$ of trees such that Ψ maps $\mathscr{T}$ to $\mathscr{C}$. This remains an open question nearly three decades after it was first posed despite considerable research effort around it. By a theorem of Courcelle and Engelfriet [7], it is known that the classes of graphs obtained by MSO interpretations from trees are exactly those of bounded clique-width. Thus, Seese's conjecture can be understood as saying that any class of graphs of unbounded clique-width has an undecidable MSO theory.

If we similarly formalize the second observation above about grids and combine it with this, we can formulate the following stronger conjecture, which we refer to below as the *strong Seese conjecture*: every class $\mathscr{C}$ of graphs of unbounded clique-width admits an MSO interpretation that defines arbitrarily large grids. Seese's conjecture is often formulated in this stronger form as it seems the only reasonable route to proving it. It can be seen as an interesting analogue of the Robertson-Seymour grid minor theorem to the effect that any class of graphs of unbounded treewidth admits arbitrarily large grids as minors.

In recent years there has been growing interest in clique-width as a measure of the complexity of graphs from a structural and algorithmic point of view, quite separate from questions of logic [10, 4, 18, 11]. In particular, it provides a route for extending algorithmic methods that have had great success on sparse graph classes [17] to more general classes of graphs. A class of graphs may be of bounded clique-width while containing dense graphs – the classic example being the class of cliques.

In the context of the structural study of classes of bounded clique-width, there is particular interest in *hereditary classes*, that is, classes of graphs closed under the operation of taking induced subgraphs. This is because the induced subgraph relation behaves well with respect to clique-width. If a graph H is a subgraph or a minor of a graph G, the clique-width of H can be greater than that of G but if H is an *induced subgraph* of G, then the clique-width of H is no more than that of G. Hence, the hereditary closure of a class $\mathscr{C}$ of bounded clique-width still has bounded clique-width.

The induced subgraph relation is not as well-behaved as the graph minor relation. By the Robertson-Seymour graph minor theorem [19], the graph minor relation is a well-quasi-order. This is not true for the induced subgraph relation. By the same token, the classes of graphs of unbounded treewidth are well understood in that they are precisely the classes which have grid minors of unbounded size, the picture for classes of graphs of unbounded clique-width is somewhat less clear. The relationship between a class having unbounded clique-width and admitting a well-quasi-order of the induced subgraph relation has been the subject of much investigation. It is possible to construct as we see below, infinite descending chains, under inclusion, of hereditary classes of graphs, each of unbounded clique-width.

Lozin [15] identified the first example of a hereditary class $\mathscr{C}$ of graphs of unbounded clique-width that are *minimal* with this property – that is, no hereditary proper subclass of $\mathscr{C}$ has unbounded clique-width. Since then, many other such classes have been constructed. Collins et al. [3] show how to obtain an infinite family of such classes. Their construction has been recently extended by Brignall and Cocks [2] to obtain uncountably many examples. Atminas et al. [1] construct instances of such classes which are characterized by a finite collection of forbidden induced subgraphs. Lozin et al. [16] construct a minimal hereditary class of unbounded clique-width that is well-quasi-ordered under the induced subgraph relation.

This exploration of novel classes of unbounded clique-width also suggests an approach to establishing Seese's conjecture for finite graphs. We establish in Section 3 that a proof of Seese's conjecture would follow from the conjunction of the following two statements: (1) every collection of graphs of unbounded clique-width that forms an infinite anti-chain under the induced subgraph relation interprets arbitrarily large grids; and (2) every minimal hereditary class of unbounded clique-width interprets arbitrarily large grids. This suggests a programme to establish Seese's conjecture by systematically studying antichains and minimal hereditary classes of unbounded clique-width. We do not yet know of a complete classification of minimal hereditary classes of unbounded clique-width, which makes a systematic approach to this programme challenging. Nevertheless, we examine in Sections 4–5 all known classes satisfying

these conditions and show that in all cases we can indeed interpret grids of unbounded size. Thus none of these provides a counterexample to Seese's conjecture. Our construction shows a uniform method of proving that these classes have unbounded clique-width. The proof is often simpler than the *ad hoc* methods by which this was proved for each class in the literature.

It is worth mentioning some significant lines of investigation related to Seese's conjecture. Courcelle [5] shows that proving Seese's conjecture for finite graphs is equivalent to proving the relativized version of the conjecture for particular classes of graphs, two examples being bipartite graphs and split graphs. He further shows the conjecture to be true when relativized to uniformly k-sparse graphs and interval graphs. Another line of work addresses variants of Seese's conjecture obtained by considering logics other than MSO. One such result by Seese [20] shows that guarded second-order logic (GSO) is undecidable on any class of unbounded clique-width. Similarly, Courcelle and Oum [9] show that the extension C_2MSO of MSO obtained by considering modulo 2 counting quantifiers is also undecidable on classes of unbounded clique-width. In all of these cases, the proof goes via interpreting grids in unbounded clique-width classes. There has also been interesting progress looking at Seese's conjecture for structures other than graphs. A significant paper here is by Hliněný and Seese [12] who show the conjecture to be true for matroids representable over any finite field.

2 Preliminaries

The graphs we consider in this paper are simple, undirected and loop-free. For a graph G, we write $V(G)$ for the vertices of G and $E(G)$ for the edges. A graph H is an *induced subgraph* of G if $V(H) \subseteq V(G)$ and for any $x, y \in V(H)$, $\{x, y\} \in E(H)$ if, and only if, $\{x, y\} \in E(G)$. We write $H \subseteq G$ to denote that H is an induced subgraph of G. A class of graphs is said to be *hereditary* if it is closed under induced subgraphs. For any graph class $\mathscr{C}$, we write $\mathscr{C}\downarrow$ to denote the hereditary closure of $\mathscr{C}$ – that is, the class of graphs that are induced subgraphs of the graphs in $\mathscr{C}$. We consider monadic second-order logic (MSO) over vocabularies τ containing the binary relation E and finitely many unary relation symbols. A τ-structure can be thought of as an *expansion* of a graph $G = (V, E)$ with unary relations that interpret the unary symbols in τ. Such a structure can be thought of as a vertex-coloured graph. An MSO formula over the vocabulary τ is an expression that is inductively constructed from atomic MSO formulae using the Boolean connectives $\wedge, \vee$, and $\neg$, and existential quantification over vertex variables and set variables. Here an atomic MSO formula is an expression of the form $E(x, y)$ or $Q(x)$ or $X(y)$ or $x = y$ where x, y are vertex variables, the predicates E, Q belong to τ and X is a set variable. A *first order*, or FO, formula is an MSO formula that does not contain any set variable. We often write $\varphi(\bar{x}, \bar{X})$ to denote a formula whose free variables are among $\bar{x}$ and $\bar{X}$, the former being a tuple of vertex variables and the latter a tuple of set variables. Given such a formula, and a graph G along with a tuple $\bar{a}$ of vertices that interprets $\bar{x}$ and a tuple $\bar{A}$ of unary relations that interprets $\bar{X}$, we write $(G, \bar{A}) \models \phi[\bar{a}]$ to denote that the formula ϕ is satisfied in G in this interpretation.

Given a graph G and an MSO formula $\varphi(\bar{x}, \bar{X})$ where the length of $\bar{x}$ is k, we can think of φ as defining a k-ary relation on an expansion of G with an interpretation $\bar{A}$ of $\bar{X}$. Specifically this relation, denoted $\varphi^{(G,\bar{A})}$, is given by $\varphi^{(G,\bar{A})} = \{\bar{a} \mid (G, \bar{A}) \models \varphi[\bar{a}]\}$. Given a sequence $\bar{Z}$ of set variables, an MSO *graph interpretation with parameters* $\bar{Z}$ is a pair $\Psi(\bar{Z}) = (\psi_V(x, \bar{Z}), \psi_E(x, y, \bar{Z}))$ of MSO formulas over the vocabulary $\{E\} \cup \{Z_i \mid Z_i$ is an element of $\bar{Z}\}$. Given a graph G together with unary relations $\bar{A}$ interpreting the set variables $\bar{Z}$ in G, the interpretation $\Psi(\bar{Z})$ defines a possibly directed graph $H = \Psi(G, \bar{A})$.

This graph has (i) vertex set $\psi_V^{(G,\bar{A})}$, and (ii) edge set $\psi_E^{(G,\bar{A})}$. In this paper, we are only interested in the case where $\Psi(\bar{Z})$ defines an undirected graph (that is, $\psi_E(x, y, \bar{Z})$ defines an irreflexive and symmetric binary relation). Thus $\Psi(\bar{Z})$ defines a function from the expansions of graphs with $|\bar{Z}|$ unary predicates, to graphs, and therefore in general defines a relation on graphs. Where it causes no confusion, we also refer to the relation defined by an interpretation as an MSO interpretation. If $\bar{Z}$ is emplty, we call the interpretation Ψ *parameterless*, and such a Ψ defines a function from graphs to graphs. An example of a parameterless interpretation is $\Theta = (\theta_V(x), \theta_E(x, y))$ where $\theta_V(x) := (x = x)$ and $\theta_E(x, y) := \neg E(x, y)$; the function it defines maps a graph to its complement. An example of an interpretation with parameters is $\Gamma(Z) = (\gamma_V(x, Z), \gamma_E(x, y, Z))$ where $\gamma_V(x, Z) := Z(x)$ and $\gamma_E(x, y, Z) := E(x, y)$. The function that it defines maps an expansion (G, A) of a graph G to the subgraph of G induced by A; thus the relation on graphs that $\Gamma(Z)$ defines maps a graph to all its induced subgraphs. Given a class $\mathscr{C}$ of graphs and an interpretation Ψ with parameters $\bar{Z}$, we denote by $\Psi(\mathscr{C})$ the class of graphs given by $\Psi(\mathscr{C}) = \{\Psi(G, \bar{A}) \mid G \in \mathscr{C}$ and $\bar{A}$ is an interpretation of $\bar{Z}$ in $G\}$. For example, for the interpretation Γ above and a class $\mathscr{C}$ of graphs, the class $\Gamma(\mathscr{C})$ is exactly the hereditary closure of $\mathscr{C}$. Since they are relations, one can compose interpretations and it is known that the class of MSO interpretations is closed under composition [13]. We call MSO interpretations with parameters simply MSO interpretations for ease of readability, and denote them with the uppercase Greek letters $\Phi, \Gamma, \Psi, \Theta$, etc.

The notion of clique-width is a structural parameter of graphs that was introduced by Courcelle, Engelfriet and Rozenberg in [8] as a generalization of the well-known notion of treewidth. We do not give the definitions of clique-width and treewidth here as we need only specific properties of these for our results that we state below; we point the reader to [7, 17] for more about the notions and results concerning them. We write $\mathrm{cwd}(G)$ and $\mathrm{twd}(G)$ for the clique-width and tree-width of a graph G, respectively. As examples, a clique has clique-width 1, and a cograph has clique-width 2. It is known for any graph G that $\mathrm{cwd}(G) \leq 4 \cdot 2^{\mathrm{twd}(G)-1} + 1$ [10] and for planar G we even have $\mathrm{cwd}(G) \leq 6\mathrm{twd}(G) - 2$ [6]. A class of graphs is said to have *bounded* clique-width if for some number $k \geq 1$, every graph in the class has clique-width at most k. Thus, the class of cliques, the class of cographs and all classes of bounded treewidth have bounded clique-width. A graph class has *unbounded* clique-width if it does not have bounded clique-width. Examples of graph classes of unbounded clique-width include grids, interval graphs, and line graphs [5].

The class of all graphs of clique-width at most k is hereditary since the clique-width of an induced subgraph of G is never more than the clique-width of G. An *antichain* under the induced subgraph relation is a set $\mathcal{A}$ of graphs such that if G and H are distinct graphs in $\mathcal{A}$, then neither of $G \subseteq H$ or $H \subseteq G$ holds. Usually when we say "antichain" without further qualification, we mean an antichain under the induced subgraph relation. A graph class $\mathscr{C}$ is said to be *well-quasi-ordered* (WQO) under induced subgraphs if it does not contain any infinite antichains. For example, the class of all cliques is WQO under induced subgraphs.

The MSO theory of a graph class $\mathscr{C}$ is the class of all MSO sentences that are true in all graphs of $\mathscr{C}$. This theory is decidable if, and only if, the following problem is decidable: given an MSO sentence ϕ decide if ϕ is true in some graph in $\mathscr{C}$. Seese's conjecture states any class whose MSO theory is decidable has bounded clique-width. An $m \times n$ grid is a graph $G = (V, E)$ on $m \cdot n$ vertices with $V = \{(i, j) \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ and $E = \{\{(i, j), (i, j + 1)\} \mid 1 \leq i \leq m, 1 \leq j < n\} \cup \{\{(i, j), (i + 1, j)\} \mid 1 \leq i < m, 1 \leq j \leq n\}$. The grid is *square* if $m = n$. We say a class $\mathscr{C}$ of graphs *interprets grids* via an MSO interpretation Φ, if $\Phi(\mathscr{C})$ contains graphs isomorphic to arbitrarily large square grids. Any class of graphs that contains arbitrarily large grids has undecidable MSO theory [7, Thm. 5.6].

Morover, since MSO decidability is preserved by interpretations [7, Thm. 7.54], any class of graphs that interprets grids via an MSO interpretation has an undecidable MSO theory. The *strong Seese conjecture* is that any class of unbounded clique-width interprets grids via an MSO interpretation. It is known that if the clique-width of a class $\mathscr{C}$ is bounded and Φ is an MSO interpretation, then the clique-width of $\Phi(\mathscr{C})$ is also bounded [7, Cor. 7.38]. A simple observation about classes interpreting grids is the following.

▶ **Proposition 1.** *Suppose $\mathscr{C}$ is a graph class that interprets grids, and $\mathscr{D}$ is a graph class for which there exists an MSO interpretation Ξ such that the hereditary closure of $\Xi(\mathscr{D})$ contains $\mathscr{C}$. Then $\mathscr{D}$ interprets grids as well.*

Specifically, if Θ is the interpretation mapping $\mathscr{C}$ to a class containing arbitrarily large grids, and Γ is the interpretation defined above taking any class to its hereditary closure, then an interpretation Ω such that $\Omega(\mathscr{D})$ contains arbitrarily large square grids, is given by $\Omega = \Theta \circ \Gamma \circ \Xi$ (viewing Θ, Γ and Ξ as functions) where $\circ$ denotes composition.

We say that a class of graphs $\mathscr{C}$ is HUCW if it is hereditary and has unbounded clique-width. An HUCW graph class is said to be *minimal* if it does not contain a proper subclass that is HUCW. For example, bipartite permutation graphs and unit interval graphs are two minimal HUCW graph classes [15]. The existence of countably many minimal HUCW classes is established in [3], and this has been recently extended to uncountably many minimal HUCW classes in [2].

3 Minimal Classes and Well-Quasi-Ordering

In this section we lay out an approach to studying Seese's conjecture that motivates our study of MSO decidability for minimal HUCW classes. The first observation is that, if $\mathscr{C}$ is a counter-example to Seese's conjecture, then so is $\mathscr{C}{\downarrow}$. Recall that a counter-example to Seese's conjecture would be a class $\mathscr{C}$ that has unbounded clique-width and a decidable MSO theory. Clearly if $\mathscr{C}$ has unbounded clique-width, then so does $\mathscr{C}{\downarrow}$. The following proposition is folklore. It follows immediately from the fact that MSO decidability is preserved by interpretations and the existence of the interpretation Γ defined above which takes a class to its hereditary closure.

▶ **Proposition 2.** *If the* MSO *theory of $\mathscr{C}$ is decidable, then so is the* MSO *theory of $\mathscr{C}{\downarrow}$.*

Hence, if there is a counter-example to Seese's conjecture, we have one that is a hereditary class of unbounded clique-width, i.e. an HUCW class. In the present section, we establish some basic facts about the HUCW classes that allow us to structure the search for such a counter-example, or indeed the attempt to show that there is none.

The relation of being an induced subgraph is not a well-quasi-order as it admits infinite anti-chains. As an example, let I_n be the graph on $n+4$ vertices $e_0, e_1, e_2, e_3, c_1, \ldots, c_n$ where for each $i < n$ there is an edge between c_i and c_{i+1}, and in addition we have edges $e_0 - c_1$, $e_1 - c_1$, $e_2 - c_n$ and $e_3 - c_n$. In short, there is a path of length n with two additional vertices at each end to mark the ends. Then, it is clear the collection $(I_n)_{n\in\mathbb{N}}$ is an antichain in the induced subgraph order. This particular antichain has bounded clique-width. It is also possible to construct antichains of unbounded clique-width (which therefore must be infinite). An example is obtained by taking the collection of $n \times n$ grids and adding two extra vertices at each corner to form a triangle. In what follows, whenever we refer to an *antichain* we mean one under the induced subgraph relation.

From an antichain of unbounded clique-width, it is possible to construct (as we show below) an infinite descending chain of classes of graphs (under the inclusion relation) all of which are HUCW. Thus, it was a significant discovery to find that there are actually HUCW classes $\mathscr{C}$ that are *minimal*: no proper hereditary subclass of $\mathscr{C}$ has unbounded clique-width. The first such example is due to Lozin [15]. Collins et al. [3] constructed an infinite family of such classes and Lozin et al. [16] give an example that is itself well-quasi-ordered under the induced substructure relation. We examine these in some detail in subsequent sections.

If it were the case that every class that is HUCW contains as a subclass a minimal HUCW class, then showing that every minimal HUCW class interprets grids would suffice to prove Seese's conjecture. Indeed, if $\mathscr{C}$ interprets grids of unbounded size, so does every class that contains $\mathscr{C}$. However, Korpelainen has shown [14] that there are HUCW classes that contain no minimal HUCW subclass. We give a construction of such a class in Section 3.2. This is linked to the existence of antichains of unbounded clique-width. Specifically, we establish the following facts.

1. If $\mathscr{C}$ is a minimal HUCW class, then it cannot contain an antichain of unbounded clique-width (Theorem 6 in Section 3.1).
2. If $\mathscr{C}$ is an HUCW class that contains no minimal class, it must contain an antichain of unbounded clique-width (Theorem 7 in Section 3.1).

From these, the theorem below follows, which suggests a programme for proving Seese's conjecture.

▶ **Theorem 3.** *The strong Seese conjecture holds if, and only if, both of the following are true:*

1. *every antichain of unbounded clique-width interprets grids; and*
2. *every minimal* HUCW *class interprets grids.*

3.1 Antichains and Minimal Classes

We first establish the relationship between the existence of antichains of unbounded clique-width and the minimality of HUCW classes. These are established in Theorems 6 and 7.

We say that a sequence $(\mathscr{C}_i)_{i\in\omega}$ is an infinite *strictly descending* HUCW*-chain* if for each i, $\mathscr{C}_i$ is an HUCW class and $\mathscr{C}_{i+1}$ is a proper subclass of $\mathscr{C}_i$. We say that $\mathscr{C}$ contains an infinite strictly descending HUCW-chain if there is such a chain with $\mathscr{C}_i \subseteq \mathscr{C}$ for all i.

▶ **Lemma 4.** *The following are equivalent:*

1. *$\mathscr{C}$ contains an infinite strictly descending* HUCW*-chain whose intersection is a class of bounded clique-width.*
2. *$\mathscr{C}$ contains an infinite strictly descending* HUCW*-chain whose intersection is empty.*
3. *$\mathscr{C}$ contains an antichain of unbounded clique-width.*

Proof.

3 ⇒ 2. If $\{G_1, G_2, \ldots\}$ is such an antichain, then let $\mathscr{C}_i$ be the hereditary closure of $\{G_i, G_{i+1}, \ldots\}$ for $i \geq 1$. Then $\mathscr{C}_1 \supsetneq \mathscr{C}_2 \supsetneq \ldots$ is an infinite strictly descending HUCW-chain whose intersection is empty.

2 ⇒ 1. Trivial since the empty class has clique-width 0.

1 ⇒ 3. Let $\mathscr{C}_1 \supsetneq \mathscr{C}_2 \supsetneq \ldots$ be such a descending HUCW-chain and $\mathscr{C}_\omega = \bigcap_{i\geq 1} \mathscr{C}_i$. Let $\mathscr{D}_i = \mathscr{C}_i \setminus \mathscr{C}_{i+1}$ for $i \geq 1$. Then for $1 \leq i < j$, we have $\mathscr{D}_i \cap \mathscr{C}_j = \emptyset$; hence $\mathscr{D}_i \cap \mathscr{D}_j = \mathscr{D}_i \cap \mathscr{C}_\omega = \emptyset$. Further, $\mathscr{C}_i = \left(\biguplus_{i\leq k<\omega} \mathscr{D}_k\right) \uplus \mathscr{C}_\omega$.

▷ Claim 5. The following are true:

1. For $1 \leq i < j$, no graph in $\mathscr{D}_i$ is an induced subgraph of a graph in $\mathscr{D}_j$.
2. For $i \geq 1$, for every graph $G \in \mathscr{D}_i$, there exists a number $f(G) > i$ such that for all $j \geq f(G)$, no graph in $\mathscr{C}_j \setminus \mathscr{C}_\omega$ is an induced subgraph of G.

Proof.

1. If $G \subseteq H$ for some $G \in \mathscr{D}_i$ and $H \in \mathscr{D}_j$, then since $\mathscr{D}_j \subseteq \mathscr{C}_j$ and $\mathscr{C}_j$ is hereditary, we would have $G \in \mathscr{C}_j$; but that contradicts the fact that $\mathscr{D}_i \cap \mathscr{C}_j = \emptyset$.
2. Let $H_1, \ldots, H_r$ be an enumeration of the induced subgraphs of G that are not in $\mathscr{C}_\omega$ – clearly r is finite since G is finite. Since $\mathscr{C}_i = \left(\biguplus_{i \leq j < \omega} \mathscr{D}_j\right) \uplus \mathscr{C}_\omega$, there exist numbers $j_1, \ldots, j_r \in [i, \omega)$ such that $H_i \in \mathscr{D}_{j_i}$ for $i \in \{1, \ldots, r\}$. It then follows by the properties of the $\mathscr{D}_i$'s above that $f(G) = \max\{j_i \mid 1 \leq i \leq r\} + 1$ is indeed as desired. ◁

We now use the above claim to inductively construct an antichain of $\mathscr{C}$ of unbounded clique-width. Let G_0 be a graph in $\mathscr{D}_0$. Assume that we have constructed graphs $G_0, \ldots, G_i$ for $i \geq 0$ such that (i) $G_j \in \mathscr{D}_{l_j}$ and $l_j > l_{j-1}$ for $1 \leq j \leq i$; (ii) $\{G_0, \ldots, G_i\}$ is an antichain; and (iii) the clique-width of G_j is strictly greater than that of G_{j-1} for $1 \leq j \leq i$. Let $k = \max\{f(G_j) \mid 1 \leq j \leq i\} > l_i$ where f is as in Claim 5. Consider the class $\mathscr{C}_k \setminus \mathscr{C}_\omega$ – by Lemma 5, all graphs in this class are incomparable with each of $G_0, \ldots, G_i$ in the induced subgraph order. Further, since $\mathscr{C}_k$ has unbounded clique-width while $\mathscr{C}_\omega$ has bounded clique width, we have that $\mathscr{C}_k \setminus \mathscr{C}_\omega$ has unbounded clique-width, whereby there exists $G_{i+1} \in \mathscr{C}_k \setminus \mathscr{C}_\omega$ such that G_{i+1} has clique-width greater than that of G_i. Let $l_{i+1} \geq k > l_i$ be such that $G_{i+1} \in \mathscr{D}_{l_{i+1}}$. Then we see that G_{i+1} is indeed as desired to complete the induction. ◀

We are now ready to prove the two results linking minimality of HUCW classes and the existence of antichains of unbounded clique-width.

▶ **Theorem 6.** *If $\mathscr{C}$ is a minimal* HUCW *class, then $\mathscr{C}$ does not contain an antichain of unbounded clique-width.*

Proof. If $\mathscr{C}$ contains an antichain of unbounded clique-width, then by Lemma 4, $\mathscr{C}$ contains an infinite strictly descending HUCW-chain, and hence in particular a proper subclass that is HUCW. Hence $\mathscr{C}$ is not minimal. ◀

▶ **Theorem 7.** *If $\mathscr{C}$ is* HUCW *and does not contain a minimal* HUCW *class, then there exists in $\mathscr{C}$ an antichain of unbounded clique-width.*

Proof. We assume without loss of generality that the vertices of the graphs of $\mathscr{C}$ belong to the set $\mathbb{N}$ of natural numbers, so that $\mathscr{C}$ is countable. Suppose that $\mathscr{C}$ does not contain a minimal class. Consider the sequence $(\mathscr{C}_\lambda)_{\lambda \geq 0}$ of classes of structures, for ordinals λ, defined inductively as follows. Let $\mathscr{C}_0 = \mathscr{C}$ and inductively, assume that for all $\nu < \lambda$, the class $\mathscr{C}_\nu$ has been defined and that $\mathscr{C}_\nu \subseteq \mathscr{C}$ for all $\nu < \lambda$. If λ is a limit ordinal, define $\mathscr{C}_\lambda = \bigcap_{\nu < \lambda} \mathscr{C}_\nu$. If λ is a successor ordinal of say λ^-, then define $\mathscr{C}_\lambda$ as follows. If $\mathscr{C}_{\lambda^-}$ is not HUCW, then $\mathscr{C}_\lambda = \mathscr{C}_{\lambda^-}$. Otherwise $\mathscr{C}_{\lambda^-}$ is HUCW and $\mathscr{C}_{\lambda^-} \subseteq \mathscr{C}$; then $\mathscr{C}_{\lambda^-}$ cannot be minimal since by our premise, $\mathscr{C}$ does not contain any minimal HUCW class. Let $\mathscr{C}_\lambda$ be any proper subclass of $\mathscr{C}_{\lambda^-}$ that is HUCW. This completes the construction of the sequence $(\mathscr{C}_\lambda)_{\lambda \geq 0}$.

Consider now the set $\mathcal{P}$ of ordinals defined as $\mathcal{P} = \{\lambda \mid \mathscr{C}_\lambda \text{ is not HUCW}\}$. This set is non-empty – since $\mathscr{C}$ is a class of finite graphs whose vertices are natural numbers, $\mathscr{C}$ is countable and hence $\mathscr{C}_\lambda = \emptyset$ for all uncountable λ. By the definition above, if $\lambda \in \mathcal{P}$, then all ordinals greater than λ are in $\mathcal{P}$ as well. Now since the ordinals are well ordered, $\mathcal{P}$ has a minimum, call it μ^*. We make the following observations about μ^*:

1. μ^* must be a limit ordinal. If it is a successor ordinal of say λ, then $\mathscr{C}_\lambda$ must be HUCW since μ^* is the minimum ordinal in $\mathcal{P}$. But if $\mathscr{C}_\lambda$ is HUCW, then $\mathscr{C}_{\mu^*}$ must be a HUCW class by the inductive definitions above. Therefore, $\mathscr{C}_{\mu^*} = \bigcap_{\nu<\mu^*} \mathscr{C}_\nu$ where $\mathscr{C}_\nu$ is HUCW for all $\nu < \mu^*$.
2. μ^* is countable – this is because $\mathscr{C}$ is countable.
3. $\mathscr{C}_{\mu^*}$ is a hereditary class of bounded clique-width. Let $G \in \mathscr{C}_{\mu^*}$ and $H \subseteq G$. Then by (1) above, $G \in \mathscr{C}_\nu$ for all $\nu < \mu^*$. Since each $\mathscr{C}_\nu$ is hereditary, we have $H \in \mathscr{C}_\nu$ for all $\nu < \mu^*$. Then $H \in \mathscr{C}_{\mu^*}$. So $\mathscr{C}_{\mu^*}$ is hereditary. That $\mathscr{C}_{\mu^*}$ has bounded clique-width now follows from the fact that $\mathscr{C}_{\mu^*}$ is not HUCW.

Now since μ^* is countable, it has cofinality ω so that there exists an increasing function $f : \mathbb{N} \to \mu^*$ (where μ^* is seen as the set of ordinals less than μ^*) such that if $\mathscr{F}_i = \mathscr{C}_{f(i)}$ for $i \in \mathbb{N}$, then $\bigcap_{i\in\mathbb{N}} \mathscr{F}_i = \mathscr{C}_{\mu^*}$. We observe that $\mathscr{F}_1 \supsetneq \mathscr{F}_2 \supsetneq \ldots$ is an infinite strictly descending HUCW-chain in $\mathscr{C}$, whose intersection $\mathscr{C}_{\mu^*}$ is a class of bounded clique-width. It now follows by Lemma 4 that $\mathscr{C}$ contains an antichain of unbounded clique-width. ◀

The converse of Theorem 7 does not hold. That is to say, we can construct an HUCW class that both contains a minimal HUCW class and contains an antichain of unbounded clique-width. Indeed, if $\mathcal{C}_1$ is a minimal HUCW class and $\mathcal{C}_2$ the hereditary closure of an antichain of unbounded clique-width then clearly $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ has this property.

3.2 HUCW Classes which Contain No Minimal Class

Theorem 7 raises the obvious question of whether there exists any class $\mathscr{C}$ which is HUCW but does not contain a minimal HUCW class. The existence of such a class was demonstrated by Korpelainen [14]. Here we give a similar construction which we arrived at independently.

▶ **Theorem 8.** *There is an* HUCW *class* $\mathcal{T}$ *that does not contain any minimal* HUCW *class.*

It suffices to show that if $\mathcal{C}$ is any hereditary subclass of $\mathcal{T}$ of unbounded clique-width, it contains an antichain of unbounded clique-width.

Towards this, let $G_{n,n}$ denote the $n \times n$ grid. Note that, in $G_{n,n}$, every vertex has degree 2, 3 or 4, and there are exactly four vertices (at the corners) of degree 2. For $n \geq 3$, we define T_n as the graph obtained from $G_{n,n}$ by:

1. removing every vertex v of degree 2 and inserting an edge between the two neighbours of v; and
2. replacing every vertex v of degree 4 by four new vertices v_1, v_2, v_3, v_4 that are connected in a 4-cycle so that the four edges incident on v are now each incident on one of the four new vertices.

It is easily seen that T_n is 3-regular, and it is more convenient to work with than grids. The number of vertices in T_n is less than $4n^2$.

Recall that a graph H is a *subdivision* of a graph G if it is obtained from G by replacing every edge by a simple path. For a positive integer t, we write G^t for the t-subdivision of G: the graph obtained from G by replacing each edge of G by a path of length t. We make the following simple observation for later use:

▶ **Lemma 9.** *If H is a subdivision of G and $twd(G) = k$, then $k \leq twd(H) \leq \max(k, 3)$.*

Proof. The lower bound on $\mathrm{twd}(H)$ follows immediately from the fact that G is a minor of H so $\mathrm{twd}(G) \leq \mathrm{twd}(H)$.

Suppose now that (T, β) is a tree decomposition of G of width k. To obtain a tree decomposition of H, consider an edge $\{u, v\}$ of G which is subdivided into a path $u = p_0, \ldots, p_t = v$ in H. As $\{u, v\}$ is an edge of G, there must be a node a of T such that $\{u, v\} \subseteq \beta(t)$. We attach a path $a_1, \ldots, a_t$ of length t to a and let $\beta(a_i) = \{u, v, p_i, p_{i+1}\}$. Doing this for each edge gives us a tree decomposition of H whose width is $\max(k, 3)$. ◀

Define the class $\mathcal{T} = \{H \mid H \subseteq T_n^n \text{ for some } n > 2\}$, i.e. the collection of graphs that are induced subgraphs of the n-subdivision of T_n for some n. We consider the graphs $H \in \mathcal{T}$ where every vertex has degree 2 or 3. We call such graphs *skeleton graphs* and the vertices of degree 3 the *branch vertices*. Note that every graph in $\mathcal{T}$ is an induced subgraph of a skeleton graph.

The next two lemmas establish some useful properties of the graphs in $\mathcal{T}$.

▶ **Lemma 10.** *If $H \in \mathcal{T}$ is a skeleton graph with at most $m > 2$ branch vertices, then $\mathrm{cwd}(H) \leq 6m - 2$.*

Proof. Since H has at most m branch vertices, it is the subdivision of some graph G with m vertices. Hence, by Lemma 9, the treewidth of H is at most m. Note further that all graphs in $\mathcal{T}$ are planar and hence H is planar. For any planar graph H, $\mathrm{cwd}(H) \leq 6\mathrm{twd}(H) - 2$ [6, Thm 17], and the result follows. ◀

▶ **Lemma 11.** *If H is a subdivision of T_n for $n > 2$, then the clique-width of H is at least $(n - 1)/6$.*

Proof. Since $G_{n-2,n-2}$ is a minor of T_n and $\mathrm{twd}(G_{k,k} = k$ we have that $\mathrm{twd}(T_n) \geq n - 2$. Also, by Lemma 9 we know that $\mathrm{twd}(H) = \mathrm{twd}(T_n)$. Now, for any planar graph G we have $\mathrm{twd}(G) \leq 6\mathrm{cwd}(G) - 1$ by [7, Prop. 2.115]. Since H is planar, the result follows. ◀

Proof of Theorem 8. The class $\mathcal{T}$ is hereditary by definition and has unbounded clique-width by Lemma 11. Thus, it remains to show that for every class $\mathcal{C} \subseteq \mathcal{T}$, if $\mathcal{C}$ has unbounded clique-width, then $\mathcal{C}$ contains an antichain of unbounded clique-width.

So, suppose $\mathcal{C} \subseteq \mathcal{T}$ has unbounded clique-width. For a graph $H \in \mathcal{T}$, write $\mathrm{mn}(H)$ for the length of the shortest path between two branch vertices of H. We define the following sequence of graphs. First, let G_0 be any graph in $\mathcal{C}$ containing at least two branch vertices. Suppose we have defined G_i for $i \geq 0$, and let $t = \max(\mathrm{cwd}(G_i), \mathrm{mn}(G_i))$. We then choose G_{i+1} to be any graph in $\mathcal{C}$ with $\mathrm{cwd}(G_{i+1}) > 24t^2 - 2$.

It is clear that the sequence of graphs $(G_i : i \in \omega)$ is of unbounded clique-width, since $\mathrm{cwd}(G_i) < \mathrm{cwd}(G_{i+1})$ for all i. We now argue that this is also an antichain. For any $i < j$, clearly G_j cannot be an induced subgraph of G_i since $\mathrm{cwd}(G_i) < \mathrm{cwd}(G_j)$, so it remains to show that G_i is not an induced subgraph of G_j. Since $\mathrm{cwd}(G_j) > 24t^2 - 2$, where $t = \max(\mathrm{cwd}(G_i), \mathrm{mn}(G_i))$, it follows by Lemma 10 that G_j has more than $4t^2$ branch vertices. Since T_n^n contains fewer than $4n^2$ branch vertices, it follows that G_j is not an induced subgraph of T_n^n for any $n \leq t$. Hence, $\mathrm{mn}(G_j)$ is at least $t + 1$. However, by the choice of t, $\mathrm{mn}(G_i) \leq t$ and so G_i contains two branch vertices at distance at most t. We conclude that G_i is not an induced subgraph of G_j. ◀

4 Grid-Like Classes

We begin our systematic exploration of all known minimal hereditary classes of unbounded clique-width. Many such classes are defined in terms of a grid-like structure and this is used to show that they have unbounded clique-width. The challenge in these cases is to show how this grid structure can be drawn out through an MSO interpretation. We begin with a collection of minimal HUCW classes (indeed, an uncountable collection of them) defined in terms of certain infinite words and show in Section 4.1 that they interpret grids. This is then extended by reductions in Section 4.2 to a number of other classes.

4.1 Word-defined minimal classes

Our starting point is a construction given by Brignall and Cocks [2] to demonstrate that there are uncountably many minimal HUCW classes, extending a construction by Collins et al. [3] showing the existence of infinitely many such classes. They construct a hereditary class $\mathscr{S}_\alpha$ of graphs for each ω-word $\alpha \in \{0,1,2,3\}^\omega$ and show that as long as α contains infinitely many non-zero letters, the class $\mathscr{S}_\alpha$ has unbounded clique-width. Moreover, for uncountably many distinct such α, $\mathscr{S}_\alpha$ is also minimal. The conditions under which $\mathscr{S}_\alpha$ is minimal need not concern us here. We are able to show that whenever α contains infinitely many non-zero letters $\mathscr{S}_\alpha$ interprets grids via MSO interpretations. In particular, this covers all minimal classes $\mathscr{S}_\alpha$ of unbounded clique-width, including those defined in [3]. Before we proceed to a proof, we give a precise definition of the classes $\mathscr{S}_\alpha$.

The class $\mathscr{S}_\alpha$ is defined as the class of all finite induced subgraphs of a single countably infinite graph $\mathcal{P}_\alpha$. The set of vertices of $\mathcal{P}_\alpha$ is $\{v_{i,j} \mid i,j \in \mathbb{N}\}$. We think of the set as an infinite collection of *columns* $V_j = \{v_{i,j} \mid i \in \mathbb{N}\}$. All edges are between vertices in adjacent columns, i.e. there is no edge between $v_{i,j}$ and $v_{i',j'}$ unless $j' = j+1$ or $j' = j-1$. The edges between successive columns are defined by the word α according to the following rules.

1. If $\alpha_j = 0$, then $\{v_{i,j}, v_{k,j+1}\} \in E(\mathcal{P}_\alpha)$ if, and only if, $i = k$.
2. If $\alpha_j = 1$, then $\{v_{i,j}, v_{k,j+1}\} \in E(\mathcal{P}_\alpha)$ if, and only if, $i \neq k$ for $i,k \in \mathbb{N}$.
3. If $\alpha_j = 2$, then $\{v_{i,j}, v_{k,j+1}\} \in E(\mathcal{P}_\alpha)$ if, and only if, $i \leq k$ for $i,k \in \mathbb{N}$.
4. If $\alpha_j = 3$, then $\{v_{i,j}, v_{k,j+1}\} \in E(\mathcal{P}_\alpha)$ if, and only if, $i \geq k$ for $i,k \in \mathbb{N}$.

The class $\mathscr{S}_\alpha$ is now given by $\mathscr{S}_\alpha = \{G \mid G \text{ is a finite induced subgraph of } \mathcal{P}_\alpha\}$. We show the following theorem in this section.

▶ **Theorem 12.** *Let $\alpha \in \{0,1,2,3\}^\omega$ be such that α contains infinitely many non-zero letters. Then there exists an* MSO *interpretation Θ such that $\Theta(\mathscr{S}_\alpha)$ contains the class of all square grids.*

To prove Theorem 12, we show the existence of an MSO interpretation Ψ such that the hereditary closure of $\Psi(\mathscr{S}_\alpha)$ contains the class of all square grids. Proposition 1 ensures that this indeed suffices. It is clear that graphs in $\mathscr{S}_\alpha$ have a built-in grid-like structure with vertices arranged in rows and columns. The main challenge is to show that a sufficient part of this structure can be made explicit using an MSO interpretation. We give an outline of the construction.

What we show is that we can find in S_α a sequence of graphs G_n for $n \in \mathbb{N}$ within which we can interpret *upper triangular grids*. One can think of an upper triangular grid U_t as the subgraph of the $t \times t$ grid induced by the vertices above the main diagonal, i.e. those vertices in the set $\{(i,j) \mid 1 \leq i,j \leq t\}$ with $i \leq j$. It is clear that U_t has as an induced subgraph an $r \times r$ grid, where $r = \lfloor \frac{t}{2} \rfloor$.

Let $\alpha \in \{0,1,2,3\}^\omega$ be an ω-word containing infinitely many non-zero letters. We write α_i for the i^{th} letter of α. Let $p < \omega$ be the least value such that $\alpha_p \neq 0$. Fix $n \geq 1$ and let l be the length of the shortest contiguous subsequence of α starting at α_p that contains exactly $2n+2$ elements which are not 0. We write $\beta_0 \cdots \beta_{l-1}$ for this sequence, so $\beta_0 = \alpha_p$.

Recall that the vertices of P_α are $\{v_{i,j} \mid i,j \in \mathbb{N}\}$, and we write V_j for the set $\{v_{i,j} \mid i \in \mathbb{N}\}$. We define the graph G_n to be the subgraph of P_α induced by the set $\mathrm{C} = \bigcup_{i=0}^{i=l-1} \mathrm{C}_i$ where $\mathrm{C}_i \subseteq V_{p+i}$ is defined as follows for $0 \leq i < l$.

1. $\mathrm{C}_0 = \{v_{0,p}, v_{1,p}, v_{2,p}, v_{3,p}\}$; and
2. $\mathrm{C}_{i+1} = \{v_{0,p+i+1}, v_{1,p+i+1}, \ldots, v_{t-1,p+i+1}\}$ where $t = |\mathrm{C}_i|$ if $\beta_{i+1} = 0$ and $t = |\mathrm{C}_i| + 1$ otherwise.

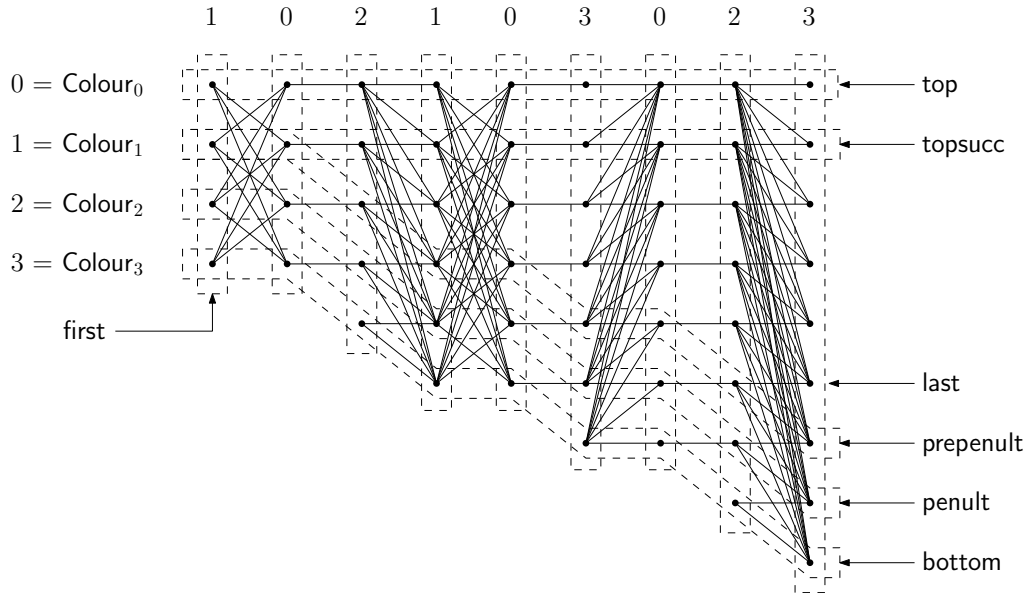


Figure 1 The graph H_2 for $\alpha = (102103023)^\omega$. The unlabeled graph underlying H_2 is G_2.

It is clear that $G_n \in \mathscr{S}_\alpha$. We show that we can interpret upper triangular grids in this class of graphs. The key challenge in defining the required interpretation is to define the two binary relations: one that relates vertices that are in the same column and the other that relates vertices that are in the same row. In constructing the interpretation we make use of a number of set parameters to obtain a labeled version H_n of G_n as illustrated in Figure 1. In particular, H_n uses unary predicates for the vertices corresponding to the possible values of β_i, for the first and last column, the top, bottom and penultimate rows, and the rows immediately succeeding and preceding the top and penultimate rows respectively. The "diagonal" nature of the bottom row is vital to allowing us to define when two vertices are in successive columns, which we need in order to define the two relations of being in the same row and in the same column.

4.2 Composing Interpretations

We now consider the classes of graphs shown to be minimal HUCW in [15, 1], and prove that these interpret grids using Theorem 12 above. Specifically, we show that for each class $\mathscr{C}$ among them, there is some $\alpha \in \{0,1,2,3\}^\omega$ and an MSO interpretation Ξ such that the hereditary closure of $\Xi(\mathscr{C})$ contains $\mathscr{S}_\alpha$; then $\mathscr{C}$ interprets grids by Proposition 1.

▶ **Theorem 13.** *The following minimal* HUCW *classes of graphs interpret grids:*
1. *Bichain graphs*
2. *Split permutation graphs*
3. *Bipartite permutation graphs*
4. *Unit interval graphs*

▶ Remark 14. We mention that Theorem 13(4) follows from the results of Courcelle in [5]. It is shown in [5] that Seese's conjecture holds for the class of interval graphs. More specifically, it can be inferred from the results in [5] that any unbounded clique-width subclass of interval graphs admits MSO interpretability of grids. It follows, in particular, that this is true of the unit interval graphs. We therefore show parts (1)–(3) of Theorem 13 to complete its proof.

Bichain graphs

We need some terminology to talk about these graphs. Given a graph G, a sequence $v_1, \ldots, v_k$ of vertices of G is said to be a *chain* if $N(v_i) \subseteq N(v_j)$ whenever $i \leq j$, where $N(v) := \{u \mid E(u, v)\}$ denotes the neighbourhood of v. A bipartite graph $(A \cup B, E)$ is called a *k-chain graph* if each of the two parts A and B can be further partitioned into at most k chains. A *bichain graph* is a 2-chain graph.

We now describe the bichain graph Z_n as defined in [1]. This graph is *n-universal* in that all bichain graphs on at most n vertices are induced subgraphs of Z_n. The graph has vertex set $\{z_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq n\}$ (which can thus be seen as an $n \times n$ grid of points), and $\{z_{i,j}, z_{i',j'}\}$ is an edge if, and only if, one of the following holds: (i) j is odd, $j' = j + 1$ and $i < i'$; (ii) j is even, $j' = j + 1$ and $i' \leq i$; or (iii) j is even, j' is odd and $j' \geq j + 3$. Since bichain graphs are hereditary, it follows that the class Bichain of all bichain graphs is exactly the hereditary closure of the class $\{Z_n \mid n \geq 1\}$.

Again, the grid structure is implicit in the graph Z_n. What we show is that when Z_n is expanded with unary relations for the bottom row $\{z_{n,j} \mid 1 \leq j \leq n\}$ and the last column $\{z_{i,n} \mid 1 \leq i \leq n\}$, we can construct an FO interpretation to a class that contains the class $\mathscr{S}_\alpha$ for $\alpha = (23)^\omega$ in its hereditary closure.

Split permutation graphs

Recall that a *split graph* is a graph G whose vertex set can be partitioned into two sets C and I such that C induces a clique in G and I is an independent set in G. A *permutation graph* is a graph whose vertices represent the domain of a permutation, and each of whose edges determines an inversion in the permutation. Following [1], we use the following characterization of split permutation graphs.

▶ **Proposition 15** ([1, Prop. 2.3])**.** *Let G be a split graph given together with a partition of its vertex set into a clique C and an independent set I. Let H be the bipartite graph obtained from G by deleting the edges of C. Then G is a split permutation graph if, and only if, H is a bichain graph.*

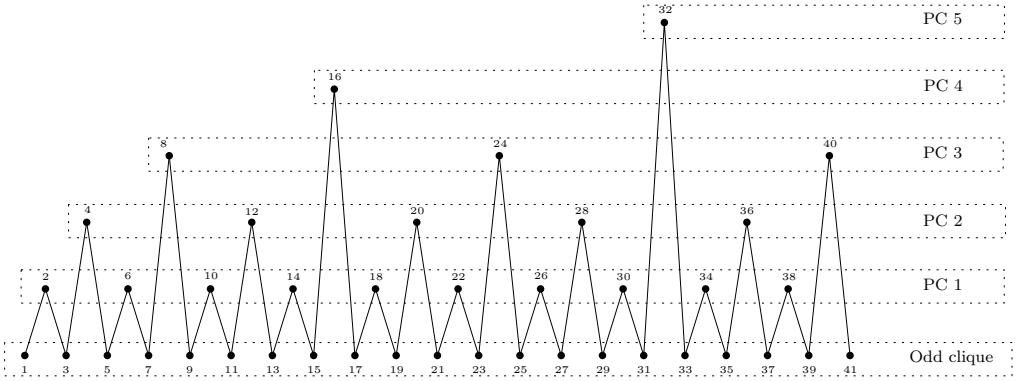
Let G be a split permutation graph with (C, I) being a partition of its vertex set into a clique C and an independent set I. Let G^* be the expansion of G with a unary predicate P which is interpreted as the set C. Consider the FO interpretation Ψ which removes from G^* all edges inside P. It is easy to see that $\Psi(G^*)$ is a bichain graph by Proposition 15.

Let Ψ be the FO interpretation as described above and SP be the class of split permutation graphs. Then $\Psi(\mathsf{SP})$, and hence its hereditary closure, contains the class Bichain. We are then done by Theorem 13(1) and Proposition 1.

Bipartite permutation graphs

These graphs are graphs that are bipartite as well as being permutation graphs. For our purposes, the following characterization is useful. Consider the graph P_n on vertex set $\{v_{i,j} \mid 1 \leq i, j \leq n\}$ where the only edges are between $v_{i,j}$ and $v_{i+1,j'}$ for $j' \leq j$. Then, the class of bipartite permutation graphs is exactly the hereditary closure of the class $\{P_n \mid n \geq 1\}$ [15]. Now, it is easily seen that this class is exactly the class $\mathscr{S}_\alpha$ as described in Section 4.1, for $\alpha = 2^\omega$, and this has been observed in [3]. Thus, Theorem 13(3) follows from Theorem 12.

5 Power Graphs



D_{41}

Figure 2 The power graph D_{41} with "PC i" denoting the power clique corresponding to i.

In this section, we consider the class of *power graphs* as defined in [16] in the context of well-quasi-ordering and clique-width. Most of the classes that we have seen so far can be shown to not be well-quasi ordered under the induced subgraph relation. In particular, all word-defined classes, unit interval graphs and bipartite permutation graphs can be seen to contain the antichain $\{I_n \mid n \geq 1\}$ described after Proposition 2. We do not know whether bichain graphs and split permutation graphs are well-quasi ordered, though it has been shown that their expansion with two labels is not a well-quasi ordered class [1]. In contrast, power graphs constitute a class of graphs that is HUCW, that *is* well-quasi ordered [16] and, as we show, is a minimal HUCW class. It was introduced precisely to demonstrate an HUCW class that is well-quasi ordered. Minimality follows from arguments contained in [16], but was not observed there. We now define the class of power graphs. We show that they are minimal and then in the remainder of the section show that they admit interpretability of grids.

For $n \geq 1$, we define the graph D_n as follows. The vertex set of D_n is $[n] = \{1, \ldots, n\}$. For each $i < n$, there is an edge between i and $i + 1$ – we call these *path edges*. Furthermore, there is an edge between i and j if the largest power of 2 that divides i is the same as the largest power of 2 that divides j – we call these *clique edges*. To understand this terminology, note that we can see D_n as consisting of a simple path of length n, along with, for each k such that $2^k \leq n$, a clique on all vertices $j = 2^k \cdot (2r + 1)$ for some $r \geq 0$ – we call this the *power clique* corresponding to k. In particular, taking $k = 0$, there is a clique formed by all the odd elements, which we call the *odd clique*. Observe that the path edges, which are the

only edges with endpoints in different power cliques always have one end point in the odd clique, and one outside it. The class of power graphs, denoted Power-graphs, is now defined as the hereditary closure of the class $\{D_n \mid n \geq 1\}$.

5.1 Minimality of Power Graphs

▶ **Proposition 16.** *The class* Power-graphs *is a minimal hereditary class of unbounded clique-width.*

That Power-graphs is a hereditary class of unbounded clique-width has already been shown in [16]. Thus, we only need to show that no proper subclass has this property.

Given a graph $G \in$ Power-graphs which is a subgraph of D_n, define an *interval* in G to be a set $S \subseteq [n]$ of vertices of G such that if $i, j \in S$ with $i < j$ and k is a vertex of G with $i < k < j$ then $k \in S$. We call a subgraph of G induced by an interval a *factor* of G. We now recall the following two results proved in [16].

▶ **Lemma 17** (Lemma 11, [16])**.** *Let G be a graph in* Power-graphs*. Then there exists an integer $t = t(G)$ such that for any $n \geq t$, every factor of D_n of length at least t contains G as an induced subgraph.*

▶ **Theorem 18** (Theorem 2, [16])**.** *Let G be a graph in* Power-graphs *such that the length of the longest factor in G is t. Then the clique-width of G is at most $2(\log t + 4)$.*

Proof of Proposition 16. Consider a proper hereditary subclass $\mathscr{S}$ of Power-graphs; then $\mathscr{S}$ excludes a graph $G \in$ Power-graphs. Let $t = t(G)$ be as given by Lemma 17. Let $\mathscr{S} = \mathscr{S}_1 \cup \mathscr{S}_2$ where $\mathscr{S}_1 = \mathscr{S} \cap \{D_n \mid n < t\}\downarrow$ and $\mathscr{S}_2 = \mathscr{S} \cap \{D_n \mid n \geq t\}\downarrow$. Observe that $\mathscr{S}_1$ has finitely many graphs up to isomorphism.

We show that for each $X \in \mathscr{S}_2$, every factor of X has length $< t$. For otherwise X has a factor Y of length $\geq t$ and there is $p \geq 1$ such that $X \subseteq D_p$ and so Y is also a factor of D_p. Hence by Lemma 17, we have G is an induced subgraph of Y, whereby it is also an induced subgraph of X. Since $\mathscr{S}$ is hereditary, $G \in \mathscr{S}$ which is a contradiction.

By Theorem 18, every $X \in \mathscr{S}_2$ has clique-width $\leq k = 2(\log t + 4)$. Then $\mathscr{S}_2$ has bounded clique-width, and hence so does $\mathscr{S}$ since $\mathscr{S}_1$ is finite. ◀

5.2 Interpreting grids in Power Graphs

We now establish the main result of this section, showing that power graphs do not provide a counter-example to Seese's conjecture.

▶ **Theorem 19.** *There exists an* MSO *interpretation Θ such that Θ(*Power-graphs*) contains all square grids.*

We show Theorem 19 by showing that there exists an MSO interpretation Φ such that the hereditary closure of Φ(Power-graphs) contains all bipartite permutation graphs. We are then done by Theorem 13 and Proposition 1. Indeed, it suffices to show that we can interpret grids in a subset of Power-graphs and we do this for the set $\{D_n \mid n \in \mathbb{N} \text{ even and } n > 9\}$.

We first show that there exists an FO formula $\mathsf{odd}(x)$ such that if x is a number in D_n with $n \geq 9$, then $\mathsf{odd}(x)$ is true if, and only if, x is an odd number. The formula asserts that there exist three elements y, z, w which together with x form a clique except that there is no edge $z - w$. It is easy to see that for $n \geq 9$, all odd numbers in D_n satisfy $\mathsf{odd}(x)$. If x is odd with $x < n - 3$, this is witnessed by $y = x + 2$, $w = x + 4$ and $z = x + 1$, otherwise by $y = x - 2$, $w = x - 4$ and $z = x - 1$.

To show that the even numbers of D_n do not satisfy $\mathsf{odd}(x)$, first observe that in any power clique other than the odd clique, since the numbers in the clique are of the form $2^k \cdot (2r+1)$ for fixed k, the difference between any two numbers in the clique is at least 2^{k+1}, which is at least 4 since $k \geq 1$. Suppose now that x is an even number in D_n and x, y, z form a 3-clique. We argue that any w that is adjacent to both x and y must also be adjacent to z showing that $\mathsf{odd}(x)$ is not satisfied. Consider the two cases:

- The edge between x and y is a clique edge. Then $|x - y| \geq 4$. If z is in a different power clique, then $|x - z| = 1$ and $|z - y| = 1$, whereby $|x - y| \leq 2$ – a contradiction. Thus z is in the same power clique as x and y. By the same argument, w is the same power clique as x and y, so there is a clique edge $z - w$.
- The edge between x and y is a path edge and so $|x - y| = 1$. Then the edges from z to x and y cannot both be path edges, as you cannot have a triangle of such edges. So, one of them is a clique edge. If z is in the same power clique as x, then $|x - z| \geq 4$ and $|y - z| = 1$, which is a contradiction, so z must be in the same power clique as y. By the same argument, w is in the same clique as y, so there is a clique edge $z - w$.

With the formula $\mathsf{odd}(x)$, we can distinguish path edges from clique edges. Indeed, an edge is a path edge if, and only if, it has exactly one end point that is odd. In D_n, the path edges form a simple path of length $n - 1$ and, if n is even, then only one of the two end points satisfies $\mathsf{odd}(x)$. This allows us to give this simple path an orientation: for each path edge $(x, x+1)$ we can identify the direction $x \to x+1$. The transitive closure of this relation (which is definable in MSO), gives us a definition of the natural linear order on D_n.

Once we have defined a linear order $\leq$ on D_n, this induces a linear order on the power cliques: namely, a clique C is below C' if the $\leq$-minimal element of C is less than the $\leq$-minimal element of C'. Indeed, we can also define a successor relation on cliques from this. From these, we define a relation that relates a pair x and y precisely if y occurs after x in the linear order $\leq$ and occurs in the power clique that is successor to the power clique containing x. It is easy to see that the graph induced by this relation contains arbitrarily large bipartite permutation graphs P_k as defined on page 13.

6 Conclusion

The study of monadic second-order logic on graphs has attracted great attention in recent years. An important aspect of work on this logic is to identify classes of graphs on which MSO is well behaved. Seese's conjecture is an important focus of this classification effort. In its stronger form it offers a dichotomy: any class of graphs is either interpretable in trees and therefore has bounded clique-width and is well-behaved *or* it interprets arbitrarily large grids and its MSO theory is then undecidable.

We show that Seese's conjecture could be established by considering two kinds of graph classes: minimal hereditary classes of unbounded clique-width and antichains of unbounded clique-width. Showing that all such classes interpret unbounded grids would suffice. While we do not have a complete taxonomy of such classes, we investigated all the ones known and showed that none of them provides a counter-example to Seese's conjecture.

One could weaken the strong conjecture by requiring only that the classes of unbounded clique-width admit MSO *transductions* of grids, rather than interpretations (see [7] for a discussion of transductions). This would still suffice to establish Seese's conjecture. In all the cases we consider, however, we establish the stronger form, i.e. an interpretation of grids.

It is also worth pointing out that for many of the classes we consider, the original proofs that they have unbounded clique-width require sophisticated bespoke arguments. The interpretation of grids in the classes also provides a uniform method of proving that they have unbounded clique-width.

As a final remark, it is worth noting that there are standard graph operations which allow us to construct new minimal HUCW graph classes from the ones we have. For example, taking the graph complement of all graphs in a class $\mathscr{C}$ yields a class that is also minimal HUCW if $\mathscr{C}$ is. Since this operation is itself an MSO interpretation, the results about interpreting arbitrarily large grids apply to the resulting classes as well.

References

1 Aistis Atminas, Robert Brignall, Vadim V. Lozin, and Juraj Stacho. Minimal classes of graphs of unbounded clique-width defined by finitely many forbidden induced subgraphs. *arXiv preprint*, 2015. arXiv:1503.01628.

2 Robert Brignall and Daniel Cocks. Uncountably many minimal hereditary classes of graphs of unbounded clique-width. *arXiv preprint*, 2021. arXiv:2104.00412.

3 Andrew Collins, Jan Foniok, Nicholas Korpelainen, Vadim Lozin, and Victor Zamaraev. Infinitely many minimal classes of graphs of unbounded clique-width. *Discrete Applied Mathematics*, 248:145–152, 2018.

4 Derek G Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

5 Bruno Courcelle. The monadic second-order logic of graphs XV: On a conjecture by D. Seese. *Journal of Applied Logic*, 4(1):79–114, 2006. doi:10.1016/j.jal.2005.08.004.

6 Bruno Courcelle. From tree-decompositions to clique-width terms. *Discret. Appl. Math.*, 248:125–144, 2018. doi:10.1016/j.dam.2017.04.040.

7 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

8 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.

9 Bruno Courcelle and Sang il Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese. *Journal of Combinatorial Theory, Series B*, 97(1):91–126, 2007. doi:10.1016/j.jctb.2006.04.003.

10 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

11 Robert Ganian, Petr Hliněný, and Jan Obdrzálek. Clique-width: When hard does not mean impossible. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10–12, 2011, Dortmund, Germany*, volume 9 of *LIPIcs*, pages 404–415. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.STACS.2011.404.

12 Petr Hliněnỳ and Detlef Seese. Trees, grids, and MSO decidability: From graphs to matroids. *Theoretical computer science*, 351(3):372–393, 2006.

13 Wilfrid Hodges. *Model theory*. Cambridge University Press, 1993.

14 Nicholas Korpelainen. A new graph construction of unbounded clique-width. *Electronic Notes in Discrete Mathematics*, 56:31–36, 2016.

15 Vadim V. Lozin. Minimal classes of graphs of unbounded clique-width. *Annals of Combinatorics*, 15(4):707–722, 2011.

16 Vadim V. Lozin, Igor Razgon, and Viktor Zamaraev. Well-quasi-ordering does not imply bounded clique-width. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 351–359. Springer, 2015.

17 Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

18 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

19 Neil Robertson and P.D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. Special Issue Dedicated to Professor W.T. Tutte. doi:10.1016/j.jctb.2004.08.001.

20 Detlef Seese. The structure of the models of decidable monadic theories of graphs. *Annals of pure and applied logic*, 53(2):169–195, 1991.

Constructive Many-One Reduction from the Halting Problem to Semi-Unification

Andrej Dudenhefner
Saarland University, Saarbrücken, Germany

Abstract

The undecidability of semi-unification (unification combined with matching) has been proven by Kfoury, Tiuryn, and Urzyczyn in the 1990s. The original argument is by Turing reduction from Turing machine immortality (existence of a diverging configuration).

There are several aspects of the existing work which can be improved upon. First, many-one completeness of semi-unification is not established due to the use of Turing reductions. Second, existing mechanizations do not cover a comprehensive reduction from Turing machine halting to semi-unification. Third, reliance on principles such as König's lemma or the fan theorem does not support constructivity of the arguments.

Improving upon the above aspects, the present work gives a constructive many-one reduction from the Turing machine halting problem to semi-unification. This establishes many-one completeness of semi-unification. Computability of the reduction function, constructivity of the argument, and correctness of the argument is witnessed by an axiom-free mechanization in the Coq proof assistant. The mechanization is incorporated into the existing Coq library of undecidability proofs. Notably, the mechanization relies on a technique invented by Hooper in the 1960s for Turing machine immortality.

An immediate consequence of the present work is an alternative approach to the constructive many-one equivalence of System F typability and System F type checking, compared to the argument established in the 1990s by Wells.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases constructive mathematics, undecidability, mechanization, semi-unification

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.18

Supplementary Material *Software (Source Code)*: `https://github.com/uds-psl/coq-library-undecidability/tree/coq-8.12/theories/SemiUnification`

Acknowledgements The author is grateful for encouragement and assistance by Paweł Urzyczyn and the members of the programming systems lab led by Gert Smolka at Saarland University.

1 Introduction

Semi-unification is the combination of first-order unification and first-order matching. That is, given a finite set of pairs of first-order terms, is there a valuation φ such that for each pair (σ, τ) in the set of first-order terms we have $\psi(\varphi(\sigma)) = \varphi(\tau)$ for some valuation ψ? Semi-unification naturally arises in type inference for functional and logic programs [30, 26, 19] which allow for polymorphic recursion [34]. Intuitively, the valuation φ establishes global code invariants, and the individual valuations ψ establish additional local conditions for each polymorphic function application. While both first-order unification and first-order matching are decidable problems, the status of semi-unification remained open throughout the 1980s, until answered negatively by Kfoury, Tiuryn, and Urzyczyn [25, 27]. The undecidability of semi-unification impacted programming language design and analysis with respect to polymorphic recursion [31, 21], loop detection [36], and data flow [12]. Another prominent result based on the undecidability of semi-unification is the undecidability of System F [18, 37] typability and type checking [39]. Of course, the negative result motivated the complementary line of work [31] in search for expressive, decidable fragments of semi-unification. A notable decidable fragment is *acyclic* semi-unification, used for standard ML typability [28].

Due to the importance of semi-unification in functional programming, it is natural to ask for *surveyable* evidence (both locally and globally in the sense of [4]) for its undecidability. The original undecidability proof [25] is quite sophisticated, and it was recently simplified [10] and partially mechanized (in the Coq proof assistant). Unfortunately, there are several aspects that obstruct surveyability of previous work.

First, existing arguments rely on the undecidability of Turing machine immortality, shown by Hooper [20]. The corresponding construction has received more attention [24], however, it was never published in full detail. Hooper remarks:

> *A routine and unimaginative analysis-of-cases proof would point this out more clearly; but it has remained unwritten since, as a rather tedious insult to the alert, qualified reader, it would surely remain unread.*

While the omissions are justified by accessibility, they hinder verification in full detail. The existing mechanization [10] of the undecidability of semi-unification does not improve upon this aspect, as it treats the undecidability of Turing machine immortality axiomatically.

Second, existing arguments use principles such as excluded middle, König's lemma [25], or the fan theorem [10] that do not support constructivity of the arguments. As a result, anti-classical theories, such as synthetic computability theory [5], may be in conflict with such constructions. The question arises, whether non-constructive principles are inherent to semi-unification or could be avoided.

Third, existing arguments use Turing reductions and are insufficient to establish many-one completeness of semi-unification. Hitherto, a many-one reduction from Turing machine halting to semi-unification is not given.

This work improves upon the above aspects as follows. It provides a comprehensive chain of many-one reductions from Turing machine halting to semi-unification, replacing immortality with uniform boundedness. Crucially, each many-one reduction is mechanized in full detail in the Coq proof assistant [8]. The mechanization witnesses correctness and constructivity of the argument. Specifically, the notion of a *constructive* proof is identified with an axiom-free Coq mechanization (cf. calculus of inductive constructions). It neither assumes functional extensionality (cf. homotopy type theory), Markov's principle (cf. Russian constructivism), nor the fan theorem (cf. Brouwer's intuitionism). Finally, the mechanization is integrated into the Coq Library of Undecidability Proofs [17], and contributes a (first of its kind) mechanized variant of Hooper's immortality construction [20].

The described improvements allow for an alternative approach to show many-one equivalence of System F typability and System F type checking, compared to the argument established in the 1990s by Wells [39]. The original argument interreduces type checking and typability directly, which requires a technically sophisticated argument. Having a constructive many-one reduction from Turing machine halting to semi-unification at our disposal (together with recursive enumerability of System F typability and type checking), it suffices (and is simpler) to reduce semi-unification to type checking and typability individually [11].

Synopsis

The reduction from Turing machine halting to semi-unification is divided into several reduction steps. Each reduction step is many-one, constructive, and mechanized as part of the Coq Library of Undecidability Proofs [17]. In particular, a predicate P over the domain X *constructively many-one* reduces to a predicate Q over the domain Y, if there exists a computable function $f : X \to Y$ such that for all $x \in X$ we constructively have $P(x) \iff Q(f(x))$.

Section 2: Turing machine halting is reduced to two-counter machine halting (Lemma 3) using Minsky's argument [33, Section 14.1]. This step simplifies the machine model.

Section 3: Two-counter machine halting is reduced to one-counter machine 1-halting (Lemma 12), adapting Minsky's observation [33, Section 14.2]. This step prepares the machine model for nested simulation via two-stack machines without symbol search.

Section 4.1: One-counter machine 1-halting is reduced to deterministic, length-preserving two-stack machine uniform boundedness (Lemma 26), adapting Hooper's construction for Turing machine immortality [20]. This step transitions the machine problem from halting to uniform boundedness.

Section 4.2: Deterministic, length-preserving two-stack machine uniform boundedness is reduced to confluent, simple two-stack machine uniform boundedness (Lemma 35), simplifying machine structure. This step enables reuse of the existing mechanized reduction from a uniform boundedness problem to semi-unification [10].

Section 5.1: Confluent, simple two-stack machine uniform boundedness is reduced to simple semi-unification (Lemma 46), strengthening previous work [10]. This step transitions to an undecidable fragment of semi-unification.

Section 5.2: Simple semi-unification is reduced to right-uniform, two-inequality semi-unification (Lemma 50), establishing the main result (Theorem 53).

Section 6: Outline of the mechanization of the above reduction steps.

2 Two-counter Machines

The key insight of recent work [10] establishes a direct correspondence between semi-unification and a uniform boundedness problem for a machine model. In order to reduce Turing machine halting to such a problem, in this section we consider *two-counter machines* as a well-understood, mechanized [16], and more convenient intermediate model of computation.

Two-counter machines, pioneered by Minsky [33, Section 14.1], are a restricted form of register machines and constitute a particularly simple, Turing-complete model of computation. A two-counter machine (Definition 1) stores data in two *counters*, each containing a natural number. A program instruction may either increment or decrement a counter value, and modify the current program index. To avoid partiality, we model halting via a trivial cycle. The size of a two-counter machine is the length, denoted $|\cdot|$, of the list of its instructions.

▶ **Definition 1** (Two-counter Machine ($\mathcal{M}$)). *A* two-counter machine $\mathcal{M}$ *is a list of* instructions *of shape either* $\mathtt{inc}_0$, $\mathtt{inc}_1$, $\mathtt{dec}_0\, j$, *or* $\mathtt{dec}_1\, j$, *where* $j \in \mathbb{N}$ *is a* program index.

A configuration *of* $\mathcal{M}$ *is of shape* $(i, (a, b))$, *where* $i \in \mathbb{N}$ *is the current* program index *and* $a, b \in \mathbb{N}$ *are the current* counter values.

The step relation *of* $\mathcal{M}$ *on configurations, written* $(\longrightarrow_{\mathcal{M}})$, *is given by*

- *if* $|\mathcal{M}| \leq i$, *then* $(i, (a, b)) \longrightarrow_{\mathcal{M}} (i, (a, b))$ *and we say* $(i, (a, b))$ halts
- *if* $\mathtt{inc}_0$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (a, b)) \longrightarrow_{\mathcal{M}} (i + 1, (a + 1, b))$
- *if* $\mathtt{inc}_1$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (a, b)) \longrightarrow_{\mathcal{M}} (i + 1, (a, b + 1))$
- *if* $\mathtt{dec}_0\, j$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (0, b)) \longrightarrow_{\mathcal{M}} (i + 1, (0, b))$ *and* $(i, (a + 1, b)) \longrightarrow_{\mathcal{M}} (j, (a, b))$
- *if* $\mathtt{dec}_1\, j$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (a, 0)) \longrightarrow_{\mathcal{M}} (i + 1, (a, 0))$ *and* $(i, (a, b + 1)) \longrightarrow_{\mathcal{M}} (j, (a, b))$

The reachability relation *of* $\mathcal{M}$ *on configurations, written* $(\longrightarrow^*_{\mathcal{M}})$, *is the reflexive, transitive closure of* $(\longrightarrow_{\mathcal{M}})$.

A configuration $(i, (a, b))$ *is* terminating *in* $\mathcal{M}$, *if we have* $(i, (a, b)) \longrightarrow^*_{\mathcal{P}} (i', (a', b'))$ *for some halting configuration* $(i', (a', b'))$.

Despite its remarkable simplicity, the halting problem for two-counter machines (Problem 2) is undecidable (Corollary 4).

▶ **Problem 2** (Two-counter Machine Halting)**.** Given a two-counter machine $\mathcal{M}$ and two natural numbers $a, b \in \mathbb{N}$, is the configuration $(0, (a, b))$ terminating in $\mathcal{M}$?

▶ **Lemma 3.** *The Turing machine halting problem many-one reduces to the two-counter machine halting problem (Problem 2).*

Proof Sketch. Minsky describes the simulation of Turing machines by machines with four registers [33, Section 11.2] working on Gödel encodings. Then, machines with four registers are simulated by two-counter machines [33, Theorem 14.1-1]. ◀

▶ **Corollary 4.** *Two-counter machine halting (Problem 2) is undecidable.*

▶ Remark 5. Minsky's original argument is constructive and is sufficient for our technical result. However, for mechanization we rely on existing work by Forster et al. [16], which is part of the Coq Library of Undecidability Proofs [17]. This approach many-one reduces Turing machine halting via the Post correspondence problem [35, 13] and Conway's FRACTRAN halting [7, 29] to two-counter machine halting.

Commonly, two-counter machines are easily simulated by other machine models and therefore serve a key role in undecidability proofs for machine immortality problems [20, 24]. However, they have one drawback with respect to uniform boundedness. That is, there is no natural increasing measure on configurations along the step relation, as it allows for non-trivial cycles (Example 6).

▶ **Example 6.** Consider $\mathcal{M} = [\mathtt{inc}_0, \mathtt{dec}_0\, 0]$. For any counter values $a, b \in \mathbb{N}$ the configuration $(0, (a, b))$ is non-terminating in $\mathcal{M}$ because of the non-trivial cycle $(0, (a, b)) \longrightarrow_{\mathcal{M}} (1, (a + 1, b)) \longrightarrow_{\mathcal{M}} (0, (a, b))$.

One could eliminate non-trivial cycles by introducing a third counter, which increases at every step. While this induces a natural increasing measure (value of the third counter), it incurs additional bookkeeping. A simulation of such a three-counter machine by an acyclic two-counter machine is possible [24], however, it again obscures the underlying measure.

We address this drawback of two-counter machines with respect to uniform boundedness in the following Section 3, building upon Minsky's notion of machines with *one* register.

3 One-counter Machines

As Minsky observed [33, Section 14.2], with multiplication and division by constants the halting problem is undecidable for *one-counter machines*. Specifically, increase (resp. decrease) operations for two values a and b can be simulated by multiplication (resp. division) by 2 and 3 for one value $2^a 3^b$.

In this section, we further develop Minsky's construction (similarly to [40]) of universal machines with one counter in pursuit of two goals. First, machine runs should be easy to simulate in the stack machine model (Remark 16). Second, we need a measure on machine configurations that increases along the step relation, directly connecting non-termination and unboundedness (Lemma 10).

A program instruction of a one-counter machine (Definition 7), besides modifying the program index, conditionally multiplies the current counter value with either $\frac{2}{1}$, $\frac{3}{2}$, $\frac{4}{3}$, or $\frac{5}{4}$. Notably, such a multiplication by $\frac{d+1}{d}$ for $d \in \{1, 2, 3, 4\}$ is both easy to simulate uniformly, and strictly increases a (positive) counter value.

▶ **Definition 7** (One-counter Machine ($\mathcal{P}$)). *A* one-counter machine $\mathcal{P}$ *is a list of* instructions *of shape* (j, d)*, where* $j \in \mathbb{N}$ *is a* program index *and* $d \in \{1, 2, 3, 4\}$ *is a* counter modifier.

A configuration *of* $\mathcal{P}$ *is a pair* (i, c)*, where* $i \in \mathbb{N}$ *is the current* program index *and* $c \in \mathbb{N}$ *such that* $c > 0$ *is the current* counter value.

The step relation *of* $\mathcal{P}$ *on configurations, written* $(\longrightarrow_{\mathcal{P}})$*, is given by*

- *if* $|\mathcal{P}| \leq i$*, then* $(i, c) \longrightarrow_{\mathcal{P}} (i, c)$ *and we say* (i, c) halts
- *if* (j, d) *is the* i*-th instruction of* $\mathcal{P}$ *and* d *divides* c*, then* $(i, c) \longrightarrow_{\mathcal{P}} (j, c \cdot \frac{d+1}{d})$
- *if* (j, d) *is the* i*-th instruction of* $\mathcal{P}$ *and* d *does not divide* c*, then* $(i, c) \longrightarrow_{\mathcal{P}} (i + 1, c)$

The reachability relation *of* $\mathcal{P}$ *on configurations, written* $(\longrightarrow_{\mathcal{P}}^{*})$*, is the reflexive, transitive closure of* $(\longrightarrow_{\mathcal{P}})$*.*

A configuration (i, c) *is* terminating *in* $\mathcal{P}$*, if we have* $(i, c) \longrightarrow_{\mathcal{P}}^{*} (i', c')$ *for some halting configuration* (i', c')*.*

▶ **Example 8.** Consider $\mathcal{P} = [(1, 1), (0, 2)]$. The configuration $(0, 1)$ is not terminating in $\mathcal{P}$ because of the infinite configuration chain

$$(0,1) \longrightarrow_{\mathcal{P}} (1, 1 \cdot \frac{2}{1}) \longrightarrow_{\mathcal{P}} (0, 2 \cdot \frac{3}{2}) \longrightarrow_{\mathcal{P}} (1, 3 \cdot \frac{2}{1}) \longrightarrow_{\mathcal{P}} (0, 6 \cdot \frac{3}{2}) \longrightarrow_{\mathcal{P}} (1, 9 \cdot \frac{2}{1}) \longrightarrow_{\mathcal{P}} \cdots$$

The step relation for one-counter machines is total (Lemma 9.1), functional (Lemma 9.2), and forms increasing chains (Lemma 9.3 and Lemma 9.4) up to a halting configuration.

▶ **Lemma 9** (One-counter Machine Step Relation Properties).

1. ***Totality:***
   For all configurations (i, c) *there is a configuration* (i', c') *such that* $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$*.*
2. ***Functionality:***
   If $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$ *and* $(i, c) \longrightarrow_{\mathcal{P}} (i'', c'')$*, then* $(i', c') = (i'', c'')$*.*
3. ***Increasing Measure:***
   If $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$ *and* (i, c) *is not halting, then* $|\mathcal{P}| \cdot c + i < |\mathcal{P}| \cdot c' + i'$*.*
4. ***Monotone Counter:***
   a. *If* $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$*, then* $c \leq c'$*.*
   b. *If* $(i, c) \longrightarrow_{\mathcal{P}}^{|\mathcal{P}|+1} (i', c')$ *and* (i', c') *is not halting, then* $c < c'$*.*

Proof. Routine case analysis. ◀

The above Lemma 9.3 gives an increasing along $(\longrightarrow_{\mathcal{P}})$ measure $|\mathcal{P}| \cdot c + i$ on non-halting configurations (i, c). Therefore, any configuration cycle is trivial, i.e. the corresponding configuration is halting. Additionally, by Lemma 9.4, the counter value is guaranteed to increase after $|\mathcal{P}| + 1$ steps, unless a halting configuration is reached. This results in a characterization of termination via boundedness of reachable counter values (Lemma 10).

▶ **Lemma 10.** *Let* $\mathcal{P}$ *be a one-counter machine. A configuration* (i, c) *is terminating in* $\mathcal{P}$ *iff there is a* $k \in \mathbb{N}$ *such that for all configurations* (i', c') *with* $(i, c) \longrightarrow_{\mathcal{P}}^{*} (i', c')$ *we have* $c' < k$*.*

Proof. If from (i, c) the machine $\mathcal{P}$ halts after n steps, then $k = 1 + c \cdot 2^n$ bounds the reachable from (i, c) counter values. Conversely, if k bounds the reachable from (i, c) counter values, then after at most $k \cdot (|\mathcal{P}| + 1)$ steps (i, c) reaches a halting configuration by Lemma 9.4. ◀

The halting problem for one-counter machines (Problem 11) starting from the configuration $(0, 1)$ is undecidable by reduction from the halting problem for two-counter machines (Lemma 12).

▶ **Problem 11** (One-counter Machine 1-Halting). Given a one-counter machine $\mathcal{P}$, is the configuration $(0, 1)$ terminating in $\mathcal{P}$?

▶ **Lemma 12.** *Two-counter machine halting (Problem 2) many-one reduces to one-counter machine* 1*-halting (Problem 11).*

Proof. Let $\mathcal{M}$ be a two-counter machine and let $a_0, b_0 \in \mathbb{N}$ be two values. We represent a pair (a, b) of counter values of $\mathcal{M}$ by the family of counter values $2^a 3^b 5^m$ where $m \in \mathbb{N}$.

We simulate instructions of $\mathcal{M}$ on two counters (a, b) by instructions of $\mathcal{P}$ on one counter $c = 2^a 3^b 5^m$ as follows. Increase a is simulated by $2^a 3^b 5^m \cdot \frac{2}{1} = 2^{a+1} 3^b 5^m$. Increase b is simulated by $2^a 3^b 5^m \cdot \frac{2}{1} \cdot \frac{3}{2} = 2^a 3^{b+1} 5^m$. Decrease a is simulated by $2^{a+1} 3^b 5^m \cdot \frac{3}{2} \cdot \frac{4}{3} \cdot \frac{5}{4} = 2^a 3^b 5^{m+1}$. Decrease b is simulated by $2^a 3^{b+1} 5^m \cdot \frac{4}{3} \cdot \frac{5}{4} = 2^a 3^b 5^{m+1}$. Failed decrease instructions may rely on a simulation of an unconditional jump instruction via $2^a 3^b 5^m \cdot \frac{2}{1} \cdot \frac{2}{1} \cdot \frac{5}{4} = 2^a 3^b 5^{m+1}$. Initialization of counter values (a_0, b_0) is simulated via $2^0 3^0 5^0 \cdot (\frac{2}{1})^{a_0+b_0} \cdot (\frac{3}{2})^{b_0} = 2^{a_0} 3^{b_0} 5^0$. Overall, the configuration $(0, (a_0, b_0))$ is terminating in the two-counter machine $\mathcal{M}$ iff the configuration $(0, 1)$ is terminating in the one-counter machine $\mathcal{P}$. ◀

▶ **Corollary 13.** *One-counter machine* 1*-halting (Problem 11) is undecidable.*

The following Example 14 illustrates the construction in the proof of Lemma 12, simulating a looping two-counter machine from Example 6.

▶ **Example 14.** Consider $\mathcal{M} = [\mathtt{inc}_0, \mathtt{dec}_0\, 0]$ from Example 6 with the initial counter values $(a_0, b_0) = (1, 1)$. Following the proof of Lemma 12, we construct the one-counter machine

$$\mathcal{P} = [(1,1), (2,1), (3,2), (4,1), (5,2), (6,3), (3,4)]$$

Starting from the configuration $(0, 1) = (0, 2^0 3^0 5^0)$ a run of $\mathcal{P}$ starts with the initialization

$$(0, 2^0 3^0 5^0) \xrightarrow{(1,1)}_{\mathcal{P}} (1, 2^1 3^0 5^0) \xrightarrow{(2,1)}_{\mathcal{P}} (2, 2^2 3^0 5^0) \xrightarrow{(3,2)}_{\mathcal{P}} (3, 2^1 3^1 5^0) = (3, 2^{a_0} 3^{b_0} 5^0)$$

Next, the infinite loop of $\mathcal{M}$ is simulated, returning to the program index 3

$$(3, 2^1 3^1 5^0) \xrightarrow{(4,1)}_{\mathcal{P}} (4, 2^2 3^1 5^0) \xrightarrow{(5,2)}_{\mathcal{P}} (5, 2^1 3^2 5^0) \xrightarrow{(6,3)}_{\mathcal{P}} (6, 2^3 3^1 5^0) \xrightarrow{(3,4)}_{\mathcal{P}} (3, 2^1 3^1 5^1)$$

Overall, the configuration $(0, 1) = (0, 2^0 3^0 5^0)$ is non-terminating in $\mathcal{P}$, simulating non-termination of the configuration $(0, (a_0, b_0))$ in $\mathcal{M}$ as follows

$$(0, 2^0 3^0 5^0) \longrightarrow^3_{\mathcal{P}} (3, 2^1 3^1 5^0) \longrightarrow^4_{\mathcal{P}} (3, 2^1 3^1 5^1) \longrightarrow^4_{\mathcal{P}} (3, 2^1 3^1 5^2) \longrightarrow^4_{\mathcal{P}} (3, 2^1 3^1 5^3) \longrightarrow^4_{\mathcal{P}} \dots$$

Indeed, there is no upper bound on the counter value (cf. Lemma 10).

▶ Remark 15. One-counter machines can be understood as a variant of Conway's FRACTRAN language [7] with a relaxed program index transition rule, and restricted to the instruction set $(\frac{2}{1}, \frac{3}{2}, \frac{4}{3}, \frac{5}{4})$.

▶ Remark 16. The deliberate choice of counter multiplication by $\frac{d+1}{d}$ for $d \in \{1, 2, 3, 4\}$ has several benefits. First, instructions are of uniform shape. Therefore, simulation of and reasoning about such instructions requires less case analysis, also impacting the underlying mechanization. Second, counter modification is non-decreasing by definition. Third, for a counter value $c = k \cdot d$ multiplication by $\frac{d+1}{d}$ results in $c \cdot \frac{d+1}{d} = c + k$. Therefore, it can be simulated by rewriting a binary word $0^c 1 0^k$ to $0^{c+k} 1$. This can be performed iteratively (and uniformly) by shifting the symbol 1 to the right for every consecutive occurrence of 0^d in 0^c (cf. proof of Lemma 26).

4 Two-stack Machines

In this section, our goal is the simulation of one-counter machines in a *stack machine* model of computation without unbounded *symbol search*. Specifically, given a one-counter machine $\mathcal{P}$, we construct a two-stack machine $\mathcal{S}$ such that the configuration $(0, 1)$ is terminating in $\mathcal{P}$ iff there is a uniform bound on the number of configurations reachable in $\mathcal{S}$ from any configuration.

The main difficulty, similarly to the undecidability proof for Turing machine immortality [20], is to simulate symbol search (traverse data, searching for a particular symbol) using a uniformly bounded machine. Most problematic are unsuccessful searches that may traverse an arbitrary, i.e. not uniformly bounded, amount of data. The key idea [20, Part IV] (see also [24, 22]) is to implement unbounded symbol search by nested bounded symbol search.

In the present work, we supplement a high level explanation of the construction (cf. proof sketch of Lemma 26) with a comprehensive case analysis as a mechanized proof (in the Coq proof assistant). This approach, arguably worth striving for in general, has three advantages over existing work. First, the proof idea is not cluttered with mundane technical details, while the mechanized proof is highly precise. Second, a mechanized proof leaves little doubt regarding proof correctness and is open to scrutiny, as there is nothing left to imagination. Third, the Coq proof assistant tracks any non-constructive assumptions which may hide beneath technical details.

Let us specify the two-stack machine (Definition 17) model of computation, which we use to simulate one-counter machines. An instruction of such a machine may modify the current machine state, pop from, and push onto two stacks of binary symbols.

▶ **Definition 17** (Two-stack Machine ($\mathcal{S}$)). *A* two-stack machine $\mathcal{S}$ *is a list of* instructions *of shape* $A \mid p \mid B \to A' \mid q \mid B'$*, where* $A, B, A', B' \in \{0,1\}^*$ *are binary* words *and* $p, q \in \mathbb{S}$ *are* states*, where* $\mathbb{S}$ *is countably infinite.*

A configuration *of* $\mathcal{S}$ *is of shape* $A \mid p \mid B$ *where* $p \in \mathbb{S}$ *is the* current state*,* $A \in \{0,1\}^*$ *is the content of the* left stack *and* $B \in \{0,1\}^*$ *is the content of the* right stack*.*

The step relation *of* $\mathcal{S}$ *on configurations, written* $(\longrightarrow_{\mathcal{S}})$*, is given by*

- *if* $(A \mid p \mid B \to A' \mid q \mid B') \in \mathcal{S}$*, then for* $C, D \in \{0,1\}^*$ *we have* $CA \mid p \mid BD \longrightarrow_{\mathcal{S}} CA' \mid q \mid B'D$

The reachability relation *of* $\mathcal{S}$ *on configurations, written* $(\longrightarrow_{\mathcal{S}}^*)$*, is the reflexive, transitive closure of* $(\longrightarrow_{\mathcal{S}})$*.*

▶ Remark 18. A two-stack machine can be understood as a restricted semi-Thue system on the alphabet $\{0,1\} \cup \mathbb{S}$ in which each word contains exactly one symbol from $\mathbb{S}$. Such rewriting systems are employed in the setting of synchronous distributivity [1].

▶ Remark 19. To accommodate for arbitrary large machines, the state space $\mathbb{S}$ is not finite. However, the *effective* state space of any two-stack machine $\mathcal{S}$ is bounded by the finitely many states occurring in the instructions of $\mathcal{S}$.

The key undecidable property of two-stack machines, used in [10], is whether the number of distinct, reachable configurations from any configuration is *uniformly bounded* (Definition 20).

▶ **Definition 20** (Uniformly Bounded). *A two-stack machine* $\mathcal{S}$ *is* uniformly bounded *if there exists an* $n \in \mathbb{N}$ *such that for any configuration* $A \mid p \mid B$ *we have*

$$|\{A' \mid p' \mid B' \mid A \mid p \mid B \longrightarrow_{\mathcal{S}}^* A' \mid p' \mid B'\}| \leq n$$

Notably, uniform boundedness and *uniform termination* [32] (is every configuration chain finite?) are orthogonal notions, illustrated by the following Examples 21–22.

▶ **Example 21.** Consider $\mathcal{S} = [0|p|\epsilon \to \epsilon|p|1]$. From the configuration $0^m|p|\epsilon$, where $m \in \mathbb{N}$, reachable in $\mathcal{S}$ configurations are exactly $0^{m-i}|p|1^i$ for $i = 0 \dots m$. Therefore, there is no *uniform* bound on the number of reachable configurations. However, the length of every configuration chain in $\mathcal{S}$ is finite (bounded by one plus the length of the left stack). Overall, $\mathcal{S}$ is uniformly terminating but not uniformly bounded.

▶ **Example 22.** Consider $\mathcal{S} = [(0|p|\epsilon \to \epsilon|q|1), (\epsilon|q|1 \to 0|p|\epsilon)]$. The number of distinct, reachable in $\mathcal{S}$ configurations from any configuration is uniformly bounded by $n = 2$. However, there is an infinite configuration chain $0|p|\epsilon \longrightarrow_{\mathcal{S}} \epsilon|q|1 \longrightarrow_{\mathcal{S}} 0|p|\epsilon \longrightarrow_{\mathcal{S}} \epsilon|q|1 \longrightarrow_{\mathcal{S}} \dots$ Overall, $\mathcal{S}$ is uniformly bounded, but not uniformly terminating.

In literature [20, 24], counter machine termination is simulated using uniformly bounded Turing machines directly rather than by two-stack machines. This is reasonable when omitting technical details regarding the exact Turing machine construction. However, for verification in full detail, Turing machines are quite unwieldy, compared to two-stack machines. Unfortunately, we cannot rely on existing mechanized Turing machine programming techniques [15], as they establish functional properties, but are incapable to establish uniform boundedness.

4.1 Deterministic, Length-preserving Two-stack Machines

There are several properties of two-stack machines that are of importance in our construction in order to reuse existing work [10].

For *length-preserving* two-stack machines (Definition 23) the sum of lengths of the two stacks is invariant wrt. reachability. For each configuration, length-preservation bounds (albeit, not *uniformly*) the number of distinct, reachable configurations. Therefore, reachability is decidable for length-preserving two-stack machines.

▶ **Definition 23** (Length-preserving). *A two-stack machine $\mathcal{S}$ is* length-preserving *if for all instructions $(A|p|B \to A'|q|B') \in \mathcal{S}$ we have $0 < |A| + |B| = |A'| + |B'|$.*

▶ **Definition 24** (Deterministic). *A two-stack machine $\mathcal{S}$ is* deterministic *if for all configurations $A|p|B$, $A'|p'|B'$, and $A''|p''|B''$ such that $A|p|B \longrightarrow_{\mathcal{S}} A'|p'|B'$ and $A|p|B \longrightarrow_{\mathcal{S}} A''|p''|B''$ we have $A'|p'|B' = A''|p''|B''$.*

For example, two-stack machines in Examples 21–22 are deterministic and length-preserving.

The key undecidable problem, that in our argument assumes the role of Turing machine immortality of previous approaches [27, 10], is uniform boundedness of deterministic, length-preserving two-stack machines (Problem 25). A central insight of the present work is that using this problem as an intermediate step we neither require Turing reductions, König's lemma (cf. [27]), nor the fan theorem (cf. [10]). Additionally, this problem strikes a balance between ease to reduce to (from counter machine halting) and ease to reduce from (to semi-unification). This balance is essential for a mechanization of manageable size.

▶ **Problem 25** (Deterministic, Length-preserving Two-stack Machine Uniform Boundedness). Given a deterministic, length-preserving two-stack machine $\mathcal{S}$, is $\mathcal{S}$ uniformly bounded?

The original undecidability proof of semi-unification contains a hint [27, Proof of Corollary 6] that Turing machine immortality may be avoided in a comprehensive reduction. Accordingly, the following Lemma 26 captures the decisive step that avoids Turing machine immortality.

▶ **Lemma 26.** *One-counter machine* 1*-halting (Problem 11) many-one reduces to deterministic, length-preserving two-stack machine uniform boundedness (Problem 25).*

Proof Sketch. Let $\mathcal{P}$ be a one-counter machine. Similarly to [24, Theorem 7], we adapt Hooper's argument [20] for symbol search.

A naive simulation of $\mathcal{P}$ by a deterministic, length-preserving two-stack machine $\mathcal{S}$ is easy. A $\mathcal{P}$-configuration (i, c) is represented by the $\mathcal{S}$-configuration $1_{\mid}i_{\mid}0^c10^m$ for $m \in \mathbb{N}$, where 0^m is a large enough supply of zeroes. A $\mathcal{P}$-instruction (j, d) is simulated as follows. First, the $\mathcal{S}$-instruction $\epsilon_{\mid}i_{?\mid}0^d \longrightarrow_{\mathcal{S}} 0^d{}_{\mid}i_{?\mid}\epsilon$ tests for divisibility by d, moving consecutive blocks of d zeroes from the right to the left stack. The divisibility test fails if the $\mathcal{S}$-instruction $\epsilon_{\mid}i_{?\mid}0^k1 \longrightarrow_{\mathcal{S}} \epsilon_{\mid}i_{\#\mid}0^k1$ where $0 < k < d$ can be applied. In this case, we move the zeroes back to the right stack, and via the $\mathcal{S}$-instruction $1_{\mid}i_{\#\mid}\epsilon \longrightarrow_{\mathcal{S}} 1_{\mid}i+1_{\mid}\epsilon$ we reach the $\mathcal{S}$-configuration $1_{\mid}i+1_{\mid}0^c10^m$, representing the $\mathcal{P}$-configuration $(i+1, c)$. The divisibility test succeeds if the $\mathcal{S}$-instruction $\epsilon_{\mid}i_{?\mid}1 \longrightarrow_{\mathcal{S}} \epsilon_{\mid}i_{!\mid}1$ can be applied. In this case, multiplication by $\frac{d+1}{d}$ is simulated by shifting to the right the symbol 1 on the right stack for each block of consecutive d zeroes on the left stack (Remark 16). Finally, via the $\mathcal{S}$-instruction $1_{\mid}i_{!\mid}\epsilon \longrightarrow_{\mathcal{S}} 1_{\mid}j_{\mid}\epsilon$, we arrive at the $\mathcal{S}$-configuration $1_{\mid}j_{\mid}0^{\frac{c(d+1)}{d}}10^{m-\frac{c}{d}}$, representing the $\mathcal{P}$-configuration $(j, c \cdot \frac{d+1}{d})$.

Inductively, we obtain $(i, c) \longrightarrow^*_{\mathcal{P}} (i', c')$ iff $A1_{\mid}i_{\mid}0^c10^{c'-c}B \longrightarrow^*_{\mathcal{S}} A1_{\mid}i'_{\mid}0^{c'}1B$ for all $A, B \in \{0,1\}^*$. Therefore, the configuration $(0, 1)$ is terminating in $\mathcal{P}$ iff configurations $A1_{\mid}0_{\mid}01B$ are uniformly bounded in $\mathcal{S}$ (the uniform bound is derived from the halting counter value).

Unfortunately, the naive construction fails in general. For example, termination of $(0, 1)$ in $\mathcal{P}$ does not necessarily uniformly bound the configurations $1_{\mid}0_{\mid}0^m$ where $m \in \mathbb{N}$ is arbitrary large. At fault is a failed symbol search (for the symbol 1 on the right stack) that needs to traverse an arbitrary amount of data. Observe that in the naive construction any search for the symbol 1 is expected to succeed. The ingenious idea by Hooper [20, Part IV] is to uniformly bound symbol search via nested simulation in a sufficiently large uniformly bounded configuration space (in our case, the space of configurations $A1_{\mid}0_{\mid}01B$ for $A, B \in \{0,1\}^*$). That is, to search for the symbol 1 on the right stack, start a nested simulation from the $\mathcal{P}$-configuration $(0, 1)$ inside the space of consecutive zeros on the right stack. Specifically, use $A_{\mid}p_{\mid}0^{k+3}B \longrightarrow_{\mathcal{S}} AC1_{\mid}0_{\mid}01B$ to reset the program index p to 0 and retain the binary encoding of p in C of fixed length k. Let c be the counter value (arbitrarily large by Lemma 9.4) of the halting configuration in $\mathcal{P}$ from the $\mathcal{P}$-configuration $(0, 1)$. For the configuration $AC1_{\mid}0_{\mid}01B$ which represents the $\mathcal{P}$-configuration $(0, 1)$, there are three cases.

First, in case $B = 0^m1D$, where $m < c-1$ and $D \in \{0,1\}^*$, the number m of zeroes on the right stack is too small to accommodate for c. Eventually, the nested simulation is unable to simulate counter increase. In this situation, the initial search for the symbol 1 succeeds, and control is returned to the previous level.

Second, in case $B = 0^m$, where $m < c-1$, the size of the right stack is too small to accommodate for c. Eventually, the nested simulation is unable to apply any instruction, and halts. In this situation, the initial search for the symbol 1 fails, respecting the uniform bound derived from c.

Third, in case $B = 0^{c-1}D$, where $D \in \{0,1\}^*$, there is enough space for the nested simulation to reach a halting state in $\mathcal{P}$ from the initial configuration $(0, 1)$, respecting the uniform bound derived from c. This renders the initial search for the symbol 1 immaterial, because the simulation already achieved its ultimate purpose.

In each case (using Lemma 9), a uniform bound on the number of configurations for the nested simulation can be derived from c. Each case may require further nested computation. However, the nesting depth is at most c because each nesting level uses space inside the consecutive zeroes that represent the counter value (bounded by c) on the previous level. ◀

▶ **Corollary 27.** *Deterministic, length-preserving two-stack machine uniform boundedness (Problem 25) is undecidable.*

▶ Remark 28. The exact analysis of the nested simulation in the proof of Lemma 26 requires a tremendous inductive proof with many corner cases. Arguably, it is unreasonable for a human without mechanical assistance to write it down in full detail (cf. Hooper's remark in Section 1). Additionally, it would require a comparable amount of effort for others to verify such a massive construction. This is why, in order to guarantee its correctness, a mechanized proof of Lemma 26 is adequate (cf. Section 6) to establish the result.

4.2 Confluent, Simple Two-stack Machines

In order to build upon existing work [10], we consider two-stack machines with instructions of *simple* (Definition 29, cf. [10, Definition 16]) shape. To further streamline the construction, we relax determinism of two-stack machines to confluence (Definition 32). Overall, confluent, simple two-stack machine uniform boundedness (Problem 34) is well-suited for reduction to a fragment of semi-unification (cf. Section 5.1).

▶ **Definition 29** (Simple). *A two-stack machine $\mathcal{S}$ is* simple *if for all instructions $(A\vert p\vert B \to A'\vert q\vert B') \in \mathcal{S}$ we have $1 = |A| + |B| = |A'| + |B'| = |A| + |A'| = |B| + |B'|$.*

▶ Remark 30. A deterministic, simple two-stack machine is just another way to present a deterministic Turing machine. The left and right stacks contain the respective tape content to the left and to the right from the current head. Reading and writing at the head while moving the head position is easily presented as simple instructions (cf. [10, Remark 19]).

▶ Remark 31. Turing machine immortality is reducible to uniform boundedness of deterministic, simple two-stack machines by a bounded Turing reduction [10, Theorem 2]. However, the argument uses the fan theorem, therefore, it is crucial for the present argument not to rely on this particular reduction.

▶ **Definition 32** (Confluent). *A two-stack machine $\mathcal{S}$ is* confluent *if for all configurations $A\vert p\vert B$, $A'\vert p'\vert B'$, and $A''\vert p''\vert B''$ such that $A\vert p\vert B \longrightarrow_{\mathcal{S}}^{*} A'\vert p'\vert B'$ and $A\vert p\vert B \longrightarrow_{\mathcal{S}}^{*} A''\vert p''\vert B''$ there exists a configuration $C\vert q\vert D$ such that $A'\vert p'\vert B' \longrightarrow_{\mathcal{S}}^{*} C\vert q\vert D$ and $A''\vert p''\vert B'' \longrightarrow_{\mathcal{S}}^{*} C\vert q\vert D$.*

Clearly, any deterministic two-stack machine is confluent (but not necessarily vice versa).

▶ **Lemma 33.** *If a two-stack machine $\mathcal{S}$ is deterministic, then $\mathcal{S}$ is confluent.*

Compared to deterministic machines, confluent machines are quite practical. For example, a confluent machine may (without additional bookkeeping) "try out" different configuration chains before choosing the preferable one (cf. proof sketch of Lemma 35).

▶ **Problem 34** (Confluent, Simple Two-stack Machine Uniform Boundedness). Given a confluent, simple two-stack machine $\mathcal{S}$, is $\mathcal{S}$ uniformly bounded?

By Lemma 33, the above Problem 34 subsumes deterministic, simple two-stack machine uniform boundedness [10, Problem 26]. This, in combination with the following Lemma 35, allows for adaptation of previous work[1] in Section 5.1.

[1] It is possible to carry out the construction in the original, deterministic scenario without adaptation. However, this is technically more challenging and provides no tangible benefit.

▶ **Lemma 35.** *Deterministic, length-preserving two-stack machine uniform boundedness (Problem 25) many-one reduces to confluent, simple two-stack machine uniform boundedness (Problem 34).*

Proof Sketch. Our objective is to shorten length-preserving instructions, while maintaining confluence. This is routine, storing local stack information in additional fresh states. For example, an instruction $00\mid p\mid\epsilon \longrightarrow 11\mid q\mid\epsilon$ can be replaced by the simple instructions $0\mid p\mid\epsilon \longrightarrow \epsilon\mid p_1\mid 0$, $0\mid p_1\mid\epsilon \longrightarrow \epsilon\mid p_2\mid 0$, $\epsilon\mid p_2\mid 0 \longrightarrow 1\mid p_3\mid\epsilon$, and $\epsilon\mid p_3\mid 0 \longrightarrow 1\mid q\mid\epsilon$, where p_1, p_2, p_3 are fresh states. This results in the configuration chain $00\mid p\mid\epsilon \longrightarrow 0\mid p_1\mid 0 \longrightarrow \epsilon\mid p_2\mid 00 \longrightarrow 1\mid p_3\mid 0 \longrightarrow 11\mid q\mid\epsilon$ which simulates the instruction $00\mid p\mid\epsilon \longrightarrow 11\mid q\mid\epsilon$.

Notably, in the exemplified transformation it is difficult to maintain determinism. However, in order to maintain confluence it suffices to add reverse instructions from fresh states, i.e. $\epsilon\mid p_1\mid 0 \longrightarrow 0\mid p\mid\epsilon$, $\epsilon\mid p_2\mid 0 \longrightarrow 0\mid p_1\mid\epsilon$, and $1\mid p_3\mid\epsilon \longrightarrow \epsilon\mid p_2\mid 0$. Therefore, any failed attempt to read local stack information is reversible and computation is confluent. ◀

5 Semi-unification

Semi-unification (Problem 38) can be understood as combination of first-order unification (cf. valuation φ) and first-order matching (cf. valuations ψ). For the undecidability of semi-unification [26, Theorem 12], it suffices to restrict the syntax of the underlying terms (Definition 36) to variables together with a binary constructor ($\to$).

In this section, we recapitulate necessary definitions and properties of semi-unification from existing work [27, 10], in order to complete a constructive many-one reduction from Turing machine halting to semi-unification (Theorem 53).

▶ **Definition 36** (Terms ($\mathbb{T}$))**.** *Let α, β, γ range over a countably infinite set $\mathbb{V}$ of* variables*. The set of* terms $\mathbb{T}$*, ranged over by σ, τ, is given by the grammar $\sigma, \tau \in \mathbb{T} ::= \alpha \mid \sigma \to \tau$.*

▶ **Definition 37** (Valuation (φ), (ψ))**.** *A* valuation $\varphi : \mathbb{V} \to \mathbb{T}$ *assigns terms to variables, and is tacitly lifted to terms.*

▶ **Problem 38** (Semi-unification [27, SUP], [10, Problem 3])**.**
Given a set $\mathcal{I} = \{\sigma_1 \leq \tau_1, \ldots, \sigma_n \leq \tau_n\}$ of *inequalities*, is there a valuation φ such that for each inequality $(\sigma \leq \tau) \in \mathcal{I}$ there exists a valuation $\psi : \mathbb{V} \to \mathbb{T}$ such that $\psi(\varphi(\sigma)) = \varphi(\tau)$?

▶ Remark 39. As given by Definition 37, the set of valuations is not countable. However, in any semi-unification instance $\mathcal{I}$ the number of inequalities (consisting of first-order terms) is finite. Therefore, restricting valuations to be finite maps (from the relevant variables) does not change the expressive power of semi-unification. As a result, semi-unification is recursively enumerable.

The following Examples 40 (resp. Example 41) illustrates a positive (resp. negative) instance of semi-unification.

▶ **Example 40.** Consider $\mathcal{I} = \{\alpha \leq \alpha \to \alpha,\ \alpha \leq \alpha \to \alpha \to \alpha\}$. The semi-unification instance $\mathcal{I}$ is solved by the valuation φ such that $\varphi(\alpha) = \alpha$.
For the inequality $\alpha \leq \alpha \to \alpha$ there exists a valuation ψ such that $\psi(\alpha) = \alpha \to \alpha$, and therefore $\psi(\varphi(\alpha)) = \varphi(\alpha \to \alpha)$. For the inequality $\alpha \leq \alpha \to \alpha \to \alpha$ there exists a valuation ψ such that $\psi(\alpha) = \alpha \to \alpha \to \alpha$, and therefore $\psi(\varphi(\alpha)) = \varphi(\alpha \to \alpha \to \alpha)$.

▶ **Example 41.** Consider $\mathcal{I} = \{\alpha \to \alpha \leq \alpha\}$. The semi-unification instance $\mathcal{I}$ is not solvable. Assume that there exist valuations φ and ψ such that $\psi(\varphi(\alpha \to \alpha)) = \varphi(\alpha)$. Therefore, the size of the syntax tree of $\varphi(\alpha)$ is twice the size of the syntax tree $\psi(\varphi(\alpha))$ which is not possible for (non-empty, finite) terms.

Unfortunately, semi-unification does not admit a decision procedure based on an occurs-check, which is a common approach to both first-order unification and first-order matching [3] (see also the redex contraction procedure [27, Section 2]). However, it is challenging to construct an unsolvable example of manageable size, for which the occurs-check fails.

Originally [27, Theorem 12], semi-unification is proven undecidable by Turing reduction from Turing machine immortality [20]. As intermediate problems, the argument relies on symmetric intercell Turing machine boundedness, path equation derivability, and termination of a redex contraction procedure that is custom-tailored for semi-unification. Additionally, the argument uses König's lemma and it is not obvious whether it can be presented constructively.

A modern approach [10, Theorem 4] simplifies the traditional argument. It still relies on a Turing reduction from Turing machine immortality, but uses only deterministic, simple two-stack machine uniform boundedness to show undecidability of a fragment of semi-unification. Additionally, it relies on the fan theorem, which is strictly weaker than König's lemma and is valid in Brouwer's intuitionism. The argument is partially mechanized (treating Turing machine immortality axiomatically) in Coq.

In the remainder of this section we briefly recapitulate and reuse the modern approach [10] in the more general case of confluent, simple two-stack machines. This allows us to avoid Turing machine immortality, Turing reductions, and the fan theorem in the overall argument.

5.1 Simple Semi-unification

In this section, we recapitulate the intermediate problem of *simple semi-unification* (Problem 44) [10, Problem 15], which connects stack machine computation and semi-unification. Intuitively, term variables represent machine states, simple constraints (Definition 42) represent local stack transformations, and the model relation (Definition 43) captures machine reachability via valuations.

▶ **Definition 42** (Simple Constraint [10, Definition 6]). *A* simple constraint *has the shape* $a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$*, where* $a, b \in \{0, 1\}$ *are symbols and* $\alpha, \beta \in \mathbb{V}$ *are variables.*

▶ **Definition 43** (Model Relation [10, Definition 9]). *A valuation triple* $(\varphi, \psi_0, \psi_1)$ models *a simple constraint* $a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$*, written* $(\varphi, \psi_0, \psi_1) \models a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$*, if one of the following conditions holds*

- $b = 0$ *and* $\psi_a(\varphi(\alpha)) \to \tau = \varphi(\beta)$ *for some term* $\tau \in \mathbb{T}$
- $b = 1$ *and* $\sigma \to \psi_a(\varphi(\alpha)) = \varphi(\beta)$ *for some term* $\sigma \in \mathbb{T}$

▶ **Problem 44** (Simple Semi-unification [10, Problem 15]). Given a finite set $\mathcal{C}$ of simple constraints, do there exist valuations $\varphi, \psi_0, \psi_1 : \mathbb{V} \to \mathbb{T}$ such that for each simple constraint $(a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b) \in \mathcal{C}$ we have $(\varphi, \psi_0, \psi_1) \models a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$?

The following Example 45 illustrates a model of a set of simple constraints, i.e. a solvable instance of simple semi-unification.

▶ **Example 45.** Consider $\mathcal{C} = \{0 \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid 1, 1 \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid 0\}$. A possible valuation triple $(\varphi, \psi_0, \psi_1)$ which models each simple constraint in $\mathcal{C}$ is such that $\varphi(\alpha) = \alpha$, $\varphi(\beta) = \beta_1 \to \beta_2$, $\psi_0(\alpha) = \beta_2$, $\psi_1(\alpha) = \beta_1$. Indeed, we have $\beta_1 \to \psi_0(\varphi(\alpha)) = \beta_1 \to \beta_2 = \varphi(\beta)$ and $\psi_1(\varphi(\alpha)) \to \beta_2 = \beta_1 \to \beta_2 = \varphi(\beta)$.

The pivotal step in previous work [10] is a many-one reduction from deterministic, simple two-stack machine uniform boundedness to simple semi-unification. This result readily generalizes to the confluent case (Lemma 46).

▶ **Lemma 46.** *Confluent, simple two-stack machine uniform boundedness (Problem 34) many-one reduces to simple semi-unification (Problem 44).*

Proof Sketch. We follow exactly the argument structure of [10, Section 4]. Tacitly inject machine states into variables, i.e. let $\mathbb{S} \subseteq \mathbb{V}$. Given a confluent, simple two-stack machine $\mathcal{S}$, construct the set of simple constraints

$$\mathcal{C} = \{a \imath p \imath \epsilon \doteq \epsilon \imath q \imath b \mid (a \imath p \imath \epsilon \to \epsilon \imath q \imath b) \in \mathcal{S} \text{ or } (\epsilon \imath q \imath b \to a \imath p \imath \epsilon) \in \mathcal{S}\}$$

If $\mathcal{S}$ is uniformly bounded, then the exact construction of $(\varphi, \psi_0, \psi_1)$ [10, Definition 42] yields a model of each simple constraint in $\mathcal{C}$.

Conversely, if $(\varphi, \psi_0, \psi_1)$ models each constraint in $\mathcal{C}$, then by the exact argument of [10, Lemma 48] the maximal depth of the syntax trees in the range of φ induces a uniform bound on the number of configurations reachable from any configuration in $\mathcal{S}$. ◀

▶ Remark 47. Although the original construction [10, Lemma 45 and Lemma 48] requires determinism, only confluence is used in the actual proofs [10, Lemma 30, Lemma 39]. While tedious to verify by hand, the available mechanization allows for simple replacement of determinism by confluence, while the proof assistant guarantees correctness of any related details. This highlights the effectiveness of proof assistants to accommodate for changes in a complex argument, reevaluating its overall correctness.

5.2 Right-uniform, Two-inequality Semi-unification

In this section, we consider a restriction of semi-unification to only two inequalities with identical right-hand sides (Problem 48). Such a restriction is a convenient byproduct of the reduction from simple semi-unification, and may simplify existing undecidability proofs that rely on the undecidability of semi-unification.

▶ **Problem 48** (Right-uniform, Two-inequality Semi-unification)**.** Given two inequalities $\sigma_0 \leq \tau$ and $\sigma_1 \leq \tau$ with identical right-hand sides, do there exist valuations φ, ψ_0, ψ_1 such that $\psi_0(\varphi(\sigma_0)) = \varphi(\tau)$ and $\psi_1(\varphi(\sigma_1)) = \varphi(\tau)$?

▶ Remark 49. Simply put, the above Problem 48 can be stated as follows: Given three terms σ_0, σ_1, τ, are there valuations φ, ψ_0, ψ_1 such that $\psi_0(\varphi(\sigma_0)) = \varphi(\tau) = \psi_1(\varphi(\sigma_1))$?

It is an easy exercise to reduce simple semi-unification to (non right-uniform) two-inequality semi-unification [10, Theorem 1]. We slightly adjust the existing construction to produce right-uniform inequalities.

▶ **Lemma 50.** *Simple semi-unification (Problem 44) many-one reduces to right-uniform, two-inequality semi-unification (Problem 48).*

Proof. Given constraints $\mathcal{C} = \{a_i \imath \alpha_i \imath \epsilon \doteq \epsilon \imath \beta_i \imath b_i \mid i = 1 \dots n\}$, define $\tau = \beta_1 \to \dots \to \beta_n$. Let γ_i be fresh variables for $i = 1 \dots n$ and define $\sigma^j = \sigma^j_1 \to \dots \to \sigma^j_n$ for $j \in \{0, 1\}$ where

$$\sigma^j_i = \alpha_i \to \gamma_i \text{ if } a_i = j \text{ and } b_i = 0 \qquad \sigma^j_i = \gamma_i \to \alpha_i \text{ if } a_i = j \text{ and } b_i = 1 \qquad \sigma^j_i = \gamma_i \text{ else}$$

We show that $\mathcal{C}$ is solvable iff the right-uniform inequalities $\sigma^0 \leq \tau$ and $\sigma^1 \leq \tau$ are solvable.

First, assume that the valuation triple $(\varphi, \psi_0, \psi_1)$ models each simple constraint in $\mathcal{C}$. Wlog. $\varphi(\gamma_i) = \psi_0(\gamma_i) = \psi_1(\gamma_i) = \gamma_i$ for $i = 1 \dots n$. By routine case analysis, we may adjust $\psi_0(\gamma_i)$ and $\psi_1(\gamma_i)$ for $i = 1 \dots n$ to obtain valuations ψ'_0 and ψ'_1 such that $\psi'_0(\varphi(\sigma^0)) = \varphi(\tau) = \psi'_1(\varphi(\sigma^1))$.

Second, any solution φ, ψ_0, ψ_1 of $\sigma^0 \leq \tau$ and $\sigma^1 \leq \tau$ also models each constraint in $\mathcal{C}$. ◀

▶ **Corollary 51.** *Simple semi-unification (Problem 44) many-one reduces to semi-unification (Problem 38).*

The following Example 52 illustrates the proof of Lemma 50 on the basis of Example 45.

▶ **Example 52.** Consider $\mathcal{C} = \{0\mid\alpha\mid\epsilon \doteq \epsilon\mid\beta\mid 1, 1\mid\alpha\mid\epsilon \doteq \epsilon\mid\beta\mid 0\}$ from Example 45. Define the terms $\sigma^0 = (\gamma_1 \to \alpha) \to \gamma_2$, $\sigma^1 = \gamma_1 \to (\alpha \to \gamma_2)$, and $\tau = \beta \to \beta$. The valuation triple $(\varphi, \psi_0, \psi_1)$ from Example 45 is extended to γ_1, γ_2 to obtain a solution $\varphi, \psi_0', \psi_1'$ of the right-uniform inequalities $\sigma^0 \leq \tau$ and $\sigma^1 \leq \tau$ as follows: $\psi_0'(\gamma_1) = \beta_1$, $\psi_0'(\gamma_2) = \beta_1 \to \beta_2$, $\psi_1'(\gamma_1) = \beta_1 \to \beta_2$, and $\psi_1'(\gamma_2) = \beta_2$. We have $(\beta_1 \to \beta_2) \to (\beta_1 \to \beta_2) = \varphi(\tau) = \psi_0'(\varphi(\sigma^0)) = (\psi_1'(\varphi(\sigma^1))$.

5.3 Main Result

Finally, we compose the previously described reductions into a comprehensive, constructive many-one reduction from Turing machine halting to semi-unification (Theorem 53). This constitutes the main result of the present work.

▶ **Theorem 53.** *Turing machine halting constructively many-one reduces to semi-unification (Problem 38).*

Proof. By composition of Lemmas 3, 12, 26, 35, 46, and Corollary 51. Constructivity of the argument is witnessed by an axiom-free mechanization (cf. Section 6) using the Coq proof assistant. ◀

Since semi-unification is recursively enumerable (Remark 39), it is many-one complete (in the sense of [38, Chapter 7.2]).

▶ **Corollary 54.** *Semi-unification (Problem 38) is many-one complete.*

6 Mechanization

This section provides an overview over the constructive mechanization, using the Coq proof assistant [8], of the many-one reduction from Turing machine halting to semi-unification. The mechanization relies on, and is integrated into the growing Coq Library of Undecidability Proofs [17]. The reduction is axiom-free and spans approximately 20000 lines of code, of which 3500 is contributed by the present work.

At the core of the library is the following mechanized notion of many-one reducibility[2]

```
Definition reduction {X Y} (f: X -> Y) (P: X -> Prop) (Q: Y -> Prop) :=
  forall x, P x <-> Q (f x).

Definition reduces {X Y} (P: X -> Prop) (Q: Y -> Prop) :=
  exists f: X -> Y, reduction f P Q.

Notation "P ⪯ Q" := (reduces P Q).
```

In the above, a predicate `P` over the domain `X` many-one reduces to a predicate `Q` over the domain `Y`, denoted `P ⪯ Q`, if there exists a function `f: X -> Y` such that for all `x` in the domain `X` we have `P x` iff `Q (f x)`. In axiom-free Coq any such function `f: X -> Y` is computable, a necessity oftentimes handled with less rigor in traditional (non-mechanized) proofs. Additionally, in axiom-free Coq, a proof of `P x <-> Q (f x)` cannot rely on principles such as functional extensionality, the fan theorem, or the law of excluded middle. Notably, our main Theorem 53 is mechanized in this setting.

[2] cf. `theories/Synthetic/Definitions.v`

The key contribution of the present work is consolidated as part of the following conjunction of many-one reductions[3]

```
Theorem HaltTM_1_chain_SemiU :
  HaltTM 1 ⪯ iPCPb /\
  iPCPb ⪯ BSM_HALTING /\
  BSM_HALTING ⪯ MM2_HALTING /\
  MM2_HALTING ⪯ CM1_HALT /\
  CM1_HALT ⪯ SMNdl_UB /\
  SMNdl_UB ⪯ CSSM_UB /\
  CSSM_UB ⪯ SSemiU /\
  SSemiU ⪯ RU2SemiU /\
  RU2SemiU ⪯ SemiU.
```

The individual problems in the above chain of many-one reductions are as follows.

- `HaltTM 1` is one-tape Turing machine halting, and native to the library as the initial undecidable problem, building upon prior work [15, 2] in computability theory
- `iPCPb` is indexed, binary Post correspondence problem, mechanized in [13]
- `BSM_HALTING` is binary stack machine halting, mechanized in [16]
- `MM2_HALTING` is two-counter machine halting (Problem 2), mechanized in [16]
- `CM1_HALT` is one-counter machine 1-halting (Problem 11)
- `SMNdl_UB` is uniform boundedness of deterministic, length-preserving two-stack machines (Problem 25)
- `CSSM_UB` is uniform boundedness of confluent, simple stack machines (Problem 34)
- `SSemiU` is simple semi-unification (Problem 44), mechanized in [10]
- `RU2SemiU` is right-uniform, two-inequality semi-unification (Problem 48)
- `SemiU` is semi-unification (Problem 38), mechanized in [10]

Correctness of the argument is witnessed by the verification of `Theorem HaltTM_1_chain_SemiU` in axiom-free Coq, for which constructivity is certified using the `Print Assumptions` command [9].

By transitivity of many-one reducibility we obtain `Theorem reduction : HaltTM 1 ⪯ SemiU`[4].

As a result, the statement `HaltTM 1 ⪯ SemiU` faithfully mechanizes our overall formal goal of a constructive many-one reduction from Turing machine halting to semi-unification. In fact, the particular many-one reduction function could be extracted from the proof of `HaltTM 1 ⪯ SemiU` as a λ-term (in the call-by-value λ-calculus model of computation) using existing techniques [14].

The mechanization of `CM1_HALT`, `SMNdl_UB`, and `CSSM_UB` together with corresponding many-one reductions are contributed to the library as part of the present work. The proof of `CSSM_UB ⪯ SSemiU` is an almost verbatim copy of the corresponding `DSSM_UB ⪯ SSemiU` previous result [10, Section 5], in which determinism is replaced by confluence.

Notably, `CM1_HALT ⪯ SMNdl_UB` relies on a variant of Hooper's argument [20]. The particular mechanization details span approximately 3500 lines of code, two thirds of which verify the construction in the proof of Lemma 26. Since the proof structure for uniform bound verification is mostly by extensive case analysis and basic arithmetic, the mechanization benefits greatly from proof automation, i.e. Coq's `lia`, `nia`, and `eauto` tactics [9]. To the best of the author's knowledge, the provided mechanization is the first that implements (a variant of) Hooper's argument.

[3] cf. `theories/SemiUnification/Reductions/HaltTM_1_chain_SemiU.v`
[4] cf. `theories/SemiUnification/Reductions/HaltTM_1_to_SemiU.v`

7 Conclusion

This work gives a constructive many-one reduction from Turing machine halting to semi-unification (Theorem 53). It improves upon existing work [27, 10] in the following aspects.

First, previous approaches use Turing reductions to establish undecidability. Therefore, such arguments are unable to establish many-one completeness of semi-unification shown in the present work (Corollary 54).

Second, previous work relies on the undecidability of Turing machine immortality, which is not recursively enumerable, and obscures the overall picture. In the present work, we avoid Turing machine immortality by adapting Hooper's ingenious construction [20] (also adapted in [24]) to uniform boundedness (Lemma 26). Notably, Hooper's construction is such that the resulting machine is either both mortal and uniformly bounded, or neither [27, Proof of Corollary 6].

Third, correctness of the reduction function is proven constructively (in the sense of axiom-free Coq), whereas previous work uses the principle of excluded middle, König's lemma [27], or the fan theorem [10]. As a result, anti-classical theories, such as synthetic computability theory [5], may accommodate the presented results.

Fourth, computability of the many-one reduction function from Turing machines to semi-unification instances is established rigorously by its mechanization in the Coq proof assistant. Traditionally, this aspect is treated less formally.

Fifth, the reduction is mechanized as part of the Coq Library of Undecidability Proofs [17], building upon existing infrastructure. Arguably, a comprehensive mechanization is the *only* feasible approach to verify a reduction from Turing machine halting to semi-unification with high confidence in full detail. The provided mechanization integrates existing work [10] into the Coq Library of Undecidability Proofs, and contributes a (first of its kind) mechanized variant of Hooper's construction to avoid symbol search.

While this document provides a high-level overview over the overall argument, surveyability (both local and global in the sense of [4]) is established mechanically. Local surveyability is supported by the modular nature of the Coq Library of Undecidability Proofs. That is, the mechanization of each reduction step can be understood and verified independently. Global surveyability is supported by `Theorem HaltTM_1_chain_SemiU` and the statement `HaltTM 1` $\preceq$ `SemiU` (cf. Section 6). That is, the individually mechanized reduction steps do compose transitively.

As ultimate proof of feasibility, the provided mechanization shows the maturity of the Coq proof assistant for mechanical verification of technically challenging proofs. Admittedly, neither Hooper's exact immortality construction [20] nor the exact semi-unification construction by Kfoury, Tiuryn, and Urzyczyn [27] was mechanized. Rather, the overall argument was revised to be mechanization-friendly. For example, the simplicity and uniformity of one-counter machines as an intermediate model of computation serves exactly this purpose.

Already, building upon the present work, there is a novel mechanization showing the undecidability of System F typability and type checking [11]. In addition, we envision further mechanized results. For one, the undecidability of synchronous distributivity [1] relies on uniform boundedness of semi-Thue systems that can be described as the presented (and mechanized) two-stack machines. Further, since the underlying construction is already implemented, it is reasonable to mechanize a many-one reduction from Turing machine halting to Turing machine immortality. This would pave the way for further mechanized results. For example, the undecidability of the finite variant property [6, Section 7] as well as several tiling problems [23] rely on (variants of) Turing machine immortality.

References

1 Siva Anantharaman, Serdar Erbatur, Christopher Lynch, Paliath Narendran, and Michaël Rusinowitch. Unification modulo synchronous distributivity. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning – 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 14–29. Springer, 2012. doi:10.1007/978-3-642-31365-3_4.

2 Andrea Asperti and Wilmer Ricciotti. A formalization of multi-tape Turing machines. *Theor. Comput. Sci.*, 603:23–42, 2015. doi:10.1016/j.tcs.2015.07.013.

3 Franz Baader, Wayne Snyder, Paliath Narendran, Manfred Schmidt-Schauß, and Klaus U. Schulz. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001. doi:10.1016/b978-044450813-3/50010-2.

4 O. Bradley Bassler. The surveyability of mathematical proof: A historical perspective. *Synth.*, 148(1):99–133, 2006. doi:10.1007/s11229-004-6221-7.

5 Andrej Bauer. First steps in synthetic computability theory. In Martín Hötzel Escardó, Achim Jung, and Michael W. Mislove, editors, *Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2005, Birmingham, UK, May 18-21, 2005*, volume 155 of *Electronic Notes in Theoretical Computer Science*, pages 5–31. Elsevier, 2005. doi:10.1016/j.entcs.2005.11.049.

6 Christopher Bouchard, Kimberly A. Gero, Christopher Lynch, and Paliath Narendran. On forward closure and the finite variant property. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems – 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2013. doi:10.1007/978-3-642-40885-4_23.

7 John H. Conway. Fractran: A simple universal programming language for arithmetic. In *Open Problems in Communication and Computation*, pages 4–26. Springer, 1987.

8 The Coq Proof Assistant. https://coq.inria.fr/. Accessed: 2020-10-08.

9 The Coq Proof Assistant Reference Manual. https://coq.inria.fr/distrib/current/refman/. Accessed: 2020-07-30.

10 Andrej Dudenhefner. Undecidability of semi-unification on a napkin. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.9.

11 Andrej Dudenhefner. The undecidability of system F typability and type checking for reductionists. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–10. IEEE, 2021. doi:10.1109/LICS52264.2021.9470520.

12 Manuel Fähndrich, Jakob Rehof, and Manuvir Das. Scalable context-sensitive flow analysis using instantiation constraints. In Monica S. Lam, editor, *Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Vancouver, Britith Columbia, Canada, June 18-21, 2000*, pages 253–263. ACM, 2000. doi:10.1145/349299.349332.

13 Yannick Forster, Edith Heiter, and Gert Smolka. Verification of PCP-related computational reductions in Coq. In Jeremy Avigad and Assia Mahboubi, editors, *Interactive Theorem Proving – 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2018. doi:10.1007/978-3-319-94821-8_15.

14 Yannick Forster and Fabian Kunze. A certifying extraction with time bounds from Coq to call-by-value lambda calculus. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITP.2019.17.

15 Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified programming of Turing machines in Coq. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 114–128. ACM, 2020. doi:10.1145/3372885.3373816.

16 Yannick Forster and Dominique Larchey-Wendling. Certified undecidability of intuitionistic linear logic via binary stack machines and Minsky machines. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 104–117. ACM, 2019. doi:10.1145/3293880.3294096.

17 Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq library of undecidable problems. In *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020)*, 2020. URL: https://github.com/uds-psl/coq-library-undecidability.

18 Jean-Yves Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. Thèse d'État, Éditeur inconnu, 1972.

19 Fritz Henglein. Type inference with polymorphic recursion. *ACM Trans. Program. Lang. Syst.*, 15(2):253–289, 1993. doi:10.1145/169701.169692.

20 Philip K. Hooper. The undecidability of the Turing machine immortality problem. *J. Symb. Log.*, 31(2):219–234, 1966. doi:10.2307/2269811.

21 Said Jahama and Assaf J. Kfoury. A general theory of semi-unification. Technical report, Boston University Computer Science Department, 1993.

22 Emmanuel Jeandel. On immortal configurations in Turing machines. In S. Barry Cooper, Anuj Dawar, and Benedikt Löwe, editors, *How the World Computes – Turing Centenary Conference and 8th Conference on Computability in Europe, CiE 2012, Cambridge, UK, June 18-23, 2012. Proceedings*, volume 7318 of *Lecture Notes in Computer Science*, pages 334–343. Springer, 2012. doi:10.1007/978-3-642-30870-3_34.

23 Jarkko Kari. The tiling problem revisited (extended abstract). In Jérôme Olivier Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*, volume 4664 of *Lecture Notes in Computer Science*, pages 72–79. Springer, 2007. doi:10.1007/978-3-540-74593-8_6.

24 Jarkko Kari and Nicolas Ollinger. Periodicity and immortality in reversible computing. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2008. doi:10.1007/978-3-540-85238-4_34.

25 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. The undecidability of the semi-unification problem (preliminary report). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 468–476. ACM, 1990. doi:10.1145/100216.100279.

26 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. Type reconstruction in the presence of polymorphic recursion. *ACM Trans. Program. Lang. Syst.*, 15(2):290–311, 1993. doi:10.1145/169701.169687.

27 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. The undecidability of the semi-unification problem. *Inf. Comput.*, 102(1):83–101, 1993. doi:10.1006/inco.1993.1003.

28 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. *J. ACM*, 41(2):368–398, 1994. doi:10.1145/174652.174659.

29 Dominique Larchey-Wendling and Yannick Forster. Hilbert's tenth problem in Coq. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.27.

30 Hans Leiß. Polymorphic recursion and semi-unification. In Egon Börger, Hans Kleine Büning, and Michael M. Richter, editors, *CSL '89, 3rd Workshop on Computer Science Logic, Kaiserslautern, Germany, October 2-6, 1989, Proceedings*, volume 440 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 1989. doi:10.1007/3-540-52753-2_41.

31 Hans Leiß and Fritz Henglein. A decidable case of the semi-unification problem. In Andrzej Tarlecki, editor, *Mathematical Foundations of Computer Science 1991, 16th International Symposium, MFCS'91, Kazimierz Dolny, Poland, September 9-13, 1991, Proceedings*, volume 520 of *Lecture Notes in Computer Science*, pages 318–327. Springer, 1991. doi:10.1007/3-540-54345-7_75.

32 Yuri V. Matiyasevich and Géraud Sénizergues. Decision problems for semi-Thue systems with a few rules. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 523–531. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561469.

33 M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

34 Alan Mycroft. Polymorphic type schemes and recursive definitions. In Manfred Paul and Bernard Robinet, editors, *International Symposium on Programming, 6th Colloquium, Toulouse, France, April 17-19, 1984, Proceedings*, volume 167 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 1984. doi:10.1007/3-540-12925-1_41.

35 Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

36 Paul Walton Purdom. Detecting looping simplifications. In Pierre Lescanne, editor, *Rewriting Techniques and Applications, 2nd International Conference, RTA-87, Bordeaux, France, May 25-27, 1987, Proceedings*, volume 256 of *Lecture Notes in Computer Science*, pages 54–61. Springer, 1987. doi:10.1007/3-540-17220-3_5.

37 John C. Reynolds. Towards a theory of type structure. In Bernard Robinet, editor, *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974. doi:10.1007/3-540-06859-7_148.

38 Hartley Rogers. *Theory of Recursive Functions and Effective Computability (Reprint from 1967)*. MIT Press, 1987. URL: http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=3182.

39 Joe B. Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Log.*, 98(1-3):111–156, 1999. doi:10.1016/S0168-0072(98)00047-5.

40 Arkadiusz Wojna. Counter machines. *Inf. Process. Lett.*, 71(5-6):193–197, 1999. doi:10.1016/S0020-0190(99)00116-7.

Dynamic Cantor Derivative Logic

David Fernández-Duque
Department of Mathematics, Ghent University, Belgium
Institute of Computer Science of the Czech Academy of Sciences, Prague, Czech Republic

Yoàv Montacute
Computer Laboratory, University of Cambridge, UK

Abstract

Topological semantics for modal logic based on the Cantor derivative operator gives rise to derivative logics, also referred to as d-logics. Unlike logics based on the topological closure operator, d-logics have not previously been studied in the framework of dynamical systems, which are pairs (X, f) consisting of a topological space X equipped with a continuous function $f\colon X \to X$.

We introduce the logics **wK4C**, **K4C** and **GLC** and show that they all have the finite Kripke model property and are sound and complete with respect to the d-semantics in this dynamical setting. In particular, we prove that **wK4C** is the d-logic of all dynamic topological systems, **K4C** is the d-logic of all T_D dynamic topological systems, and **GLC** is the d-logic of all dynamic topological systems based on a scattered space. We also prove a general result for the case where f is a homeomorphism, which in particular yields soundness and completeness for the corresponding systems **wK4H**, **K4H** and **GLH**.

The main contribution of this work is the foundation of a general proof method for finite model property and completeness of dynamic topological d-logics. Furthermore, our result for **GLC** constitutes the first step towards a proof of completeness for the trimodal topo-temporal language with respect to a finite axiomatisation – something known to be impossible over the class of all spaces.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases dynamic topological logic, Cantor derivative, temporal logic, modal logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.19

Related Version *Full Version*: `https://arxiv.org/abs/2107.10349`

1 Introduction

Dynamic (topological) systems are mathematical models of processes that may be iterated indefinitely. Formally, they are defined as pairs $\langle \mathfrak{X}, f\rangle$ consisting of a topological space $\mathfrak{X} = \langle X, \tau\rangle$ and a continuous function $f\colon X \to X$; the intuition is that points in the space $\mathfrak{X}$ "move" along their orbit, $x, f(x), f^2(x), \dots$ which usually simulates changes in time. *Dynamic topological logic* (**DTL**) combines modal logic and its topological semantics with linear temporal logic (see Pnueli [23]) in order to reason about dynamical systems in a decidable framework.

Due to their rather broad definition, dynamical systems are routinely used in many pure and applied sciences, including computer science. To cite some recent examples, in data-driven dynamical systems, data-related problems may be solved through data-oriented research in dynamical systems as suggested by Brunton and Kutz [4]. Weinan [6] proposes a dynamic theoretic approach to machine learning where dynamical systems are used to model nonlinear functions employed in machine learning. Lin and Antsaklis's [20] hybrid dynamical systems have been at the centre of research in control theory, artificial intelligence and computer-aided verification. Mortveit and Reidys's [22] sequential dynamical systems generalise aspects of systems such as cellular automata, and also provide a framework through which we can study dynamical processes in graphs. Another example of dynamical systems

and computer science can be found in the form of linear dynamical systems, i.e. systems with dynamics given by a linear transformation. Examples of such systems in computer science include Markov chains, linear recurrence sequences and linear differential equations. Moreover, there are known strong connections between dynamical systems and algorithms. This may be found for example in the work of Hanrot, Pujol and Stehlé [15], and in the work of Chu [5].

Such applications raise a need for effective formal reasoning about topological dynamics. Here, we may take a cue from modal logic and its topological semantics. The study of the latter dates back to McKinsey and Tarski [21], who proved that the modal logic **S4** is complete for a wide class of spaces, including the real line. Artemov, Davoren and Nerode [1] extended **S4** with a "next" operator in the spirit of **LTL**, producing the logic **S4C**. They proved that this logic is sound and complete with respect to the class of all dynamic topological systems. The system **S4C** was enriched with the "henceforth" tense by Kremer and Mints, who dubbed the new logic *dynamic topological logic* (**DTL**). Later, Konev et al. [16] showed that **DTL** is undecidable, and Fernández-Duque [11] showed that it is not finitely axiomatisable on the class of all dynamic topological spaces.

The aforementioned work on dynamic topological logic interprets the modal operator $\Diamond$ as a closure operator. However, McKinsey and Tarski had already contemplated semantics that are instead based on the Cantor derivative [21]: the *Cantor derivative* of a set A, usually denoted by $d(A)$, is the set of points x such that x is in the closure of $A \setminus \{x\}$ (see Section 2). This interpretation is often called *d-semantics* and the resulting logics are called *d-logics.* These logics were first studied in detail by Esakia, who showed that the d-logic **wK4** is sound and complete with respect to the class of all topological spaces [8]. It is well-known that semantics based on the Cantor derivative are more expressive than semantics based on the topological closure. For example, consider the property of a space $\mathfrak{X}$ being *dense-in-itself*, meaning that $\mathfrak{X}$ has no isolated points (see Section 3.2). The property of being dense-in-itself cannot be expressed in terms of the closure operator, but it *can* be expressed in topological d-semantics by the formula $\Diamond\top$.

Logics based on the Cantor derivative appear to be a natural choice for reasoning about dynamical systems. However, there are no established results of completeness for such logics in the setting of dynamical systems, i.e. when a topological space is equipped with a continuous function. Our goal is to prove the finite Kripke model property, completeness and decidability of logics with the Cantor derivative operator and the "next" operator $\bigcirc$ over some prominent classes of dynamical systems: namely, those based on arbitrary spaces, on T_D spaces (spaces validating the 4 axiom $\Box p \to \Box\Box p$) and on *scattered spaces* (see Section 3.2 for definitions). The reason for considering scattered spaces is to circumvent the lack of finite axiomatisability of **DTL** by restricting to a suitable subclass of all dynamical systems. In the study of dynamical systems and topological modal logic, one often works with *dense-in-themselves* spaces. This is a sensible consideration when modelling physical spaces, as Euclidean spaces are dense-in-themselves. However, as we will see in Section 3.2, some technical issues that arise when studying **DTL** over the class of all spaces disappear when restricting our attention to scattered spaces, which in contrast have many isolated points. Further, we consider dynamical systems where f is a *homeomorphism,* i.e. where f^{-1} is also a continuous function. Such dynamical systems are called *invertible.*

The basic dynamic d-logic we consider is **wK4C**, which consists of **wK4** and the temporal axioms for the continuous function f. In addition, we investigate two extensions of **wK4C**: **K4C** and **GLC**. As we will see, **K4C** is the d-logic of all T_D dynamical systems, and **GLC** is the d-logic of all dynamical systems based on a scattered space. Unlike the generic logic of the trimodal topo-temporal language $\mathcal{L}^{\circ *}_{\Diamond}$, we conjecture that a complete finite axiomatisation

for **GLC**, extended with axioms for the "henceforth" operator, will not require changes to the trimodal language. This logic is of special interest to us as it would allow for the first finite axiomatisation and completeness results for a logic based on the trimodal topo-temporal language.

Outline. This paper is structured as follows: in Section 2 we give the required definitions and notations necessary to understand the paper. In Section 3 we provide some background on prior work on the topic of dynamic topological logics. Moreover, we motivate our interest in **GLC**, the most unusual logic we work with.

In Section 4 we present the canonical model, and in Section 5 we construct a "finitary" accessibility relation on it. Both are then used in Section 6 in order to develop a proof technique that, given the right modifications, would work for many d-logics above **wK4C**. In particular, we use it to prove the finite model property, soundness and completeness for the d-logics **wK4C**, **K4C** and **GLC**, with respect to the appropriate classes of Kripke models.

In Section 7 we prove topological d-completeness of **K4C**, **wK4C** and **GLC** with respect to the appropriate classes of dynamical systems. In Section 8 we present logics for systems with homeomorphisms and provide a general completeness result which, in particular, applies to the d-logics **wK4H**, **K4H** and **GLH**. Finally, in Section 9 we provide some concluding remarks.

2 Preliminaries

In this section we review some basic notions required for understanding this paper. We work with the general setting of *derivative spaces*, in order to unify the topological and Kripke semantics of our logics.

▶ **Definition 1** (topological space). *A topological space is a pair $\mathfrak{X} = \langle X, \tau \rangle$, where X is a set and τ is a subset of $\wp(X)$ that satisfies the following conditions:*

- $X, \varnothing \in \tau$;
- *if* $U, V \in \tau$, *then* $U \cap V \in \tau$;
- *if* $\mathcal{U} \subseteq \tau$, *then* $\bigcup \mathcal{U} \in \tau$.

The elements of τ are called open sets, and we say that τ forms a topology on X. A complement of an open set is called a closed *set.*

The main operation on topological spaces we are interested in is the *Cantor derivative.*

▶ **Definition 2** (Cantor derivative). *Let $\mathfrak{X} = \langle X, \tau \rangle$ be a topological space. Given $S \subseteq X$, the* Cantor derivative *of S is the set $d(S)$ of all limit points of S, i.e. $x \in d(S) \iff \forall U \in \tau$ s.t. $x \in U$, $(U \cap S) \backslash \{x\} \neq \varnothing$. We may write $d(S)$ or dS indistinctly.*

When working with more than one topological space, we will often denote the Cantor derivative of the topological space $\langle X, \tau \rangle$ by d_τ. Given subsets $A, B \subseteq X$, it is not difficult to verify that $d = d_\tau$ satisfies the following properties:

1. $d(\varnothing) = \varnothing$;
2. $d(A \cup B) = d(A) \cup d(B)$;
3. $dd(A) \subseteq A \cup d(A)$.

In fact, these conditions lead to a more general notion of *derivative spaces:*[1]

[1] Derivative spaces are a special case of *derivative algebras* introduced by Esakia [9], where $\wp(X)$ is replaced by an arbitrary Boolean algebra.

▶ **Definition 3.** *A* derivative space *is a pair $\langle X, \rho\rangle$, where X is a set and $\rho\colon \wp(X) \to \wp(X)$ is a map satisfying properties 1-3 (with ρ in place of d).*

Accordingly, if $\mathfrak{X} = \langle X, \tau\rangle$ is a topological space and d_τ is the Cantor derivative on $\mathfrak{X}$, then $\langle X, d_\tau\rangle$ is a derivative space. However, there are other examples of derivative spaces. The standard topological closure may be defined by $c(A) = A \cup d_\tau(A)$. Then, $\langle X, c\rangle$ is also a derivative space, which satisfies the additional property $A \subseteq c(A)$ (and, *a fortiori,* $cc(A) = c(A)$); we call such derivative spaces *closure spaces.* For convenience, we denote the closure of the topological space $\langle X, \tau\rangle$ by c_τ.

Another example of derivative spaces comes from weakly transitive Kripke frames. For the sake of succinctness, we call these frames *derivative frames.* Below and throughout the text, we write $\exists x \sqsupset y\ \varphi$ instead of $\exists x(y \sqsubset x \wedge \varphi)$, and adopt a similar convention for the universal quantifier and other relational symbols.

▶ **Definition 4.** *A* derivative frame *is a pair $\mathfrak{F} = \langle W, \sqsubset\rangle$ where W is a non-empty set and $\sqsubset$ is a* weakly transitive *relation on W, meaning that $w \sqsubset v \sqsubset u$ implies that $w \sqsubseteq u$, where $\sqsubseteq$ is the reflexive closure of $\sqsubset$.*

We chose the notation $\sqsubset$ because it is suggestive of a transitive relation, but remains ambiguous regarding reflexivity, as there may be irreflexive and reflexive points. Note that $\sqsubset$ is weakly transitive iff $\sqsubseteq$ is transitive. Given $A \subseteq W$, we define $\downarrow_\sqsubset$ as a map $\downarrow_\sqsubset\colon \wp(W) \to \wp(W)$ such that

$$\downarrow_\sqsubset(A) = \{w \in W : \exists v \sqsupset w (v \in A)\}.$$

The following is then readily verified:

▶ **Lemma 5.** *If $\langle W, \sqsubset\rangle$ is a derivative frame, then $\langle W, \downarrow_\sqsubset\rangle$ is a derivative space.*

There is a connection between derivative frames and topological spaces. Given a derivative frame $\langle W, \sqsubset\rangle$, we define a topology $\tau_\sqsubset$ on W such that $U \in \tau_\sqsubset$ iff U is upwards closed under $\sqsubset$, in the sense that if $w \in U$ and $v \sqsupset w$ then $v \in U$. Topologies of this form are *Aleksandroff topologies.* The following is well known and easily verified.

▶ **Lemma 6.** *Let $\langle W, \sqsubset\rangle$ be a derivative frame and $\tau = \tau_\sqsubset$. Then, $d_\tau = \downarrow_\sqsubset$ iff $\sqsubset$ is irreflexive and $c_\tau = \downarrow_\sqsubset$ iff $\sqsubset$ is reflexive.*

Dynamical systems consist of a topological space equipped with a continuous function. Recall that if $\langle X, \tau\rangle$ and $\langle Y, \upsilon\rangle$ are topological spaces and $f\colon X \to Y$, then f is *continuous* if whenever $U \in \upsilon$, it follows that $f^{-1}(U) \in \tau$. The function f is *open* if $f(V)$ is open whenever V is open, and f is a *homeomorphism* if f is continuous, open and bijective. It is well known (and not hard to check) that f is continuous iff $c_\tau f^{-1}(A) \subseteq f^{-1}c_\upsilon(A)$ for any $A \subseteq Y$. By unfolding the definition of the closure operator, this becomes $f^{-1}(A) \cup d_\tau f^{-1}(A) \subseteq f^{-1}(A) \cup f^{-1}d_\upsilon(A)$, or equivalently, $d_\tau f^{-1}(A) \subseteq f^{-1}(A) \cup f^{-1}d_\upsilon(A)$. We thus arrive at the following definition.

▶ **Definition 7.** *Let $\langle X, \rho_X\rangle$ and $\langle Y, \rho_Y\rangle$ be derivative spaces. We say that $f\colon X \to Y$ is* continuous *if for all $A \subseteq Y$, $\rho_X f^{-1}(A) \subseteq f^{-1}(A) \cup f^{-1}\rho_Y(A)$. We say that f is a* homeomorphism *if it is bijective and $\rho_X f^{-1}(A) = f^{-1}\rho_Y(A)$.*

It is worth checking that these definitions coincide with their standard topological counterparts.

▶ **Lemma 8.** *If $\langle X,\tau\rangle$, $\langle Y,\upsilon\rangle$ are topological spaces with Cantor derivatives d_τ and d_υ respectively, and $f\colon X\to Y$, then*

1. *if f is continuous as a function between topological spaces, it is continuous as a function between derivative spaces, and*
2. *if f is a homeomorphism as a function between topological spaces, it is a homeomorphism as a function between derivative spaces.*

We are particularly interested in the case where $X=Y$, which leads to the notion of *dynamic derivative system.*

▶ **Definition 9.** *A* dynamic derivative system *is a triple $\mathfrak{S}=\langle X,\rho,f\rangle$, where $\langle X,\rho\rangle$ is a derivative space and $f\colon X\to X$ is continuous. If f is a homeomorphism, we say that $\mathfrak{S}$ is* invertible.

If $\mathfrak{S}=\langle X,\rho,f\rangle$ is such that $\rho=d_\tau$ for some topology τ, we say that $\mathfrak{S}$ is a *dynamic topological system* and identify it with the triple $\langle X,\tau,f\rangle$. If $\rho=\downarrow_\sqsubset$ for some weakly transitive relation $\sqsubset$, we say that $\mathfrak{S}$ is a *dynamic Kripke frame* and identify it with the triple $\langle X,\sqsubset,f\rangle$.

It will be convenient to characterise dynamic Kripke frames in terms of the relation $\sqsubset$.

▶ **Definition 10** (monotonicity and weak monotonicity)**.** *Let $\langle W,\sqsubset\rangle$ be a derivative frame. A function $f\colon W\to W$ is* monotonic *if $w\sqsubset v$ implies $f(w)\sqsubset f(v)$, and* weakly monotonic *if $w\sqsubset v$ implies $f(w)\sqsubseteq f(v)$.*

The function f is persistent *if it is a bijection and for all $w,v\in W$, $w\sqsubset v$ if and only if $f(w)\sqsubset f(v)$. We say that a Kripke frame is* invertible *if it is equipped with a persistent function.*

▶ **Lemma 11.** *If $\langle W,\sqsubset\rangle$ is a derivative frame and $f\colon W\to W$, then*

1. *if f is weakly monotonic then it is continuous with respect to $\downarrow_\sqsubset$, and*
2. *if f is persistent then it is a homeomorphism with respect to $\downarrow_\sqsubset$.*

Our goal is to reason about various classes of dynamic derivative systems using the logical framework defined in the next section.

3 Dynamic Topological Logics

In this section we discuss dynamic topological logic in the general setting of dynamic derivative systems. Given a non-empty set PV of propositional variables, the language $\mathcal{L}^\circ_\Diamond$ is defined recursively as follows:

$$\varphi ::= p \mid \varphi\wedge\varphi \mid \neg\varphi \mid \Diamond\varphi \mid \bigcirc\varphi,$$

where $p\in\mathsf{PV}$. It consists of the Boolean connectives $\wedge$ and $\neg$, the temporal modality $\bigcirc$, and the modality $\Diamond$ for the derivative operator with its dual $\Box:=\neg\Diamond\neg$. The interior modality may be defined by $\boxdot\varphi:=\varphi\wedge\Box\varphi$.

▶ **Definition 12** (semantics)**.** *A* dynamic derivative model *(DDM) is a quadruple $\mathfrak{M}=\langle X,\rho,f,\nu\rangle$ where $\langle X,\rho,f\rangle$ is a dynamic derivative system and $\nu\colon\mathsf{PV}\to\wp(X)$ is a valuation function assigning a subset of X to each propositional letter in PV. Given $\varphi\in\mathcal{L}^\circ_\Diamond$, we define the truth set $\|\varphi\|\subseteq X$ of φ inductively as follows:*

- $\|p\|=\nu(p)$;
- $\|\neg\varphi\|=X\setminus\|\varphi\|$;
- $\|\varphi\wedge\psi\|=\|\varphi\|\cap\|\psi\|$;
- $\|\Diamond\varphi\|=\rho(\|\varphi\|)$;
- $\|\bigcirc\varphi\|=f^{-1}(\|\varphi\|)$.

We write $\mathfrak{M},x\models\varphi$ if $x\in\|\varphi\|$, and $\mathfrak{M}\models\varphi$ if $\|\varphi\|=X$. We may write $\|\cdot\|_\mathfrak{M}$ or $\|\cdot\|_\nu$ instead of $\|\cdot\|$ when working with more than one model or valuation.

We define other connectives (e.g. $\vee, \to$) as abbreviations in the usual way. The fragment of $\mathcal{L}^\circ_\Diamond$ that includes only $\Diamond$ will be denoted by $\mathcal{L}_\Diamond$. Since our definition of the semantics applies to any derivative space and a general operator ρ, we need not differentiate in our results between d-logics, logics based on closure semantics and logics based on relational semantics. Instead, we indicate the specific class of derivative spaces to which the result applies.

In order to keep with the familiar axioms of modal logic, it is convenient to discuss the semantics of $\Box$. Accordingly, we define the dual of the derivative, called the *co-derivative*.

▶ **Definition 13** (co-derivative). *Let $\langle X, \rho\rangle$ be a derivative space. For each $S \subseteq X$ we define $\hat{\rho}(S) := X \backslash \rho(X \backslash S)$ to be the* co-derivative *of S.*

The co-derivative satisfies the following properties, where $A, B \subseteq X$:

1. $\hat{\rho}(X) = X$;
2. $A \cap \hat{\rho}(A) \subseteq \hat{\rho}\hat{\rho}(A)$;
3. $\hat{\rho}(A \cap B) = \hat{\rho}(A) \cap \hat{\rho}(B)$.

It can readily be checked that for any dynamic derivative model $\langle X, \rho, f, \nu\rangle$ and any formula φ, $\|\Box\varphi\| = \hat{\rho}(\|\varphi\|)$. The co-derivative can be used to define the standard *interior* of a set, given by $i(A) = A \cap \hat{\rho}(A)$ for each $A \subseteq X$. This implies that $U \subseteq \hat{\rho}(U)$ for each open set U, but not necessarily $\hat{\rho}(U) \subseteq U$. Next, we discuss the systems of axioms that are of interest to us. Let us list the axiom schemes and rules that we will consider in this paper:

Taut := All propositional tautologies
K := $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$
T := $\Box\varphi \to \varphi$
w4 := $\varphi \wedge \Box\varphi \to \Box\Box\varphi$
L := $\Box(\Box\varphi \to \varphi) \to \Box\varphi$
4 := $\Box\varphi \to \Box\Box\varphi$
$\text{Next}_\neg$:= $\neg\bigcirc\varphi \leftrightarrow \bigcirc\neg\varphi$

$\text{Next}_\wedge := \bigcirc(\varphi \wedge \psi) \leftrightarrow \bigcirc\varphi \wedge \bigcirc\psi$
C := $\bigcirc\varphi \wedge \bigcirc\Box\varphi \to \Box\bigcirc\varphi$
H := $\Box\bigcirc\varphi \leftrightarrow \bigcirc\Box\varphi$
MP := $\dfrac{\varphi \quad \varphi \to \psi}{\psi}$
$\text{Nec}_\Box := \dfrac{\varphi}{\Box\varphi}$
$\text{Nec}_\bigcirc := \dfrac{\varphi}{\bigcirc\varphi}$

The "base modal logic" over $\mathcal{L}_\Diamond$ is given by

$$\mathbf{K} := \text{Taut} + \text{K} + \text{MP} + \text{Nec}_\Box,$$

but we are mostly interested in proper extensions of $\mathbf{K}$. Let Λ, Λ' be logics over languages $\mathcal{L}$ and $\mathcal{L}'$. We say that Λ extends Λ' if $\mathcal{L}' \subseteq \mathcal{L}$ and all the axioms and rules of Λ' are derivable in Λ. A logic over $\mathcal{L}_\Diamond$ is *normal* if it extends $\mathbf{K}$. If Λ is a logic and φ is a formula, $\Lambda + \varphi$ is the least extension of Λ which contains every substitution instance of φ as an axiom.

We then define $\mathbf{wK4} := \mathbf{K} + \text{w4}$, $\mathbf{K4} := \mathbf{K} + 4$, $\mathbf{S4} := \mathbf{K4} + \text{T}$ and $\mathbf{GL} := \mathbf{K4} + \text{L}$. These logics are well known and characterise certain classes of topological spaces and Kripke frames which we review below. In addition, for a logic Λ over $\mathcal{L}_\Diamond$, $\Lambda\mathbf{F}$ is the logic over $\mathcal{L}^\circ_\Diamond$ given by[2]

$$\Lambda\mathbf{F} := \Lambda + \text{Next}_\neg + \text{Next}_\wedge + \text{Nec}_\bigcirc.$$

This simply adds axioms of linear temporal logic to Λ, which hold whenever $\bigcirc$ is interpreted using a function. Finally, we define $\Lambda\mathbf{C} := \Lambda\mathbf{F} + \text{C}$ and $\Lambda\mathbf{H} := \Lambda\mathbf{F} + \text{H}$, which as we will see correspond to derivative spaces with a continuous function or a homeomorphism respectively. The following is well known and dates back to McKinsey and Tarski [21].

[2] Logics of the form $\Lambda\mathbf{F}$ correspond to dynamical systems with a possibly discontinuous function. We will not discuss discontinuous systems in this paper; see [1] for more information.

▶ **Theorem 14.** **S4** *is the logic of all topological closure spaces, the logic of all transitive, reflexive derivative frames, and the logic of the real line with the standard closure.*

The logic **K4** includes the axiom $\Box p \to \Box\Box p$, which is not valid over the class of all topological spaces. The class of spaces satisfying this axiom is denoted by T_D, defined as the class of spaces in which every singleton is the result of an intersection between an open set and a closed set. Moreover, Esakia showed that this is the logic of transitive derivative frames [9].

▶ **Theorem 15.** **K4** *is the logic of all T_D topological derivative spaces, as well as the logic of all transitive derivative frames.*

Many familiar topological spaces, including Euclidean spaces, satisfy the T_D property, making **K4** central in the study of topological modal logic. A somewhat more unusual class of spaces, which is nevertheless of particular interest to us, is the class of *scattered spaces.*

▶ **Definition 16** (scattered space)**.** *A topological space $\langle X, \tau\rangle$ is* scattered *if for every $S \subseteq X$, $S \subseteq d(S)$ implies $S = \varnothing$.*

This is equivalent to the more common definition of a scattered space where a topological space is called scattered if every non-empty subset has an isolated point. Scattered spaces are closely related to converse well-founded relations. Below, recall that $\langle W, \sqsubset\rangle$ is *converse well-founded* if there is no infinite sequence $w_0 \sqsubset w_1 \sqsubset \dots$ of elements in W.

▶ **Lemma 17.** *If $\langle W, \sqsubset\rangle$ is an irreflexive frame, then $\langle W, \tau_\sqsubset\rangle$ is scattered iff $\sqsubset$ is converse well-founded.*

▶ **Theorem 18** (Simmons [24] and Esakia [7])**.** **GL** *is the logic of all scattered topological derivative spaces, as well as the logic of all converse well-founded derivative frames and the logic of all finite, transitive, irreflexive derivative frames.*

Aside from its topological interpretation, the logic **GL** is of particular interest as it is also the logic of provability in Peano arithmetic, as was shown by Boolos [3]. Meanwhile, logics with the C and H axioms correspond to classes of dynamical systems.

▶ **Lemma 19.**

1. *If Λ is sound for a class of derivative spaces Ω, then $\Lambda\mathbf{C}$ is sound for the class of dynamic derivative systems $\langle X, \rho, f\rangle$, where $\langle X, \rho\rangle \in \Omega$ and f is continuous.*
2. *If Λ is sound for a class of derivative spaces Ω, then $\Lambda\mathbf{H}$ is sound for the class of dynamic derivative systems $\langle X, \rho, f\rangle$, where $\langle X, \rho\rangle \in \Omega$ and f is a homeomorphism.*

The above lemma is easy to verify from the definitions of continuous functions and homeomorphisms in the context of derivative spaces (Definition 7).

3.1 Prior Work

The study of dynamic topological logic originates with Artemov, Davoren and Nerode, who observed that it is possible to reason about dynamical systems within modal logic. They introduced the logic **S4C** and proved that it is decidable, as well as sound and complete for the class of all dynamic closure systems (i.e. dynamic derivative systems based on a closure space). Kremer and Mints [18] considered the logic **S4H**, and also showed it to be sound and complete for the class of dynamic closure systems where f is a homeomorphism.

The latter also suggested adding the "henceforth" operator, $*$, from Pnueli's linear temporal logic (**LTL**) [23], leading to the language we denote by $\mathcal{L}^{\circ *}_{\Diamond}$. The resulting trimodal system was named *dynamic topological logic* (**DTL**). Kremer and Mints offered

an axiomatisation for **DTL**, but Fernández-Duque proved that it is incomplete; in fact, **DTL** is not finitely axiomatisable [11]. Fernández-Duque also showed that **DTL** enjoys a natural axiomatisation when extended with the *tangled closure* [13]. In contrast, Konev et al. established that **DTL** over the class of dynamical systems with a homeomorphism is non-axiomatisable [17].

3.2 The tangled closure on scattered spaces

Our interest in considering the class of scattered spaces within dynamic topological logic is motivated by results of Fernández-Duque [13]. He showed that the set of valid formulas of $\mathcal{L}^{\circ *}_{\Diamond}$ over the class of all dynamic closure systems is not finitely axiomatisable. Nevertheless, he found a natural (yet infinite) axiomatisation by introducing the *tangled closure* and adding it to the language of **DTL** [10]. Here, we use the more general *tangled derivative*, as defined by Goldblatt and Hodkinson [14].

▶ **Definition 20** (tangled derivative). *Let $\langle X, d\rangle$ be a derivative space and let $\mathcal{S} \subseteq \wp(X)$. Given $A \subseteq X$, we say that $\mathcal{S}$ is tangled in A if for all $S \in \mathcal{S}$, $A \subseteq d(S \cap A)$. We define the* tangled derivative *of $\mathcal{S}$ as*

$$\mathcal{S}^* := \bigcup \{A \subseteq X : \mathcal{S} \text{ is tangled in } A\}.$$

The *tangled closure* is then the special case of the tangled derivative where d is a closure operator. Fernández-Duque's axiomatisation is based on the extended language $\mathcal{L}^{\circ *}_{\Diamond\!\!\!\!\Diamond}$. This language is obtained by extending $\mathcal{L}^{\circ *}_{\Diamond}$ with the following operation.

▶ **Definition 21** (tangled language). *We define $\mathcal{L}^{\circ *}_{\Diamond\!\!\!\!\Diamond}$ by extending the recursive definition of $\mathcal{L}^{\circ *}_{\Diamond}$ in such a way that if $\varphi_1, \ldots, \varphi_n \in \mathcal{L}^{\circ *}_{\Diamond\!\!\!\!\Diamond}$, then $\Diamond\!\!\!\!\Diamond\{\varphi_1, \ldots, \varphi_n\} \in \mathcal{L}^{\circ *}_{\Diamond\!\!\!\!\Diamond}$. The semantic clauses are then extended so that on any model $\mathfrak{M}$,*

$$\|\Diamond\!\!\!\!\Diamond\{\varphi_1, \ldots, \varphi_n\}\| = \{\|\varphi_1\|, \ldots, \|\varphi_n\|\}^*.$$

The logic **DGL** is an extension of **GLC** that includes the temporal operator $*$. Unlike the complete axiomatisation of **DTL** that requires the tangled operator, in the case of **DGL**, we should be able to avoid this and use the original spatial operator $\Diamond$ alone. This is due to the following:

▶ **Theorem 22.** *Let $\mathfrak{X} = \langle X, \tau\rangle$ be a scattered space and $\{\varphi_1, \ldots, \varphi_n\}$ a set of formulas. Then*

$$\Diamond\!\!\!\!\Diamond\{\varphi_1, \ldots, \varphi_n\} \equiv \bot.$$

This leads to the conjecture that the axiomatic system of Kremer and Mints [18], combined with **GL**, will lead to a finite axiomatisation for **DGL**. While such a result requires techniques beyond the scope of the present work, the completeness proof we present here for **GLC** is an important first step. Before proving topological completeness for this and the other logics we have mentioned, we show that they are complete and have the finite model property for their respective classes of dynamic derivative frames.

4 The Canonical Model

The first step in our Kripke completeness proof will be a fairly standard canonical model construction. A *maximal Λ-consistent set* (Λ-MCS) w is a set of formulas that is Λ-consistent, i.e. $w \nvdash_\Lambda \bot$, and every set of formulas that properly contains it is Λ-inconsistent.

Given a logic Λ over $\mathcal{L}^{\circ}_{\Diamond}$, let $\mathfrak{M}^{\Lambda}_{c} = \langle W_c, \sqsubset_c, g_c, \nu_c\rangle$ be the canonical model for Λ, where

1. W_c is the set of all Λ-MCSs;
2. $w \sqsubset_c v$ iff for all formulas φ, if $\Box\varphi \in w$, then $\varphi \in v$;
3. $g_c(w) = \{\varphi : \bigcirc\varphi \in w\}$;
4. $\nu_c(p) = \{w : p \in w\}$.

It can easily be verified that **wK4C** defines the class of all weakly transitive Kripke models with a weakly monotonic map. Moreover, **K4C** defines the class of all transitive Kripke models with a weakly monotonic map. We call these models **wK4C** models and **K4C** models respectively.

▶ **Lemma 23.** *If Λ extends* **wK4C***, then the canonical model for Λ is a* **wK4C** *model. If Λ extends* **K4C***, then the canonical model of Λ is a* **K4C** *model.*

It is a well-known fact that the transitivity axiom $\Box p \to \Box\Box p$ is derivable in **GL** (see [25]). Therefore, **GLC** extends the system **K4C**. The proofs of the following two lemmas are standard and can be found for example in [2].

▶ **Lemma 24** (existence lemma)**.** *Let Λ be a normal modal logic and let $\mathfrak{M}^{\Lambda}_{c} = \langle W_c, \sqsubset_c, g_c, \nu_c\rangle$. Then, for every $w \in W_c$ and every formula φ in Λ, if $\Diamond\varphi \in w$ then there exists a point $v \in W_c$ such that $w \sqsubset_c v$ and $\varphi \in v$.*

▶ **Lemma 25** (truth lemma)**.** *Let Λ be a normal modal logic. For every $w \in W_c$ and every formula φ in Λ,*

$$\mathfrak{M}^{\Lambda}_{c}, w \models \varphi \text{ iff } \varphi \in w.$$

▶ **Corollary 26.** *The logic* **wK4C** *is sound and complete with respect to the class of all weakly monotonic dynamic derivative frames, and* **K4C** *is sound and complete with respect to the class of all weakly monotonic, transitive dynamic derivative frames.*

5 A finitary accessibility relation

One key ingredient in our finite model property proof will be the construction of a "finitary" accessibility relation $\sqsubset_\Phi$ on the canonical model. This accessibility relation will have the property that each point has finitely many successors, yet the existence lemma will hold for formulas in a prescribed finite set Φ.

We define *the $\sqsubset_c$-cluster $C(w)$* for each point $w \in W_c$ as

$$C(w) = \{w\} \cup \{v : w \sqsubset_c v \sqsubset_c w\}.$$

▶ **Definition 27** (φ-final set)**.** *A set w is said to be a φ-final set (or point) if w is an MCS, $\varphi \in w$, and whenever $w \sqsubset_c v$ and $\varphi \in v$, it follows that $v \in C(w)$.*

Let us write $\sqsubseteq_c$ for the reflexive closure of $\sqsubset_c$. It will be convenient to characterise $\sqsubseteq_c$ in the canonical model syntactically. Recall that $\boxdot\varphi := \varphi \wedge \Box\varphi$.

▶ **Lemma 28.** *If Λ extends* **wK4C** *and $w, v \in W_c$, then $w \sqsubseteq_c v$ if and only if whenever $\boxdot\varphi \in w$, it follows that $\boxdot\varphi \in v$.*

We are now ready to prove the main result of this section regarding the existence of the finitary relation $\sqsubset_\Phi$.

▶ **Lemma 29.** *Let Λ extend* **wK4C** *and Φ be a finite set of formulas closed under subformulas. There is an auxiliary relation $\sqsubset_\Phi$ on the canonical model of Λ such that:*

- **(i)** *$\sqsubset_\Phi$ is a subset of $\sqsubset_c$;*
- **(ii)** *For each $w \in W$, the set $\sqsubset_\Phi(w)$ is finite;*
- **(iii)** *If $\Diamond\varphi \in w \cap \Phi$, then there exists $v \in W$ with $w \sqsubset_\Phi v$ and $\varphi \in v$;*
- **(iv)** *If $w \sqsubset_c v \sqsubset_c w$ then $\sqsubset_\Phi(w) \subseteq \sqsubseteq_\Phi(v)$;*
- **(v)** *$\sqsubset_\Phi$ is weakly transitive. Moreover, if Λ extends* **K4C** *then $\sqsubset_\Phi$ is transitive, and if Λ extends* **GLC** *then $\sqsubset_\Phi$ is irreflexive.*

6 Stories and Φ-morphisms

In this subsection we show that the logics **wK4C**, **K4C** and **GLC** have the finite model property by constructing finite models and truth preserving maps from these models to the canonical model.

If $\sqsubset$ is a weakly transitive relation on A, $\langle A, \sqsubset\rangle$ is tree-like if whenever $a \sqsubseteq c$ and $b \sqsubseteq c$, it follows that $a \sqsubseteq b$ or $b \sqsubseteq a$. We will use labelled tree-like structures called *moments* to record the "static" information at a point; that is, the structure involving $\sqsubset$, but not f.

▶ **Definition 30** (moment). *A Λ-moment is a structure $\mathfrak{m} = \langle|\mathfrak{m}|, \sqsubset_\mathfrak{m}, \nu_\mathfrak{m}, r_\mathfrak{m}\rangle$, where $\langle|\mathfrak{m}|, \sqsubset_\mathfrak{m}\rangle$ is a finite tree-like Λ frame with a root $r_\mathfrak{m}$, and $\nu_\mathfrak{m}$ is a valuation on $|\mathfrak{m}|$.*

In order to also record "dynamic" information, i.e. information involving the transition function, we will stack up several moments together to form a "story". Below, $\bigsqcup$ denotes a disjoint union.

▶ **Definition 31** (story). *A* story (with duration I) *is a structure $\mathfrak{S} = \langle|\mathfrak{S}|, \sqsubset_\mathfrak{S}, f_\mathfrak{S}, \nu_\mathfrak{S}, r_\mathfrak{S}\rangle$ such that there are $I < \omega$, moments $\mathfrak{S}_i = \langle|\mathfrak{S}_i|, \sqsubset_i, \nu_i, r_i\rangle$ for each $i \leq I$, and functions $(f_i)_{i<I}$ such that:*

1. $|\mathfrak{S}| = \bigsqcup_{i\leq I} |\mathfrak{S}_i|$;
2. $\sqsubset_\mathfrak{S} = \bigsqcup_{i\leq I} \sqsubset_i$;
3. $\nu_\mathfrak{S}(p) = \bigsqcup_{i\leq I} \nu_i(p)$ *for each variable p;*
4. $r_\mathfrak{S} = r_0$;
5. $f_\mathfrak{S} = \mathrm{Id}_I \cup \bigsqcup_{i<I} f_i$ *with $f_i \colon |\mathfrak{S}_i| \to |\mathfrak{S}_{i+1}|$ being a weakly monotonic map such that $f_i(r_i) = r_{i+1}$ for all $i < I$ (we say that f_i is* root preserving*), and Id_I is the identity on $|\mathfrak{S}_I|$.*

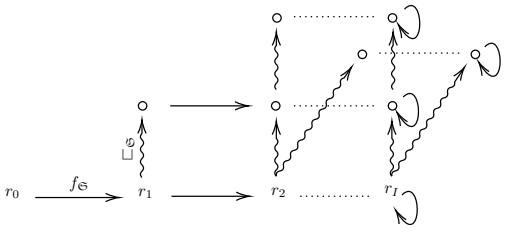


Figure 1 An example of a **GL**-story. The squiggly arrows represent the relation $\sqsubset_\mathfrak{S}$ while the straight arrows represent the function $f_\mathfrak{S}$. Each vertical slice represents a **GL**-moment. In the case of other types of stories, we may also have clusters besides singletons.

We often omit the subindices $\mathfrak{m}$ or $\mathfrak{S}$ when this does not lead to confusion. We may also assign different notations to the components of a moment, so that if we write $\mathfrak{m} = \langle W, \sqsubset, \nu, x\rangle$, it is understood that $W = |\mathfrak{m}|$, $\sqsubset = \sqsubset_{\mathfrak{m}}$, etc.

Recall that a *p-morphism* between Kripke models is a type of map that preserves validity. It can be defined in the context of dynamic derivative frames as follows:

▶ **Definition 32** (dynamic p-morphism). *Let $\mathfrak{M} = \langle W_{\mathfrak{M}}, \sqsubset_{\mathfrak{M}}, g_{\mathfrak{M}}\rangle$ and $\mathfrak{N} = \langle W_{\mathfrak{N}}, \sqsubset_{\mathfrak{N}}, g_{\mathfrak{N}}\rangle$ be dynamic derivative frames. Let $\pi\colon W_{\mathfrak{M}} \to W_{\mathfrak{N}}$. We say that π is a* dynamic p-morphism *if $w \sqsubset_{\mathfrak{M}} v$ implies that $\pi(w) \sqsubset_{\mathfrak{N}} \pi(v)$, $\pi(w) \sqsubset_{\mathfrak{N}} u$ implies that there is $v \sqsupset_{\mathfrak{M}} w$ with $\pi(v) = u$, and $\pi \circ g_{\mathfrak{M}} = g_{\mathfrak{N}} \circ \pi$.*

It is then standard that if $\pi\colon W_{\mathfrak{M}} \to W_{\mathfrak{N}}$ is a surjective, dynamic p-morphism, then any formula valid on $\mathfrak{M}$ is also valid on $\mathfrak{N}$. However, our relation $\sqsubset_\Phi$ will allow us to weaken these conditions and still obtain maps that preserve the truth of (some) formulas.

▶ **Definition 33** (Φ-morphism). *Fix a logic Λ and let $\mathfrak{M}^\Lambda_{\mathrm{c}} = \langle W_{\mathrm{c}}, \sqsubset_{\mathrm{c}}, g_{\mathrm{c}}, \nu_{\mathrm{c}}\rangle$ and $\mathfrak{S}$ be a story of duration I. A map $\pi : |\mathfrak{S}| \to W_{\mathrm{c}}$ is called a* dynamic Φ-morphism *if for all $x \in |\mathfrak{S}|$ the following conditions are satisfied:*

1. *$x \in \nu_{\mathfrak{S}}(p) \iff p \in \pi(x)$;*
2. *If $x \in |\mathfrak{S}_i|$ for some $i < I$, then $g_{\mathrm{c}}(\pi(x)) = \pi(f_{\mathfrak{S}}(x))$;*
3. *If $x \sqsubset_{\mathfrak{S}} y$ then $\pi(x) \sqsubset_{\mathrm{c}} \pi(y)$;*
4. *If $\pi(x) \sqsubset_\Phi v$ for some $v \in W_{\mathrm{c}}$, then there exists $y \in |\mathfrak{S}|$ such that $x \sqsubset_{\mathfrak{S}} y$ and $v = \pi(y)$.*

If we drop condition 2, we say that π is a Φ-morphism.

We now show that a dynamic Φ-morphism π preserves the truth of formulas of suitable $\bigcirc$-depth, where the latter is defined as usual in terms of nested occurrences of $\bigcirc$ in a formula φ.

▶ **Lemma 34** (truth preservation). *Let $\mathfrak{S}$ be a story of duration I and $x \in |\mathfrak{S}_0|$. Let π be a dynamic Φ-morphism to the canonical model of some normal logic Λ over $\mathcal{L}^\circ_\Diamond$. Suppose that $\varphi \in \Phi$ is a formula of $\bigcirc$-depth at most I. Then $\varphi \in \pi(x)$ iff $x \in \|\varphi\|_{\mathfrak{S}}$.*

We will next demonstrate that for every point w in the canonical model, there exists a suitable moment $\mathfrak{m}$ and a Φ-morphism mapping $\mathfrak{m}$ to w. In order to do this, we define a procedure for constructing new moments from smaller ones.

▶ **Definition 35** (moment construction). *Let $\Lambda \in \{\mathbf{wK4}, \mathbf{K4}, \mathbf{GL}\}$ and $C' \cup \{x\} \subseteq C(x)$ for some x in the canonical model $\mathfrak{M}^\Lambda_{\mathrm{c}}$. Let $\vec{\mathfrak{a}} = \langle \mathfrak{a}_m\rangle_{m<N}$ be a sequence of moments. We define a structure $\mathfrak{n} = \binom{\vec{\mathfrak{a}}}{C'}_x$ as follows:*

1. *$|\mathfrak{n}| = C' \sqcup \bigsqcup_{m<N} |\mathfrak{a}_m|$;*
2. *$y \sqsubset_{\mathfrak{n}} z$ if either*
   - *$y, z \in C'$, $\Lambda \neq \mathbf{GL}$ and $y \sqsubset_{\mathrm{c}} z$,*
   - *$y \in C'$ and $z \in |\mathfrak{a}_m|$ for some m, or*
   - *$y, z \in |\mathfrak{a}^m|$ and $y \sqsubset_{\mathfrak{a}_m} z$ for some m;*
3. *$\nu_{\mathfrak{n}}(p) = \{x \in C' : x \in \nu_{\mathrm{c}}(p)\} \sqcup \bigsqcup_{m<N} \nu_{\mathfrak{a}_m}(p)$;*
4. *$r_{\mathfrak{n}} = x$.*

▶ **Lemma 36.** *Given $\Lambda \in \{\mathbf{wK4C}, \mathbf{K4C}, \mathbf{GLC}\}$, for all $w \in W_{\mathrm{c}}$ there exists a Λ-moment $\mathfrak{m}$ and a Φ-morphism $\pi\colon |\mathfrak{m}| \to W_{\mathrm{c}}$ such that $\pi(r_{\mathfrak{m}}) = w$.*

Proof sketch. We prove the stronger claim that there is a moment $\mathfrak{m}$ and a map $\pi\colon |\mathfrak{m}| \to W_{\mathrm{c}}$ that is a p-morphism on the structure $(W_{\mathrm{c}}, \sqsubset_\Phi)$. We will say that π is a *p-morphism with respect to* $\sqsubset_\Phi$. Let $\sqsubset_\Phi^1$ be the strict $\sqsubset_\Phi$ successor, i.e. $w \sqsubset_\Phi^1 v$ iff $w \sqsubset_\Phi v$ and $\neg(v \sqsubset_\Phi w)$. Since $\sqsubset_\Phi^1$ is converse well-founded, we can assume inductively that for each v such that $w \sqsubset_\Phi^1 v$, there is a moment $\mathfrak{m}_v$ and a p-morphism $\pi_v : |\mathfrak{m}_v| \to W_{\mathrm{c}}$ with respect to $\sqsubset_\Phi$ that maps the root of $\mathfrak{m}_v$ to v. Accordingly, we define a moment

$$\mathfrak{m} = \begin{pmatrix} \{\mathfrak{m}_v : v \sqsupset_\Phi^1 w\} \\ C_\Phi(w) \end{pmatrix}_w.$$

It is not difficult to verify that $\mathfrak{m}$ thus defined is a Λ-moment.

Next we define a map $\pi : |\mathfrak{m}| \to W_{\mathrm{c}}$ as

$$\pi(x) = \begin{cases} x & \text{if } x \in C_\Phi(w), \\ \pi_v(x) & \text{if } x \in |\mathfrak{m}_v|. \end{cases}$$

The map π can be shown to be a p-morphism for $\sqsubset_\Phi$. Hence $\mathfrak{m}$ is a Λ-moment and π is a Φ-morphism such that $\pi(r_{\mathfrak{m}}) = w$, as required. ◀

We next define the notions of *pre-Φ-morphism* and *quotient moment* that will be essential for the rest of the proof.

▶ **Definition 37** (pre-Φ-morphism, Λ-bottom). *Let $\mathfrak{m}$ be a moment and $\pi : |\mathfrak{m}| \to W_{\mathrm{c}}$. We say that $x \in |\mathfrak{m}|$ is* at the Λ-bottom *for $\Lambda \in \{\mathbf{wK4}, \mathbf{K4}, \mathbf{GL}\}$ if*

- *$\Lambda \neq \mathbf{GL}$ and $\pi(x) \in C(\pi(r_{\mathfrak{m}}))$, or*
- *$\Lambda = \mathbf{GL}$ and for all $y \sqsubset_{\mathfrak{m}} x$, $\pi(y) = \pi(x)$.*

We will refer to "Λ-bottom" simply as "bottom" when this does not lead to confusion.

We say that $\pi : |\mathfrak{m}| \to W_{\mathrm{c}}$ is a pre-Φ-morphism *if it fulfils conditions 1 and 4 of a Φ-morphism (Definition 33), and $x \sqsubset_{\mathfrak{m}} y$ implies that either $\pi(x) \sqsubset_{\mathrm{c}} \pi(y)$ or x, y are at the bottom.*

▶ **Definition 38** (quotient moment). *Let $\Lambda \in \{\mathbf{wK4C}, \mathbf{K4C}, \mathbf{GLC}\}$. Let $\mathfrak{m}$ be a Λ-moment and $\pi : |\mathfrak{m}| \to W_{\mathrm{c}}$ be a pre-Φ-morphism. We define $x \sim y$ if either $x = y$, or x, y are at the bottom and $\pi(x) = \pi(y)$. Given $y \in |\mathfrak{m}|$ we set $[y] = \{z : z \sim y\}$.*

The quotient moment *$\mathfrak{m}/\pi$ of $\mathfrak{m}$ and its respective map $[\pi] : |\mathfrak{m}/\pi| \to W_{\mathrm{c}}$ are defined as follows, where $x, y \in |\mathfrak{m}|$:*

(a) $|\mathfrak{m}/\pi| = \{[x] : x \in |\mathfrak{m}|\}$;

(b) $[x] \sqsubset_{\mathfrak{m}/\pi} [y]$ *iff one of the following conditions is satisfied:*
- *x, y are at the bottom, $\pi(x) \sqsubset_{\mathrm{c}} \pi(y)$, and $\Lambda \neq \mathbf{GLC}$;*
- *x is at the bottom and y is not at the bottom;*
- *x, y are not at the bottom and $x \sqsubset_{\mathfrak{m}} y$;*

(c) $\nu_{\mathfrak{m}/\pi}(p) = \{[x] : x \in \nu_{\mathfrak{m}}(p)\}$;

(d) $r_{\mathfrak{m}/\pi} = [r_{\mathfrak{m}}]$;

(e) $[\pi]([x]) = \pi(x)$.

The quotient moment and its respective map hold some essential properties for the derivation of this section's main result. The idea is that by constructing a Λ-moment m with an associated pre-Φ-morphism π, we get that $\mathfrak{m}/\pi$ is still a Λ-moment, but now $[\pi]$ is a proper Φ-morphism. The next few results make this intuition precise.

▶ **Lemma 39.** *For $\Lambda \in \{\mathbf{wK4}, \mathbf{K4}, \mathbf{GL}\}$, if $\mathfrak{m}$ is any Λ-moment and $\pi : |\mathfrak{m}| \to W_{\mathrm{c}}$ is a pre-Φ-morphism, then $\mathfrak{m}/\pi$ is a Λ-moment and $[\cdot]\colon |\mathfrak{m}| \to |\mathfrak{m}/\pi|$ is weakly monotonic and root-preserving.*

▶ **Proposition 40.** *If $\mathfrak{m}$ is a moment and $\pi\colon |\mathfrak{m}| \to W_{\mathrm{c}}$ is a pre-Φ-morphism, then the map $[\pi]$ is a well-defined Φ-morphism.*

Using these properties, we can prove the existence of an appropriate story that maps to the canonical model. This is based on the following useful lemma:

▶ **Lemma 41.** *Fix $\Lambda \in \{\mathbf{wK4C}, \mathbf{K4C}, \mathbf{GLC}\}$ and let $\mathfrak{M}_{\mathrm{c}}^{\Lambda} = \langle W_{\mathrm{c}}, \sqsubset_{\mathrm{c}}, g_{\mathrm{c}}, \nu_{\mathrm{c}}\rangle$. Let $\mathfrak{m}$ be a Λ-moment and suppose that there exists a Φ-morphism $\pi\colon |\mathfrak{m}| \to W_{\mathrm{c}}$. Then, there exists a moment $\mathfrak{n}$, a weakly monotonic map $f\colon |\mathfrak{m}| \to |\mathfrak{n}|$, and a Φ-morphism $\rho\colon |\hat{\mathfrak{n}}| \to W_{\mathrm{c}}$ such that $g_{\mathrm{c}} \circ \pi = \rho \circ f$.*

Proof. We proceed by induction on the height of $\mathfrak{m}$. Let C be the cluster of $r_{\mathfrak{m}}$ and let $\vec{\mathfrak{a}} = \langle \mathfrak{a}_n\rangle_{n<N}$ be the generated sub-models of the immediate strict successors of $r_{\mathfrak{m}}$; note that each $\mathfrak{a}_n$ is itself a moment of smaller height. By the induction hypothesis, there exist moments $\langle \mathfrak{a}'_n\rangle_{n<N}$, root-preserving, weakly monotonic maps $f_n\colon |\mathfrak{a}_n| \to |\mathfrak{a}'_n|$, and Φ-morphism $\rho_n\colon |\mathfrak{a}'_n| \to W_{\mathrm{c}}$ such that $g_{\mathrm{c}} \circ \pi_n = \rho_n \circ f_n$. Moreover, for each $v \sqsupset_{\Phi} g_{\mathrm{c}}(r_{\mathfrak{m}})$, by Lemma 36 there are $\mathfrak{b}_v$ and a Φ-morphism $\rho_v\colon |\mathfrak{b}_v| \to W_{\mathrm{c}}$ mapping the root of $\mathfrak{b}_v$ to v. Let $D = g_{\mathrm{c}}\pi(C) \cup C_{\Phi}(g_{\mathrm{c}}(x))$, and let $\hat{\mathfrak{n}} = \binom{\vec{\mathfrak{a}} * \vec{\mathfrak{b}}}{D}_{g_{\mathrm{c}}(w)}$. It is not difficult to verify that the maps ρ_v can be used to define a pre-Φ-morphism $\hat{\rho}$ from $\hat{\mathfrak{n}}$ to W_{c}. Let

$$\hat{f}(w) = \begin{cases} g_{\mathrm{c}}(w) & \text{if } w \in C, \\ f_n(w) & \text{if } w \in |\mathfrak{a}_n|. \end{cases}$$

It is easy to see that $\hat{f}\colon |\mathfrak{m}| \to |\hat{\mathfrak{n}}|$ is weakly monotonic and satisfies $g_{\mathrm{c}} \circ \pi = \hat{\rho} \circ \hat{f}$. Setting $\mathfrak{n} = \hat{\mathfrak{n}}/\hat{\rho}$, $f = [\hat{f}]$ and $\rho = [\hat{\rho}]$, Proposition 40 implies that $\mathfrak{n}$, f and ρ have the desired properties. ◀

▶ **Proposition 42.** *Fix $\Lambda \in \{\mathbf{wK4C}, \mathbf{K4C}, \mathbf{GLC}\}$. Given $I < \omega$ and $w \in W_{\mathrm{c}}$, there is a story $\mathfrak{S}$ of duration I and a dynamic Φ-morphism $\pi\colon |\mathfrak{S}| \to W_{\mathrm{c}}$ with $w = \pi(r_{\mathfrak{S}})$.*

Proof. Proceed by induction on I. For $I = 0$, this is essentially Lemma 36. Otherwise, by the induction hypothesis, assume that a story $\hat{\mathfrak{S}}$ of depth I and dynamic p-morphism $\hat{\pi}$ exist. By Lemma 41, there is a moment $\mathfrak{S}_{I+1}$, map $f_I\colon |\mathfrak{S}_I| \to |\mathfrak{S}_{I+1}|$, and Φ-morphism $\pi_{I+1}\colon |\mathfrak{S}_{I+1}| \to W_{\mathrm{c}}$ commuting with f_I. We define $\mathfrak{S}$ by adding $\mathfrak{S}_{I+1}$ to $\hat{\mathfrak{S}}$ in order to obtain the desired story. ◀

It follows that any satisfiable formula is also satisfiable on a finite story, hence satisfiable on a finite model, yielding the main result of this section.

▶ **Theorem 43.** *The logics* **wK4C**, **K4C** *and* **GLC** *are sound and complete for their respective class of finite dynamic Λ-frames.*

7 Topological d-completeness

In this section we establish completeness results for classes of dynamic topological systems with continuous functions. We begin with the logic **GLC**, simply because the topological d-completeness for **GLC** is almost immediate, given that a **GLC** model is already a dynamic derivative model based on a scattered space via the standard up-set topology.

▶ **Theorem 44.** **GLC** *is the d-logic of all dynamic topological systems based on a scattered space, and enjoys the finite model property for this class.*

In order to prove topological d-completeness for **wK4C**, we first provide a definition and some generalisations of known results. We use similar constructions as in e.g. [9].

▶ **Definition 45.** *Let $\mathfrak{F} = \langle W, \sqsubset, g\rangle$ be a* **wK4C***-frame and let W^{i} and W^{r} be the sets of irreflexive and reflexive points respectively. We define a new frame $\mathfrak{F}_{\oplus} = \langle W_{\oplus}, \sqsubset_{\oplus}, g_{\oplus}\rangle$, where*

1. $W_{\oplus} = (W^{\mathrm{i}} \times \{0\}) \cup (W^{\mathrm{r}} \times \{0, 1\})$;
2. $(w, i) \sqsubset_{\oplus} (v, j)$ *iff* $w \sqsubset v$ *and* $(w, i) \neq (v, j)$;
3. $g_{\oplus}(w, i) = (g(w), 0)$.

The following is standard [9] and easily verified.

▶ **Proposition 46.** *If $\mathfrak{F} = \langle W, \sqsubset, g\rangle$ is any dynamic derivative frame, then $\mathfrak{F}_{\oplus}$ is an irreflexive dynamic derivative frame and $\pi\colon W_{\oplus} \to W$ given by $\pi(w, i) = w$ is a surjective, dynamic p-morphism.*

Proposition 46 allows us to obtain topological completeness from Theorem 43, as Lemma 6 tells us that irreflexive Kripke frames are essentially Cantor derivative spaces. We thus obtain the following:

▶ **Theorem 47.** **wK4C** *is the d-logic of all dynamic topological systems, and enjoys the finite model property for this class.*

Finally we turn our attention to **K4C**. Unlike the other two logics, **K4C** (or even **K4**) does not have the topological finite model property, despite having the Kripke finite model property. In the case of Aleksandroff spaces, the class T_D is easy to describe.

▶ **Lemma 48.** *The Aleksandroff space of a* **K4***-frame $\mathfrak{F} = \langle W, \sqsubset\rangle$ is T_D if and only if $\sqsubset$ is antisymmetric, in the sense that $w \sqsubset v \sqsubset w$ implies $w = v$.*

If we moreover want the Kripke and the d-semantics to coincide on $\mathfrak{F}$, we need $\sqsubset$ to be irreflexive. Thus we wish to "unwind" $\mathfrak{F}$ to get rid of all non-trivial clusters. The following construction achieves this.

▶ **Definition 49.** *Let $\mathfrak{F} = \langle W, \sqsubset, g\rangle$ be a dynamic* **K4** *frame. We define a new frame $\vec{\mathfrak{F}} = \langle \vec{W}, \vec{\sqsubset}, \vec{g}\rangle$, where*

- *$\vec{W}$ is the set of all finite sequences $(w_0, \ldots, w_n)$, where $w_i \sqsubset w_{i+1}$ for all $i < n$;*
- *$\mathbf{w} \vec{\sqsubset} \mathbf{v}$ iff $\mathbf{w}$ is a strict initial segment of $\mathbf{v}$;*
- *$\vec{g}(w_0, \ldots, w_n)$ is the subsequence of $(g(w_0), \ldots, g(w_n))$, obtained by deleting every entry that is equal to its immediate predecessor.*

We moreover define a map $\pi\colon \vec{W} \to W$, where $\pi(w_0, \ldots, w_n) = w_n$.

In the definition of $\vec{g}$, note that g is only weakly monotonic, so it may be that, for instance, $g(w_0) = g(w_1)$; in this case, we include only one copy of $g(w_0)$ to ensure that $\vec{g}(\mathbf{w}) \in \vec{W}$.

▶ **Proposition 50.** *If $\mathfrak{F} = \langle W, \sqsubset, g\rangle$ is any dynamic* **K4** *frame, then $\vec{\mathfrak{F}}$ is an antisymmetric and irreflexive dynamic* **K4** *frame. Moreover, $\pi\colon \vec{W} \to W$ is a surjective, dynamic p-morphism.*

Similar constructions have already appeared in e.g. [16]. From this we obtain the following:

▶ **Theorem 51.** **K4C** *is the d-logic of the class of all dynamic topological systems based on a T_D space, as well as the d-logic of the class of all dynamic topological systems based on an Aleksandroff T_D space.*

8 Invertible Systems

Recall that if $\langle X, \tau \rangle$ is a topological space and $f\colon X \to X$ is a function, then f is a *homeomorphism* if it is a bijection and both f and f^{-1} are continuous.

Unlike in the continuous case, for logics with homeomorphisms, every formula is equivalent to a formula where all occurrences of $\bigcirc$ are applied to atoms. More formally, we say that a formula is in $\bigcirc$-*normal form* if it is of the form $\varphi(\bigcirc^{k_0} p_0, \dots, \bigcirc^{k_n} p_n)$, where $\varphi(p_0, \dots, p_n)$ does not contain $\bigcirc$, the p_i's are variables, and the k_i's are natural numbers. It is readily observed that the axiom H allows us to "push" all instances of $\bigcirc$ to the propositional level. Thus we obtain the following useful representation lemma:

▶ **Lemma 52.** *For every logic Λ and every formula φ, there is a formula φ' in $\bigcirc$-normal form such that $\Lambda\mathbf{H} \vdash \varphi \leftrightarrow \varphi'$.*

As we shall see shortly, $\Lambda\mathbf{H}$ inherits completeness and the finite model property almost immediately from Λ. If $\mathfrak{X} = \langle X, \rho_X \rangle$ and $\mathfrak{Y} = \langle Y, \rho_Y \rangle$ are derivative spaces, then the *topological sum* is defined as $\mathfrak{X} \oplus \mathfrak{Y} = (X \oplus Y, \rho_X \oplus \rho_Y)$, where $X \oplus Y$ is the disjoint union of the two sets, and $\rho_X \oplus \rho_Y$ is given by $(\rho_X \oplus \rho_Y)A = \rho_X(A \cap X) \cup \rho_Y(A \cap Y)$. We say that a class Ω of derivative spaces is *closed under sums* if whenever $\mathfrak{A}, \mathfrak{B} \in \Omega$, it follows that $\mathfrak{A} \oplus \mathfrak{B} \in \Omega$.

Given a derivative space $\mathfrak{A} = \langle A, \rho \rangle$, we may write $\mathfrak{A}^n$ instead of $\mathfrak{A} \oplus \mathfrak{A} \oplus \ldots \oplus \mathfrak{A}$ (n times), and if $\mathfrak{A}$ has domain A, we may identify the domain of $\mathfrak{A}^n$ with $A^n = A \times \{0, \dots, n-1\}$. It should be clear that if $\mathfrak{A} \in \Omega$ and Ω is closed under sums, then $\mathfrak{A}^n \in \Omega$. Let $(m)_n$ denote the remainder of m modulo n. We then define a dynamical structure $\mathfrak{A}^{(n)} = (\mathfrak{A}^n, f)$, where $f\colon A^n \to A^n$ is given by $f(w, i) = (w, (i+1)_n)$.

▶ **Lemma 53.** *Let Ω be a class of derivative spaces closed under sums. Then, if $\mathfrak{A} \in \Omega$, it follows that $\mathfrak{A}^{(n)}$ is an invertible dynamic derivative system based on an element of Ω.*

For our proof of completeness, we need the notion of *extended valuation.*

▶ **Definition 54.** *Let $\mathfrak{A} = \langle A, \rho \rangle$ be a derivative space. Let $\mathsf{PV}^{\bigcirc}$ be the set of all expressions $\bigcirc^i p$, where $i \in \mathbb{N}$ and p is a propositional variable. An* extended valuation *on $\mathfrak{A}$ is a relation $\nu \subseteq \mathsf{PV}^{\bigcirc} \times A$.*

If ν is an extended valuation on $\mathfrak{A}$, we define a valuation on $\mathfrak{A}^{(n)}$ so that for any variable p and $(w, i) \in A^n$, $(w, i) \in \nu^{(n)}(p)$ if and only if $w \in \nu(\bigcirc^i p)$.

▶ **Lemma 55.** *Let $\mathfrak{A} = \langle A, \rho \rangle$ be any derivative space and ν be any extended valuation on $\mathfrak{A}$. Let $w \in A$, $i < n$, and φ be any formula in $\bigcirc$-normal form which has $\bigcirc$-depth less than $n - i$. Then, $\langle \mathfrak{A}^{(n)}, \nu^{(n)} \rangle, (w, i) \models \varphi$ if and only if $\langle \mathfrak{A}, \nu \rangle, w \models \varphi$.*

It then readily follows that φ is consistent if and only if φ' is consistent, which implies that φ' is satisfiable on Ω. This is equivalent to φ being satisfiable on the class of invertible systems based on Ω. From this, we obtain the following general result.

▶ **Theorem 56.** *Let Λ be complete for a class Ω of derivative spaces. Then, if Ω is closed under sums, it follows that $\Lambda\mathbf{H}$ is complete for the class of invertible systems based on Ω.*

▶ **Corollary 57.**

1. **wK4H** *is sound and complete for*
   a. *The class of all finite invertible dynamic* **wK4** *frames.*
   b. *The class of all finite invertible dynamic topological systems with Cantor derivative.*

2. **K4H** *is sound and complete for*
   a. *The class of all finite invertible dynamic* **K4** *frames.*
   b. *The class of all invertible, T_D dynamic topological systems with Cantor derivative.*
3. **GLH** *is sound and complete for*
   a. *The class of all finite invertible dynamic* **GL** *frames.*
   b. *The class of all finite invertible, scattered dynamic topological systems with Cantor derivative.*

9 Conclusion

We have recast dynamic topological logic in the more general setting of derivative spaces and established the seminal results for the $\{\Diamond, \bigcirc\}$-fragment for the variants of the standard logics with the Cantor derivative in place of the topological closure. Semantics on the Cantor derivative give rise to a richer family of modal logics than their counterparts based on closure. This is evident in the distinction between e.g. the logics **wKC** and **K4C**, both of which collapse to **S4C** when replaced by their closure-based counterparts. This line of research goes hand in hand with recent trends that consider the Cantor derivative as the basis of topological semantics [12, 19].

There are many natural problems that remain open. The logic **S4C** is complete for the Euclidean plane. In the context of *d*-semantics, the logic of the Euclidean plane is a strict extension of **K4**, given that punctured neighbourhoods are *connected* in the sense that they cannot be split into two disjoint, non-empty open sets. Thus one should not expect **K4C** to be complete for the plane. This raises the question: what is the dynamic *d*-logic of Euclidean spaces in general, and of the plane in particular?

The *d*-semantics also poses new lines of inquiry with respect to the class of functions considered. We have discussed continuous functions and homeomorphisms. Artemov et al. [1] also considered arbitrary functions, and we expect that the techniques used by them could be modified without much issue for *d*-semantics. On the other hand, in the setting of closure-based logics, the logic of spaces with continuous, open maps that are not necessarily bijective coincides with the logic of spaces with a homeomorphism. This is no longer true in the *d*-semantics setting, as the validity of the H axiom requires injectivity. Along these lines, the *d*-logic of *immersions* (i.e. continuous, injective functions) would validate the original continuity axiom $\bigcirc\Box p \to \Box\bigcirc p$. Thus there are several classes of dynamical systems whose closure-based logics coincide, but are split by *d*-semantics.

Finally, there is the issue of extending our language to the trimodal language with the "henceforth" operator. It is possible that the *d*-logic of all dynamic topological systems may be axiomatised using the tangled derivative, much as the tangled closure was used to provide an axiomatisation of the closure-based **DTL**. However, given that the tangled derivative is made trivial on scattered spaces, we conjecture that the trimodal *d*-logic based on this class will enjoy a natural, finite axiomatisation. The work presented here is an important first step towards proving this.

References

1 Sergei Artemov, Jennifer Davoren, and Anil Nerode. Modal logics and topological semantics for hybrid systems. *Technical Report MSI 97-05*, 1997.

2 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Scie.* Cambridge University Press, Cambridge, 2001.

3 George Boolos. The logic of provability. *The American Mathematical Monthly*, 91(8):470–480, 1984. doi:10.1080/00029890.1984.11971467.

4 Steven L. Brunton and J. Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

5 Moody T. Chu. Linear algebra algorithms as dynamical systems. *Acta Numerica*, 17:1–86, 2008.

6 Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

7 Leo Esakia. Diagonal constructions, Löb's formula and Cantor's scattered spaces. *Studies in logic and semantics*, 132(3):128–143, 1981.

8 Leo Esakia. Weak transitivity–restitution. In Nauka, editor, *Study in Logic*, volume 8, pages 244–254, 2001. In Russian.

9 Leo Esakia. Intuitionistic logic and modality via topology. *Ann. Pure Appl. Log.*, 127(1-3):155–170, 2004. doi:10.1016/j.apal.2003.11.013.

10 David Fernández-Duque. Tangled modal logic for spatial reasoning. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 857–862. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-149.

11 David Fernández-Duque. Non-finite axiomatizability of dynamic topological logic. *ACM Transactions on Computational Logic*, 15(1):4:1–4:18, 2014. doi:10.1145/2489334.

12 David Fernández-Duque and Petar Iliev. Succinctness in subsystems of the spatial μ-calculus. *Journal of Applied Logics*, 5(4):827–874, 2018. URL: https://www.collegepublications.co.uk/downloads/ifcolog00024.pdf.

13 David Fernández-Duque. A sound and complete axiomatization for dynamic topological logic. *The Journal of Symbolic Logic*, 77(3):947–969, 2012. doi:10.2178/jsl/1344862169.

14 Robert Goldblatt and Ian M. Hodkinson. Spatial logic of tangled closure operators and modal mu-calculus. *Annals of Pure and Applied Logic*, 168(5):1032–1090, 2017.

15 Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Annual Cryptology Conference*, pages 447–464. Springer, 2011.

16 Boris Konev, Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. Dynamic topological logics over spaces with continuous functions. In Guido Governatori, Ian M. Hodkinson, and Yde Venema, editors, *Advances in Modal Logic*, volume 6, pages 299–318, London, 2006. College Publications.

17 Boris Konev, Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. On dynamic topological and metric logics. *Studia Logica*, 84(1):129–160, 2006. doi:10.1007/s11225-006-9005-x.

18 Philip Kremer and Grigori Mints. Dynamic topological logic. *Annals of Pure and Applied Logic*, 131:133–158, 2005.

19 Andrey Kudinov. Topological modal logics with difference modality. In Guido Governatori, Ian M. Hodkinson, and Yde Venema, editors, *Advances in Modal Logic 6*, pages 319–332. College Publications, 2006. URL: http://www.aiml.net/volumes/volume6/Kudinov.ps.

20 Hai Lin and Panos J. Antsaklis. Hybrid dynamical systems: An introduction to control and verification. *Found. Trends Syst. Control*, 1(1):1–172, March 2014. doi:10.1561/2600000001.

21 John C. C. McKinsey and Alfred Tarski. The algebra of topology. *Annals of Mathematics*, 2:141–191, 1944.

22 Henning S. Mortveit and Christian M. Reidys. *An Introduction to Sequential Dynamical Systems*. Springer-Verlag, Berlin, Heidelberg, 2007.

23 Amir Pnueli. The temporal logic of programs. In *Proceedings 18th IEEE Symposium on the Foundations of CS*, pages 46–57, 1977.

24 Harold Simmons. Topological aspects of suitable theories. *Proceedings of the Edinburgh Mathematical Society*, 19(4):383–391, 1975. doi:10.1017/S001309150001049X.

25 Craig Smorynski. *Self-reference and modal logic*. Springer, 1985.

Global Winning Conditions in Synthesis of Distributed Systems with Causal Memory

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Manuel Gieseking
University of Oldenburg, Germany

Jesko Hecking-Harbusch
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ernst-Rüdiger Olderog
University of Oldenburg, Germany

Abstract

In the synthesis of distributed systems, we automate the development of distributed programs and hardware by automatically deriving correct implementations from formal specifications. For synchronous distributed systems, the synthesis problem is well known to be undecidable. For asynchronous systems, the boundary between decidable and undecidable synthesis problems is a long-standing open question. We study the problem in the setting of Petri games, a framework for distributed systems where asynchronous processes are equipped with causal memory. Petri games extend Petri nets with a distinction between system places and environment places. The components of a distributed system are the players of the game, represented as tokens that exchange information during each synchronization. Previous decidability results for this model are limited to local winning conditions, i.e., conditions that only refer to individual components.

In this paper, we consider global winning conditions such as mutual exclusion, i.e., conditions that refer to the state of all components. We provide decidability and undecidability results for global winning conditions. First, we prove for winning conditions given as bad markings that it is decidable whether a winning strategy for the system players exists in Petri games with a bounded number of system players and one environment player. Second, we prove for winning conditions that refer to both good and bad markings that it is undecidable whether a winning strategy for the system players exists in Petri games with at least two system players and one environment player. Our results thus show that, on the one hand, it is indeed possible to use global safety specifications like mutual exclusion in the synthesis of distributed systems. However, on the other hand, adding global liveness specifications results in an undecidable synthesis problem for almost all Petri games.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Synthesis, distributed systems, reactive systems, Petri games, decidability

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.20

Related Version *Full Version*: `https://arxiv.org/abs/2107.09280` [12]

Funding Supported by the German Research Foundation (DFG) Grant Petri Games (392735815) and Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) Grant OSARES (683300).

1 Introduction

The *synthesis problem* probes whether there exists an implementation for a formal specification and derives such an implementation if it exists. This approach automates the creation of systems. Engineers can think on a more abstract level about *what* a system should achieve instead of *how* the system should achieve its goal. The synthesis problem for a system

consisting of one component interacting with its environment is often encoded as a two-player game with complete observation between the *system* player and the *environment* player (cf. [24, 4, 2, 7, 29, 25, 23]). The system player tries to satisfy the winning condition of the game while the environment player tries to violate it. A winning strategy for the system player is a correct implementation as it encodes the system's reaction to all environment behaviors.

The synthesis problem for *distributed systems* aims to derive a correct implementation for every concurrent component of a distributed system. Each component can interact with its environment. Distributed systems can be differentiated depending on whether the components progress *synchronously* or *asynchronously*. For synchronous distributed systems, the synthesis problem is well known to be undecidable, as observed by Pnueli and Rosner [34]. For asynchronous distributed systems with causal memory, the boundary between decidable and undecidable synthesis problems is a long-standing open question [30, 15]. For the synthesis of asynchronous distributed systems, the memory model changes compared to the two-player game and the number of players increases to encode the different components of the system. In distributed systems, components observe only their local surroundings. This can be encoded by *causal memory* [16, 27, 17]: Two players share no information while they run concurrently; during every synchronization, however, they exchange their entire local histories, including all of their previous synchronizations with other players.

In this paper, we consider *reactive systems*, i.e., the components continually interact with their environment. *Control games* [17] based on *asynchronous automata* [39] and *Petri games* [15] are formalisms for the synthesis of asynchronous distributed reactive systems with causal memory. We focus on Petri games. Here, several system players play against several environment players in a Petri net. Tokens represent players and places either belong to the system or to the environment, resulting in a distribution of system and environment players. Deciding the existence of a winning strategy for the system players is EXPTIME-complete for Petri games with a bounded number of system players, one environment player, and bad places as local winning condition [15]. This also holds for Petri games with a bounded number of environment players, one system player, and bad markings as global winning condition [14]. Local winning conditions cannot express global properties like mutual exclusion.

We consider global winning conditions and contribute decidability and undecidability results regarding the synthesis of asynchronous distributed reactive systems with causal memory. In the first part of this paper, we prove that it is decidable whether a winning strategy for the system players exists in Petri games with a bounded number of system players, one environment player, and *bad markings* as global winning condition. Bad markings are a *safety* winning condition in the sense that they define markings as bad that the system players have to avoid in order to win the Petri game. Decidability is achieved by a reduction to a two-player game with complete observation and a Büchi winning condition. In the two-player game, it is encoded that transitions with the environment player fire as late as possible, i.e., transitions *without* the environment player fire before transitions *with* it. This order of transitions encodes causal memory [15]. For every sequential play of the two-player game, we need to check that no bad marking is reached for the different orders of fired concurrent transitions. The causal history of system players can grow infinitely large. We show that the finite causal history of each system player until its last synchronization with the environment player can be stored finitely and suffices to find bad markings.

In the second part of this paper, we investigate whether global winning conditions beyond bad markings are decidable. We report on two undecidability results to further underline the significance of our decidability result: We prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least two system players, one

environment player, and *good and bad markings* as winning condition. For this winning condition, no bad marking should be reached *until* a good marking is reached, which can be expressed in linear-time temporal logic (LTL) [33]. Notice that it is not required to terminate in a good marking. Good markings can be used to simulate the undecidable synchronous setting of Pnueli and Rosner [34] in the asynchronous setting of Petri games. This is realized by identifying executions as good if players deviate too much from the synchronous setting. Next, we prove that it is undecidable whether a winning strategy exists for the system players in Petri games with *good markings* and at least three players, out of which one is an environment player and each of the other two can change between being a system and an environment player. Good markings are a *liveness* winning condition in the sense that they define markings as good, one of which the system players have to reach in order to win the Petri game. Here, bad markings from the first undecidability result are encoded by repeatedly changing all players to environment players. With these results, we obtain an overview regarding decidability and undecidability for global winning conditions.

Related Work. A formal connection exists between Petri games and *control games* [17] based on *asynchronous automata* [39]: Petri games can be translated into control games and vice versa, at an exponential blow-up in each direction [1]. This translates decidability in acyclic communication architectures [17], originally obtained for control games, to Petri games, and decidability in single-process systems [14], originally obtained for Petri games, to control games. Further decidability results exist for control games with acyclic communication architectures [31]. Decidability has also been obtained for restrictions on the dependencies of actions [16] or on the synchronization behavior [26, 27] and for decomposable games [19].

The decidability result of this paper does not transfer to control games because the translation in [1] produces Petri games with as many system *and* environment players as there are processes in the control game. The undecidability results of this paper transfer to control games. System players in Petri games correspond to processes with only controllable actions in control games; environment players correspond to processes with only uncontrollable actions [1]. Bad markings from the first undecidability result can be simulated by additional uncontrollable actions for all processes preventing the reaching of good markings afterward.

For Petri games with several system and environment players, bounded synthesis is a semi-decision procedure to find winning strategies for the system players [8, 22, 21]. Bounded synthesis and the reduction for bad places are implemented in the tool AdamSYNT [13, 9, 18].

2 Motivating Example

We introduce the intuition behind Petri games and bad markings with the example in Fig. 1. There, we search for a strategy for two power plants, which should react to the energy production of renewable sources based on the weather forecast. A Petri game differentiates the places of a Petri net as *system* places (depicted in gray) and as *environment* places (depicted in white). For example, p is a system place whereas *forecast* is an environment place. The players of a Petri game are represented by tokens. The type of the place, where a token is residing, dictates whether the token represents a system or an environment player.

After transition *sunny* fires to indicate a sunny forecast, there are two system players in place p (each representing one power plant) and one environment player in place s. Causal memory implies that both system players know what the weather forecast predicts. They do not know whether the actual energy production is high (indicated by s_h firing) or low (indicated by s_l firing) producing three or two units of energy in place w. Nevertheless, each

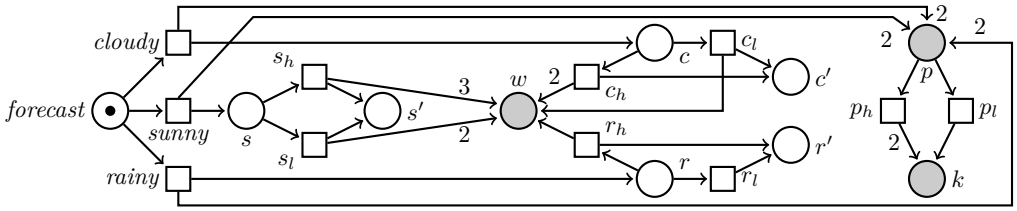


Figure 1 Two power plants observe if *sunny*, *cloudy*, or *rainy* weather is *forecast*. Depending on the actual weather, renewable sources produce up to three units of energy. The power plants produce one or two units of energy each and have to maintain the total energy production between four and five units of energy as all final markings with different energy production are bad markings.

power plant has to decide whether to produce two or one unit of energy in place k by p_h or p_l firing. The two power plants should produce together with the renewable sources either four or five units of energy. Therefore, any final marking resulting in a different energy production is a bad marking, i.e., the set of bad markings is $\{M : \mathcal{P} \to \mathbb{N} \mid (M(k) + M(w) < 4 \vee M(k) + M(w) > 5) \wedge \exists x \in \{s', c', r'\} : (M(x) = 1 \wedge \forall y \in \mathcal{P} \setminus \{x, k, w\} : M(y) = 0)\}$. The second conjunct ensures the marking being final by requiring that the only environment player is in one of the three environment places s', c', and r' and the system players are only in system places k and w. We assume that players always choose one of their successors. In Sec. 4, the finer notion of strategies being *deadlock-avoiding* is presented.

A winning strategy for the system players produces one unit of energy at both power plants for a sunny forecast, two units of energy at one power plant and one unit of energy at the other for a cloudy forecast, and two units of energy at both power plants for a rainy forecast. The specification is expressible with the local winning condition of bad places by having transitions from each bad marking leading to a bad place. This is only so as the example has no infinite behavior. For Petri games with infinite behavior and one environment player, the global winning condition of bad markings can specify losing behavior between players without requiring their synchronization which is impossible for local winning conditions.

3 Petri Nets

A *Petri net* [32, 37] $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ consists of the disjoint finite sets of *places* $\mathcal{P}$ and of *transitions* $\mathcal{T}$, the *flow relation* $\mathcal{F}$ as multiset over $(\mathcal{P}\times\mathcal{T})\cup(\mathcal{T}\times\mathcal{P})$, and the *initial marking* In as multiset over $\mathcal{P}$. For a place p, the *precondition* is the set $pre(p) = \{t \in \mathcal{T} \mid \mathcal{F}(t,p) > 0\}$ and the *postcondition* is the set $post(p) = \{t \in \mathcal{T} \mid \mathcal{F}(p,t) > 0\}$. For a transition t, the *precondition* is the multiset over $\mathcal{P}$ defined by $pre(t)(p) = \mathcal{F}(p,t)$ for all $p \in \mathcal{P}$ and the *postcondition* is the multiset over $\mathcal{P}$ defined by $post(t)(p) = \mathcal{F}(t,p)$ for all $p \in \mathcal{P}$. States of Petri nets are represented by multisets over $\mathcal{P}$, called *markings*. A marking M puts $M(p)$ *tokens* in every place $p \in \mathcal{P}$. A transition t is *enabled* in a marking M if $pre(t) \subseteq M$. If no transition is enabled in a marking M, then M is called *final*. An enabled transition t can *fire* in a marking M resulting in the successor marking $M' = M - pre(t) + post(t)$ (written $M[t\rangle M'$). For markings M and M', we write $M[t_0, \ldots, t_{n-1}\rangle M'$ if there exist markings $M_0, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $M_i[t_i\rangle M_{i+1}$ for $0 \leq i < n$. The set of *reachable markings* of $\mathcal{N}$ is defined as $\mathcal{R}(\mathcal{N}) = \{M \mid \exists n \in \mathbb{N}, t_0, \ldots, t_{n-1} \in \mathcal{T} : In[t_0, \ldots, t_{n-1}\rangle M\}$. A net $\mathcal{N}'$ is a *subnet* of $\mathcal{N}$ (written $\mathcal{N}' \sqsubseteq \mathcal{N}$) if $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{T}' \subseteq \mathcal{T}$, $In' = In$, and $\mathcal{F}' = \mathcal{F} \restriction (\mathcal{P}'\times\mathcal{T}')\cup(\mathcal{T}'\times\mathcal{P}')$. We enforce $In' = In$ to maintain all players when later defining strategies for Petri games.

We call elements in $\mathcal{P} \cup \mathcal{T}$ *nodes*. For nodes x and y, we write $x \lessdot y$ if $x \in pre(y)$. With $\leq$, we denote the reflexive, transitive closure of $\lessdot$. The *causal past* of x is $past(x) = \{y \mid y \leq x\}$. Nodes x and y are *causally related* if $x \leq y \vee y \leq x$. They are *in conflict* (written $x \sharp y$) if, for a place $p \in \mathcal{P}$, there are distinct transitions $t_1, t_2 \in post(p)$ with $t_1 \leq x \wedge t_2 \leq y$. Node x is in *self-conflict* if $x \sharp x$. We call x and y *concurrent* if they are neither causally related nor in conflict.

An *occurrence net* is a Petri net where the pre- and postcondition of transitions are sets, the initial marking coincides with places without ingoing transitions, other places have exactly one ingoing transition, no infinite path starting from any given node and following the inverse flow relation exists, and no transition is in self-conflict. A *homomorphism* maps nodes from $\mathcal{N}_1$ to $\mathcal{N}_2$ preserving the type of nodes and the pre- and postcondition of transitions.

A branching process [5, 28, 6] describes parts of possible behaviors of a Petri net. We use the *individual token semantics* [20]. A *branching process* of a Petri net $\mathcal{N}$ is a pair $\iota = (\mathcal{N}^\iota, \lambda^\iota)$ where $\mathcal{N}^\iota$ is an occurrence net and $\lambda^\iota : \mathcal{P}^\iota \cup \mathcal{T}^\iota \to \mathcal{P} \cup \mathcal{T}$ is a homomorphism from $\mathcal{N}^\iota$ to $\mathcal{N}$ that is injective on transitions with the same precondition. Intuitively, whenever a node can be reached on two distinct paths in a Petri net $\mathcal{N}$, it is split up in the branching process of $\mathcal{N}$. λ^ι labels the nodes of $\mathcal{N}^\iota$ with the original nodes of $\mathcal{N}$. The injectivity condition avoids additional unnecessary splits. The *unfolding* $\iota_U = (\mathcal{N}^U, \lambda^U)$ of $\mathcal{N}$ is a maximal branching process: Whenever there is a set of pairwise concurrent places C such that $\lambda^U[C] = pre^{\mathcal{N}}(t)$ for some transition $t \in \mathcal{T}$, then there exists $t' \in \mathcal{T}^U$ with $\lambda^U(t') = t$ and $pre^U(t') = C$.

4 Petri Games and Büchi Games

A *Petri game* [15, 14] is a tuple $\mathcal{G} = (\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{W})$. The places of the underlying Petri net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ are partitioned into *system places* $\mathcal{P}_S$ and *environment places* $\mathcal{P}_E$. We call tokens on system places *system players* and tokens on environment places *environment players*. The game is played by firing transitions in $\mathcal{N}$. Players *synchronize* when a joint transition fires. Intuitively, a strategy controls the behavior of system players by deciding which transitions to allow. Environment players are uncontrollable and transitions only dependent on environment players cannot be restricted. The *winning condition* is given by $\mathcal{W}$ as the set of *bad places* $\mathcal{P}_B \subseteq \mathcal{P}$, *bad markings* $\mathcal{M}_B \subseteq \mathcal{R}(\mathcal{N})$, or *good markings* $\mathcal{M}_G \subseteq \mathcal{R}(\mathcal{N})$ or the pair of disjoint sets of *good and bad markings* $(\mathcal{M}_G, \mathcal{M}_B) \in \mathbb{P}(\mathcal{R}(\mathcal{N})) \times \mathbb{P}(\mathcal{R}(\mathcal{N}))$. We depict Petri games as Petri nets and color system places gray and environment places white. A Petri game has *a bounded number of system players and one environment player* if, for a bound $k \in \mathbb{N}$, every system place contains at most k tokens for all reachable markings of $\mathcal{N}$ and the sum of tokens in all environment places is exactly one for all reachable markings of $\mathcal{N}$.

A *strategy* for $\mathcal{G}$ is a branching process $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ of $\mathcal{N}$ satisfying *justified refusal*: If there is a set of pairwise concurrent places C in $\mathcal{N}^\sigma$ and a transition $t \in \mathcal{T}$ with $\lambda^\sigma[C] = pre^{\mathcal{N}}(t)$, then there either is a transition $t' \in \mathcal{T}^\sigma$ with $\lambda^\sigma(t') = t$ and $C = pre^\sigma(t')$ or there is a system place $p \in C \cap (\lambda^\sigma)^{-1}[\mathcal{P}_S]$ with $t \notin \lambda^\sigma[post^\sigma(p)]$. Justified refusal enforces that only system places can prohibit transitions based on their causal past: From every situation in the game, a transition possible in the underlying net is either in the strategy or there is a system place that never allows it. A strategy is a restriction of possible transitions in the Petri game because it is a branching process which describes subsets of the behavior of a Petri net. We further require σ to be *deterministic*: For every reachable marking M of σ and system place $p \in M$, there is at most one transition from $post^\sigma(p)$ enabled in M. Notice that $post^\sigma(p)$ can contain more than one transition as long as at most one of them is enabled in the same reachable marking. This allows that the environment player decides between different branches of the Petri game and the system player later on reacts to every decision.

A strategy is *deadlock-avoiding* if, for every final, reachable marking M in the strategy, $\lambda^\sigma[M]$ is final as well. A strategy σ is *winning for bad places* $\mathcal{W} = \mathcal{P}_B$ if it is deadlock-avoiding and no reachable marking in σ contains a place corresponding to a bad place. A strategy σ is *winning for bad markings* $\mathcal{W} = \mathcal{M}_B$ if it is deadlock-avoiding and no reachable marking in σ corresponds to a bad marking. One branching process $\iota^1 = (\mathcal{N}^1, \lambda^1)$ is a *subprocess* of another $\iota^2 = (\mathcal{N}^2, \lambda^2)$ if $\mathcal{N}^1 \sqsubseteq \mathcal{N}^2$ and $\lambda^1 = \lambda^2 \restriction (\mathcal{P}^1 \cup \mathcal{T}^1)$. A *play* $\pi = (\mathcal{N}^\pi, \lambda^\pi)$ is a subprocess of a strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ with $\forall p \in \mathcal{P}^\pi : |post(p)| \leq 1$. It is *maximal* if, for each set of pairwise concurrent places C in $\mathcal{N}^\pi$ with $C = pre^\sigma(t)$ for some $t \in \mathcal{T}^\sigma$, a place $p \in C$ and a transition $t' \in \mathcal{T}^\pi$ exist with $t' \in post^\pi(p)$. A *complete firing sequence* of a play π is a possibly infinite sequence of fired transitions such that each transition of π occurs. A strategy σ is *winning for good markings* $\mathcal{W} = \mathcal{M}_G$ if, for all complete firing sequences $t_0 t_1 t_2 \ldots$ of all maximal plays π of σ with $M_0 = In^\pi$ and $M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \ldots$, there exists $i \geq 0$ with $\lambda^\pi[M_i] \in \mathcal{M}_G$. A strategy σ is *winning for good and bad markings* $\mathcal{W} = (\mathcal{M}_G, \mathcal{M}_B)$ if, for all complete firing sequences $t_0 t_1 t_2 \ldots$ of all maximal plays π of σ with $M_0 = In^\pi \wedge M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \ldots$, there exists $i \geq 0$ with $\lambda^\pi[M_i] \in \mathcal{M}_G \wedge \forall 0 \leq j < i : \lambda^\pi[M_j] \notin \mathcal{M}_B$. Terminating in a final marking as winning condition is different from reaching a good marking as players are not required to terminate in a good marking and can reach a bad marking afterward.

A *Büchi game* has two players: *Player 0* represents the *system*, *Player 1* the *environment*. Both act on complete information about the game arena and the play so far. To win, Player 0 has to ensure that an accepting state is visited infinitely often. A winning strategy for Player 0 corresponds to a correct implementation of the encoded synthesis problem. Deciding the existence of a winning strategy can be done in polynomial time [3].

Formally, a *Büchi game* $\mathbb{G} = (V, V_0, V_1, I, E, F)$ consists of the finite set of *states* V partitioned into the disjoint sets of states V_0 of Player 0 and of states V_1 of Player 1, the *initial state* $I \in V$, the *edge relation* $E \subseteq V \times V$, and the set of *accepting states* $F \subseteq V$. We assume that all states in a Büchi game have at least one outgoing edge. A *play* is a possibly infinite sequence of states which is constructed by letting Player 0 choose the next state from the successors in E whenever the game is in a state from V_0 and by letting Player 1 choose otherwise. An *initial play* is a play that starts from the initial state. A play is *winning* for Player 0 if it visits at least one accepting state infinitely often. Otherwise, the play is *winning* for Player 1. A *strategy* for Player 0 is a function $f : V^* \cdot V_0 \to V$ that maps plays ending in states of Player 0 to one possible successor according to E. A play *conforms* to a strategy f if all successors of states in V_0 are chosen in accordance with f. A strategy f is winning for Player 0 if all initial plays that conform to f are winning for Player 0.

5 Decidability in Petri Games with Bad Markings

We present a reduction from Petri games with a bounded number of system players, one environment player, and bad markings to Büchi games. In the following, we give an intuition for the main concepts of the reduction, before presenting the structure of the Büchi game in the remainder of this section. More details are in [12] and a running example is in Fig. 2.

Petri games use unfoldings, which can be of infinite size, to encode the causal memory of players. By contrast, Büchi games have two players with complete information and a finite number of states. To overcome these differences when encoding Petri games, states in the corresponding Büchi games consist of a representation of the current marking and some additional information. Edges in the Büchi game mostly correspond to a transition firing in the Petri game. We say that a transition fires in the Büchi game when it fires in the encoded Petri game. Concurrency between transitions in the Petri game is encoded by having

most possible interleavings in the Büchi game. Some interleavings are left out to encode causal memory of the players in Petri games: Causal memory is simulated in Büchi games by transitions with an environment place in their precondition firing as late as possible at *mcuts* [15]. An mcut is a situation in the Petri game where all system players have progressed maximally, i.e., the environment player can choose between all remaining possible transitions. Mcuts can only be defined for Petri games with at most one environment player. Player 1 in the Büchi game makes decisions only at states corresponding to mcuts.

We make two key additions to ensure that the idea of firing transition with the environment player only at mcuts can be lifted from local winning conditions to global winning conditions: First, we add *backward moves* to detect bad markings and nondeterministic decisions. Intuitively, backward moves allow us to rewind transitions with only system players participating. They are realized by each system player remembering its history until its last synchronization with the environment player. In every state of the Büchi game, it is checked whether the backward moves of all system players allow us to rewind the game in such a way that a bad marking is reached or a nondeterministic decision is found.

Second, we add the so-called *NES-case* to handle system players playing infinitely without synchronizing with the environment player directly in the Büchi game. The abbreviation NES stands for *no more environment synchronization* and is necessary when some system players play infinitely but without synchronization with the environment player. In [15], this situation is called the type-2 case and can be handled as a preprocessing step, because only the local winning condition of bad places is considered. This is impossible for the global winning condition of bad markings considered in this paper. Throughout this paper, the NES-case can be disregarded by adding the restriction that each system player either terminates or synchronizes infinitely often with the environment player.

For the NES-case, every system player has a three-valued flag. As long as the system player will terminate or will synchronize with the environment player in the future, the flag should be set to *negative NES-status*. When system players can play infinitely without synchronizing with the environment player, they should set their flags to *positive NES-status*. After the NES-case, participating system players obtain an *ended NES-status*, which excludes them from the remaining Büchi game. A positive NES-status triggers the NES-case. Here, the system players with positive NES-status have to prove that they can play infinitely without synchronizing with the environment player. Therefore, the usual order of all transitions without the environment player being possible until reaching an mcut is interrupted. Instead, only system players with positive NES-status are considered until their proof of playing infinitely without the environment player is successful. If the system players with positive NES-status make a mistake in their proof, then Player 0 immediately loses the Büchi game.

5.1 States and Initial State in the Büchi Game

Decision tuples represent players of the Petri game in states in the Büchi game. A *decision tuple* for a player consists of an *identifier*, a *position*, a *NES-status*, a *decision*, and a *representation of the last mcut*. The identifier uniquely determines the player. The position gives the current place of the player. System players with negative NES-status *false* claim that they will terminate or fire a transition with an environment place in its precondition and are not part of the NES-case. In the NES-case, system players go from positive NES-status *true* to ended NES-status *end* as described previously.

The decision is either $\top$ or the set of *allowed transitions* by the player. For system players, $\top$ indicates that a decision for a set of allowed transitions is missing and has to be chosen. The representation of the last mcut encodes the last known position of the

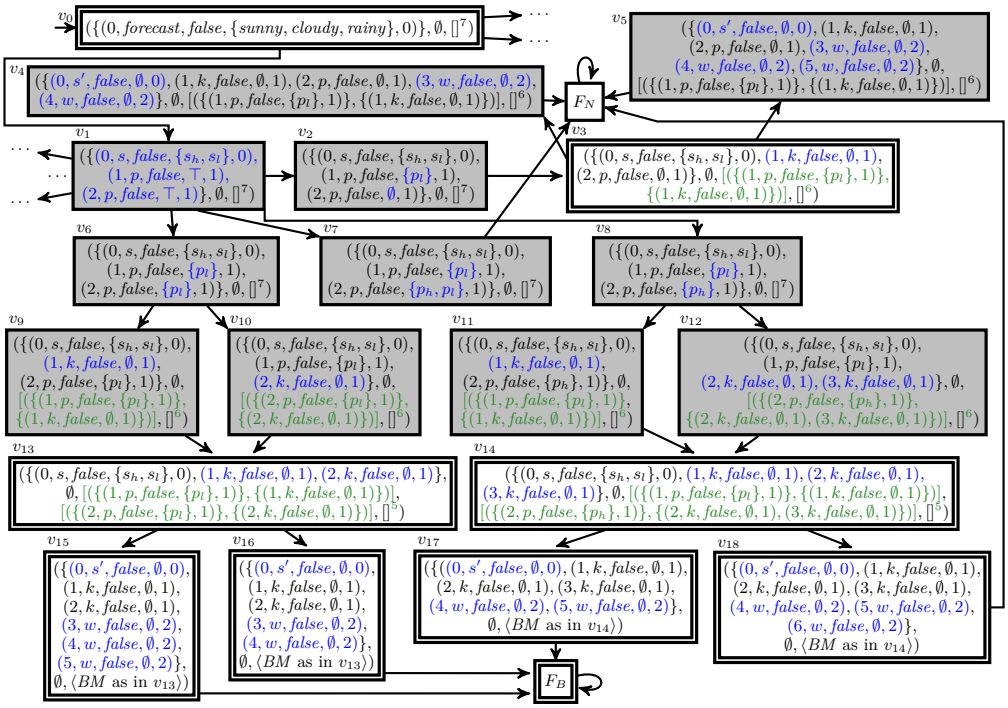
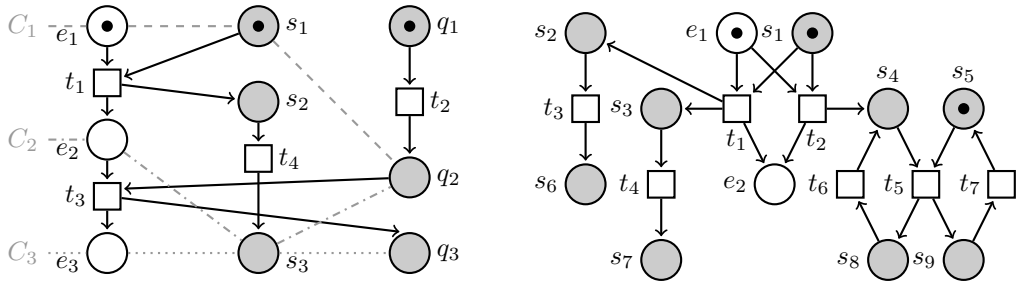


Figure 2 Part of the Büchi game for the Petri game in Fig. 1 is given. States of Player 0 are gray, states of Player 1 white. Most states are labeled for identification. Double squares are accepting states. Changes from previous states are blue for decision tuples and green for backward moves.

environment player. There can be at most as many different such positions as there are system players. Thus, a number suffices to identify the last known mcut. Let $\max_S$ be the maximal number of system players in the Petri game which are visible at the same time. The set of *system decision tuples* is $\mathcal{D}_S = \{(id, p, b, T, K) \mid id, K \in \{1, \ldots, \max_S\} \wedge p \in \mathcal{P}_S \wedge b \in \{false, true, end\} \wedge (T = \top \vee T \subseteq post(p))\}$, the set of *environment decision tuples* is $\mathcal{D}_E = \{(0, p, false, post(p), 0) \mid p \in \mathcal{P}_E\}$, and the set of all *decision tuples* is $\mathcal{D} = \mathcal{D}_S \cup \mathcal{D}_E$.

▶ **Example 1.** In Fig. 2, a branch of the Büchi game for the Petri game in Fig. 1 is shown. States with decision tuples with positive NES-status are omitted because no infinite behavior occurs. The initial state v_0 has one decision tuple for the environment player in place *forecast* and empty information for the NES-case and the backward moves. After Player 1 plays the edge for transition *sunny* firing, state v_1 with three decision tuples is reached. The decision tuples for the two system players in place p have $\top$ as decision. There are 16 combinations of decisions by the two system players, out of which four are shown. The first system player always allows transition p_l and the second system player allows no transition in v_2, only one of the two transitions p_l and p_h in v_6 and v_8, or both transitions in v_7.

Almost all states in the Büchi game contain decision tuples and additional information for the NES-case and for backward moves. The *states in the Büchi game* are defined as $V = V_{BN} \cup \mathscr{D} \times (\mathcal{P}_S \to \{0, \ldots, k\}) \times (\mathcal{B}^*)^{\max_S}$ with $V_{BN} = \{F_B, F_N\}$. Finite winning and losing behavior in the Petri game is represented in the Büchi game by the two unique states F_B and F_N in V_{BN}. A *decision marking* is a set of decision tuples corresponding to a reachable marking in the Petri game such that each identifier occurs at most once. $\mathscr{D}$ is the set of all such decision markings. The next element stores the underlying multiset over



(a) The mcuts of the unfolding are C_1, C_2, and C_3. **(b)** Markings containing s_2 and s_7 are bad markings.

Figure 3 Two Petri games illustrate mcuts, backward moves, and the NES-case.

system places of the decision marking from the start of the NES-case restricted to system players with positive NES-status. In the NES-case, repeating this multiset proves that the system players with positive NES-status can play infinitely without firing a transition with an environment place in its precondition. This element is the empty multiset if not in the NES-case. More details are in Sec. 5.5. $\mathcal{B} : \mathbb{P}(\mathcal{D}_S) \times \mathbb{P}(\mathcal{D}_S)$ is the set of backward moves to detect states corresponding to a bad marking or a nondeterministic decision. The remaining elements are $\max_S$ sequences of backward moves. Each identifier in a decision tuple maps to the position of a sequence of backward moves. More details are in Sec. 5.4.

The *initial state in the Büchi game* has as many decision tuples with unique identifier, NES-status *false*, $\top$ as decision, and last mcut 1 as there are tokens in system places in *In* of the Petri game and one decision tuple with identifier 0, NES-status *false*, the postcondition of p_E as decision, and last mcut 0 for the one environment place p_E with one token in *In*. The other parts are the empty multiset or the empty sequence of backward moves.

5.2 States of Player 0, States of Player 1, and Accepting States

Causal memory in Petri games is encoded in Büchi games by letting Player 0 fix the decisions of allowed transitions for system players as early as possible and having Player 1 fire transitions with an environment place in their precondition as late as possible at mcuts. *Cuts* are markings in unfoldings. An *mcut* is a cut where all enabled transitions have an environment place in their precondition, i.e., all system players progressed maximally on their own. With Fig. 3a, we illustrate mcuts. The initial cut $\{e_1, s_1, q_1\}$ is *not* an mcut as the enabled transition t_2 has only the system place q_1 in its precondition. After t_2 fires, the cut $C_1 = \{e_1, s_1, q_2\}$ is an mcut as the only enabled transition t_1 has environment place e_1 in its precondition. Analog arguments lead to $\{e_2, s_2, q_2\}$ not being an mcut and $C_2 = \{e_2, s_3, q_2\}$ being an mcut. The final cut $C_3 = \{e_3, s_3, q_3\}$ is an mcut as there are no enabled transitions.

A decision marking $\mathbb{D}$ in the states in the Büchi game *corresponds to an mcut* when no $\top$ and no positive NES-status are part of $\mathbb{D}$ and every transition with only system places in its precondition is not enabled or not allowed by a participating system player in $\mathbb{D}$. A state in the Büchi game can correspond to an mcut although the cut in the unfolding of the Petri game is not an mcut as the decisions of the system players in the Büchi game can disallow transitions. *States of Player 1* are F_B, F_N, and states corresponding to an mcut. *States of Player 0* are all other states. *Accepting states* are F_B and states corresponding to an mcut.

▶ **Example 2.** The Petri game from Fig. 1 has $\{forecast\}$, $\{\{e, k : i\} \mid e \in \{s, c, r\} \wedge 2 \leq i \leq 4\}$, and $\{\{s', w : w_{s'}, k : i\}, \{c', w : w_{c'}, k : i\}, \{r', w : w_{r'}, k : i\} \mid 2 \leq w_{s'} \leq 3 \wedge 1 \leq w_{c'} \leq 2 \wedge 0 \leq w_{r'} \leq 1 \wedge 2 \leq i \leq 4\}$ as mcuts, i.e., the initial cut, cuts where the power plants produced

energy while the energy production by renewable sources was not selected, and all final, reachable cuts. In the Büchi game in Fig. 2, the eight states v_0, v_3, and v_{13} to v_{18} of Player 1 have decision markings that correspond to an mcut. For states v_0, v_{13}, and v_{14}, all enabled transitions have an environment place in their precondition. For states v_{15} to v_{18}, each decision marking corresponds to a final cut. The decision marking of state v_3 corresponds to an mcut as the second system player in p decided to not allow any of its outgoing transitions.

5.3 Edges in the Büchi Game

Edges in the Büchi game mostly connect states $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ and $\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$ where $\mathbb{D}$ is a decision marking, M_{T2} is a marking, and $BM_1, \ldots, BM_{\max_S}$ are as many sequences of backward moves as the maximum number $\max_S$ of system players in the Petri game. There are five sets of edges *TOP*, *SYS*, *NES*, *MCUT*, and *STOP*. In the following description of the five sets of edges, not mentioned elements of the connected states stay the same. The formal definitions can be found in [12].

(1) Edges from *TOP* occur from states where at least one decision tuple in $\mathbb{D}$ has $\top$ as decision. To obtain $\mathbb{D}'$, Player 0 replaces each $\top$ in the decision tuples of system players with a set of allowed transitions and can change the NES-status of decision tuples for system players from *false* to *true*. The underlying marking of decision tuples with positive NES-status *true* is stored in M'_{T2} when a NES-status changes.

(2) Edges from *SYS* occur from states where all decision tuples in $\mathbb{D}$ have negative NES-status and at least one transition with only system places in its precondition is enabled and allowed by the decision tuples in $\mathbb{D}$. To get $\mathbb{D}'$, Player 0 simulates one such transition t firing by removing decision tuples $\mathbb{D}_{pre}$ for the precondition of t and adding decision tuples $\mathbb{D}_{post}$ for the postcondition of t. For $\mathbb{D}_{post}$, the last mcut of all participating players is the maximum of their previous values and Player 0 picks the decisions and can change the NES-status as in (1). Marking M'_{T2} is obtained as in (1). Backward move $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ is added to BM_{id} of all participating players with identifier *id* to get BM'_{id}.

(3) Edges from *NES* are the NES-case and occur from states where a decision tuple in $\mathbb{D}$ has positive NES-status. To obtain $\mathbb{D}'$, Player 0 fires a transition as in (2) but only from decision tuples with positive NES-status resulting in new decision tuples with positive NES-status. This includes the storage of backward moves. The NES-case is successful if the marking M_{T2} is reached again and all players in it moved. Then, decision tuples with NES-status *true* are set to NES-status *end* and M'_{T2} becomes the empty marking.

(4) Edges from *MCUT* occur from states where all enabled and allowed transitions have an environment place in their precondition. To get $\mathbb{D}'$, Player 1 fires one such transition. Decision tuples for the precondition of the transition are removed, decision tuples for the postcondition are added. Added decision tuples for system players have NES-status *false*, $\top$ as decision, an empty sequence of backward moves, and the highest last mcut. As backward moves store the past of system players until their last mcut, backward moves for system players that are part of the transition are removed. If backward moves become never applicable by firing the transition, they are removed from the successor state.

(5) Edges from *STOP* occur from states with no transition enabled or corresponding to losing behavior. They replace other outgoing edges for losing behavior. States corresponding to termination lead to the winning state F_B. States corresponding to a deadlock but not termination lead to the losing state F_N. If backward moves detect a bad marking or a nondeterministic decision, the state leads to F_N. In the NES-case, a synchronization of decision tuples with positive and negative NES-status or a deadlock or vanishing of decision tuples with positive NES-status leads to F_N. Decision tuples with positive NES-status can vanish when transitions with empty postcondition fire.

▶ **Example 3.** In Fig. 2, outgoing edges of state v_1 are in *TOP*. Outgoing edges of states v_2, v_6, and v_8 to v_{12} are in *SYS*. Outgoing edges of states v_0, v_3, v_{13}, and v_{14} are in *MCUT*. Other edges are in *STOP*. No edges in *NES* exist in the depicted part. Outgoing edges of states v_4 and v_5 represent the deadlock of the second system player in p disallowing both outgoing transitions while only they are enabled. The outgoing edge of state v_7 encodes a nondeterministic decision of the second system player, which allows two enabled transitions. Such a decision is only useful if another player ensures that at most one of the transitions becomes enabled. Outgoing edges of states v_{15} to v_{17} represent termination. The outgoing edge of state v_{18} represents a bad marking for six produced units of energy.

When, as in our construction, (I) Player 0 immediately resolves $\top$ to the decisions of system players, (II) Player 0 decides which transitions with only system places in their precondition fire following the decisions of system players, and (III) Player 1 decides as late as possible at mcuts which transitions with an environment place in their precondition fire following the decisions of system players, then the corresponding Büchi games encode causal memory [15]. Allowed transitions with only system places in their precondition fire in an order determined by Player 0 until an mcut is reached. There, Player 1 decides for the environment player which allowed transition to fire. Afterward, this process repeats itself.

5.4 Backward Moves in the Büchi Game

In the Büchi game, Player 0 can avoid markings by picking the firing order for transitions with only system places in their precondition. In Fig. 3b, the two system players in s_2 and s_3 are reached after t_1 fires. One can fire t_3, the other t_4. This results in the firing sequences $t_1t_3t_4$ and $t_1t_4t_3$. If s_2 and s_7 are in a bad marking, then Player 0 can decide for edges corresponding to the first firing sequence and the bad marking is missed. We introduce backward moves to avoid such problems. A *backward move* is a pair of decision markings. It stores the change to the decision tuples by edges from *SYS* and *NES*. For every such edge from $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ to $\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$, we obtain $\mathbb{D}_{pre}$ and $\mathbb{D}_{post}$ with $\mathbb{D}' = (\mathbb{D} \setminus \mathbb{D}_{pre}) \cup \mathbb{D}_{post}$ and add backward move $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ to the end of BM_{id} of all participating players with identifier id.

For every state $\mathcal{V}'$ in the Büchi game, it is checked with backward moves if $\mathcal{V}'$ is losing due to a bad marking or a nondeterministic decision. The decision marking $\mathbb{D}'$ and all decision markings that are reachable via backward moves are checked. Therefore, it is checked whether backward moves $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ are *applicable* to $\mathbb{D}'$, i.e., whether $\mathbb{D}_{post} \subseteq \mathbb{D}'$ and $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ is the last backward move of all participating players. In this case, the backward move is removed from the end of the sequences of backward moves of all participating players and $\mathbb{D} = (\mathbb{D}' \setminus \mathbb{D}_{post}) \cup \mathbb{D}_{pre}$ results from the application of the backward move. The underlying marking of $\mathbb{D}$ is checked to not be a bad marking and $\mathbb{D}$ is checked to have only deterministic decisions. This is repeated recursively from $\mathbb{D}$ for all applicable backward moves until no backward move is applicable. If a decision marking corresponding to a bad marking or a nondeterministic decision is detected, the current state $\mathcal{V}'$ only has an edge to F_N.

The identifier of players in decision tuples is used to map the decision tuple to the corresponding sequence of backward moves, i.e., for each system player in the Petri game, the Büchi game collects a sequence of backward moves. Edges from *MCUT* empty the sequence of backward moves of decision tuples when their system place is in the precondition of the fired transition. This removal can make backward moves not applicable because some participating players do not have the backward move as their last one anymore.

The sequence of backward moves can grow infinitely long when system players play infinitely without the environment player and without the NES-case. This would result in a Büchi game with infinitely many states. To avoid this, the Büchi game becomes losing for Player 0 when it plays in a way that corresponds to a strategy with a variant of useless repetitions [19] for the system players in the Petri game. Our variant of useless repetitions identifies the repetition of a loop consisting only of transitions without the environment player in their precondition such that the last mcut of the system players does not change, i.e., the system players repeat a loop in which they do not exchange any new information about the environment player. Thus, winning strategies have to avoid playing a useless repetition more than once between the successor of an mcut and the next mcut. This can be achieved either by continuing to the next mcut or by setting some players to NES-status *true* and completing the NES-case, i.e., playing infinitely without the environment player.

▶ **Example 4.** In Fig. 2, we include the collection of backward moves. State v_{13} represents each power plant producing one unit of energy after a sunny weather forecast. It is reached from state v_6 either via state v_9 or v_{10} depending on which power plant produces energy first. State v_{13} has a backward move for each power plant: $(\{(1, p, \mathit{false}, \{p_l\}, 1)\}, \{(1, k, \mathit{false}, \emptyset, 1)\})$ and $(\{(2, p, \mathit{false}, \{p_l\}, 1)\}, \{(2, k, \mathit{false}, \emptyset, 1)\})$. Because the three markings $\{s, k : 2\}$ (underlying marking of v_{13}), $\{s, p, k\}$ (applying one backward move), and $\{s, p : 2\}$ (applying both backward moves) are no bad markings and all decisions are deterministic, state v_{13} continues with edges for the transitions of the environment place s instead of having an edge to F_N.

5.5 Encoding the NES-Case Directly in the Büchi Game

We handle the *NES-case* where system players play infinitely without firing a transition with an environment place in its precondition directly in the Büchi game as players in the NES-case might be in a bad marking. This is in contrast to the reduction for bad places [15].

In the Büchi game, Player 0 has to reach an accepting state infinitely often in order to win the game. Only F_B and states corresponding to an mcut are accepting states. Transitions with only system places in their precondition are fired between successors of mcuts and the following mcut. Thus, if the system players can fire transitions with only system places in their precondition infinitely often, eventually a useless repetition is reached which is losing. To overcome this, we give Player 0 the possibility to change the NES-status for decision tuples of system players from negative to positive. The underlying marking of this change is stored and afterward only transitions from decision tuples with positive NES-status can be fired. Firing these transitions maintains the positive NES-status for new decision tuples. Instead of firing infinitely many transitions, the NES-case is ended if the stored marking is reached again and all players in the marking have moved. In this case, the NES-status of all decision tuples with positive NES-status is changed to ended NES-status and the Büchi game continues with the remaining decision tuples with negative NES-status. The requirement to move is necessary as otherwise too many players could get ended NES-status. Decision tuples with ended NES-status are maintained as backward moves can be applicable to them, i.e., backward moves store the NES-status and allow us to reverse it in search for a bad marking. We can thus ensure that continuing with the case where all decision tuples have negative NES-status avoids bad markings that span the NES-case.

Player 0 has to disclose decision tuples with positive NES-status if system players fire infinitely many transitions with only system places in their precondition. Otherwise, they lose the game as no accepting state is reached infinitely often. It is losing if system players with positive and negative NES-status synchronize, if players with positive NES-status deadlock,

if one such player is not moved and the marking from the start of the NES-case is repeated, if all such players vanish, or if another marking is repeated. Notice that at most one NES-case is necessary per branch in the strategy tree of the Büchi game. For a safety winning condition, possible NES-cases after the first successful one can simply terminate.

▶ **Example 5.** A Petri game with necessary NES-case in the encoding Büchi game is shown in Fig. 3b. After Player 0 allows transition t_2 and Player 1 fires it, a state is reached where the decision tuples for s_4 and s_5 can be set to positive NES-status by Player 0. After transitions t_5, t_6, and t_7 fire, the marking $\{s_4, s_5\}$ is repeated and the NES-case is successful, proving that t_5, t_6, and t_7 can fire infinitely often.

5.6 Decidability Result

We analyze the properties of the constructed Büchi game. Detailed proofs are in [12].

▶ **Lemma 6** (From Büchi game to Petri game strategies)**.** *If Player 0 has a winning strategy in the Büchi game, then there is a winning strategy for the system players in the Petri game.*

Proof Sketch. From the tree T_f representing the winning strategy f for Player 0 in the Büchi game, we inductively build a winning strategy σ for the system players in the Petri game. Each cut in σ is associated with a node in T_f, transitions are added following the edges in T_f, and the associated cut is updated if needed. This strategy σ for the system players in the Petri game is winning as it visits equivalent cuts to the reachable states in f. ◀

▶ **Lemma 7** (From Petri game to Büchi game strategies)**.** *If the system players have a winning strategy in the Petri game, then there is a winning strategy for Player 0 in the Büchi game.*

Proof Sketch. We skip unnecessary NES-cases and useless repetitions in the winning strategy σ for the system players in the Petri game. We replace $\top$ based on the post-condition of system places, disclose necessary NES-cases, fire enabled transitions with only system places in their precondition in an arbitrary but fixed order between states after an mcut and the next mcut, and add all options at mcuts. This strategy for Player 0 in the Büchi game is winning as it visits equivalent states to the reachable cuts in σ. ◀

▶ **Theorem 8** (Game solving)**.** *For Petri games with a bounded number of system players, one environment player, and bad markings, the question of whether the system players have a winning strategy is decidable in* 2-EXPTIME*. If a winning strategy for the system players exists, it can be constructed in exponential time.*

Proof Sketch. The complexity is based on the double exponential number of states in the Büchi game and polynomial solving of Büchi games. There are exponentially many states in the size of the Petri game to represent decision tuples and each of these states has to store sequences of backward moves of at most exponential length in the size of the Petri game. This transfers to the size of the winning strategy because it can be represented finitely. ◀

▶ Remark 9. In the presented construction, Player 0 in the Büchi game decides both the decisions of the system players in the Petri game and the order in which concurrent transitions with only system places in their precondition are fired. The question might arise whether it is possible that Player 1 representing the environment determines the order in which concurrent transitions with only system places in their precondition are fired. This is not possible because the system players can make different decisions depending on the order of transitions decided by the environment player. We present a detailed counterexample where this change would result in a different winner of a Petri game in the full version [12].

6 Undecidability in Petri Games with Good Markings

We prove that it is undecidable if a winning strategy exists for the system players in Petri games with at least two system and one environment player and good and bad markings by enforcing an undecidable synchronous setting in Petri games. For this winning condition, no bad marking should be reached *until* a good marking is reached, which can be expressed in LTL. Notice also that, after a good marking has been reached, it is allowed to reach a bad marking. Afterward, we prove that it is undecidable if a winning strategy exists for the system players in Petri games with only good markings and at least three players, out of which one is an environment player and each of the other two changes between system and environment player. Bad markings from the previous result are encoded by system players repeatedly changing to environment players and back. More details can be found in [12]. The underlying main idea of the first construction is also used in other settings [26, 34, 36].

6.1 Petri Game for the Post Correspondence Problem

We recall that a strategy σ is winning for good and bad markings $\mathcal{W} = (\mathcal{M}_G, \mathcal{M}_B)$ if, for all complete firing sequences $t_0 t_1 t_2 \dots$ of all maximal plays π of σ with $M_0 = In^\pi \wedge M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \dots$, there exists $i \geq 0$ with $\lambda^\pi[M_i] \in \mathcal{M}_G \wedge \forall 0 \leq j < i : \lambda^\pi[M_j] \notin \mathcal{M}_B$. The undecidability proof uses the Post correspondence problem [35]. The *Post correspondence problem* (PCP) is to determine, for a finite alphabet Σ and two finite lists $r_0, r_1, \dots, r_n$ and $v_0, v_1, \dots, v_n$ of non-empty words over Σ, if there exists a non-empty sequence $i_1, i_2, \dots, i_l \in \{0, 1, \dots, n\}$ such that $r_{i_1} r_{i_2} \dots r_{i_l} = v_{i_1} v_{i_2} \dots v_{i_l}$. This problem is undecidable.

To simulate the PCP in a Petri game, we use one environment and two system players. The three players are *independent* as they cannot communicate with each other. Each system player outputs a solution to the PCP. By firing a transition, a player *outputs the label* of the transition. The output of the first system player is $i_1 r_{i_1} \tau i_2 r_{i_2} \tau \dots i_l r_{i_l} \tau \#_1$ and the output of the second one is $j_1 v_{j_1} \tau j_2 v_{j_2} \tau \dots j_m v_{j_m} \tau \#_2$ for $i_1, \dots, i_l, j_1, \dots, j_m \in \{0, 1, \dots n\}$. Both system players output *indices* followed by the word from the index position of the respective list and τ, and end symbol $\#_1$ or $\#_2$ at the end of the sequence. Words r_i for $i \in \{i_1, \dots, i_l\}$ and v_j for $j \in \{j_1, \dots, j_m\}$ are output letter-by-letter. A correct solution to the PCP fulfills $l > 0$, $m > 0$, $l = m$, $i_1 = j_1$, $i_2 = j_2$, $\dots$, $i_l = j_m$, and $r_{i_1} r_{i_2} \dots r_{i_l} = v_{j_1} v_{j_2} \dots v_{j_m}$.

We ensure that strategies for the two system players can only win by outputting the same sequence of indices at both players. This permits to decide for these strategies if a good marking is reached where both system players have output a correct solution. Using the independence of the three players and depending on a choice by the environment player, we either check the equality of the output sequences of indices or of the letter-by-letter output sequences of words. Therefore, the strategy for the system players has to behave as if both is tested. With good markings, we restrict the asynchronous setting of Petri games to turn-taking firing sequences on the output indices or letters. Thus, we consider equivalent firing sequences to the synchronous setting and can check the conditions for a correct solution to the PCP after both system players have output the end symbol. With bad markings, we identify when output indices or output letters do not match. System players can only output the end symbol after outputting at least one index and word to ensure non-empty solutions.

6.2 Linear Firing Sequences via Good Markings

We use MOD-3 counters to restrict the asynchronous setting of Petri games to firing sequences equivalent to the synchronous setting of Pnueli and Rosner [34, 38]. The main idea is that we are just interested in runs where the first system player is only zero or one step ahead of

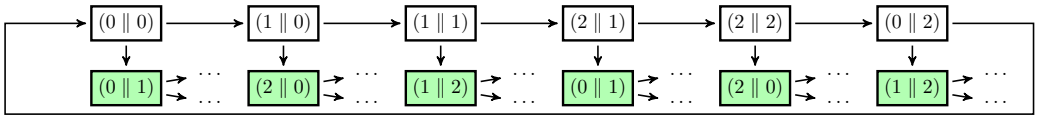


Figure 4 The reachability graph for the two system players is depicted when only considering either the values of their MOD-3 index counters or of their MOD-3 letter counters and differentiating markings depending on if a good marking is reached before. Good markings are colored green. All behavior after a good marking (including reaching a bad marking) is winning by definition. To compare output indices or letters, only the specific firing sequence in white has to be considered.

the second system player. When the second system player is ahead of the first one or the first system player is two or more steps ahead of the second one, then a good marking is reached and the possible reaching of bad markings afterwards does not matter. Hence, bad markings are only checked for runs where the first system player is zero or one step ahead of the second system player until they reach a good marking for giving an answer to the PCP.

Formally, for each system player, we introduce two *MOD-3 counters* to count the number of output indices and of output letters modulo three. When a player outputs an index, the respective index counter is increased by one, and accordingly for output letters and the letter counter. If a counter would reach value three, it is reset to zero. We define good markings based on the two MOD-3 index counters and the two MOD-3 letter counters. In a *linear firing sequence for indices (letters)*, the two system players output the indices (letters) alternately with the first system player preceding the second one at each turn. We ensure that the environment player first decides that either the output indices or letters are checked for equality. Afterward, a good marking is reached when a firing sequence is not a linear firing sequence for indices or letters, depending on the decision by the environment player.

In Fig. 4, we visualize the reachability graph for the two system players when only considering either the values of their MOD-3 counters for indices or letters. Markings are differentiated in the reachability graph depending on if a good marking is reached before, e.g., position $(0 \parallel 1)$ does not lead to position $(1 \parallel 1)$ as the path to $(1 \parallel 1)$ does not include a good marking. With linear firing sequences, we only consider firing sequences where the first system player outputs the first index or letter before the second system player as the opposite cases are good markings. For firing sequences not reaching a good marking, equality of output indices or letters is checked at positions $(0 \parallel 0)$, $(1 \parallel 1)$, and $(2 \parallel 2)$. Thereby, equality of output indices or letters at the same position can be checked without storing all outputs and it is ensured that solutions have the same length.

Notice that linear firing sequences for indices do not restrict the order in which the two system players output letters between two indices, and vice versa. Also, we at least need a MOD-3 counter. For a MOD-2 counter, the good marking $(2 \parallel 0)$ is replaced by $(0 \parallel 0)$, implying that all firing sequences contain a good marking. A MOD-3 counter prevents that one player overtakes the other. Thus, indices or letters at different positions are not compared, i.e., output indices or letters at position $(0 \parallel 3)$ (not modulo three) can be different.

6.3 Preventing Untruthful Termination

The good markings to only consider linear firing sequences introduce new possibilities for the system players to be winning. These possibilities arise when the system players can enforce all firing sequences to reach a good marking. They occur when a system player terminates without the end symbol ($\#_1$ or $\#_2$) and are called *untruthful termination*. Untruthful termination is prevented by letting the environment player decide which system player it

believes to not terminate with the end symbol or that everything is okay. This decision happens together with the initial choice of the environment player between checking equality of indices or letters. Due to the independence of the players, each system player has to behave as if the environment player is anticipating it to untruthfully terminate and has to output the end symbol to avoid this. Therefore, no untruthful termination can occur.

6.4 Undecidability Results

A winning strategy exists in the Petri game iff there exists a solution to the instance of the PCP. The only strategy with a chance to be winning for the two system players is to output the same solution to the PCP and we can translate solutions between both cases. We obtain:

▶ **Theorem 10.** *For Petri games with good and bad markings and at least two system and one environment player, the question if the system players have a winning strategy is undecidable.*

Bad markings can be encoded by system players repeatedly changing to environment players and back. Players commit to transitions and then system players become environment players. Environment players either follow the committed transition and fire a transition returning to the respective system and environment players or fire a transition with all other environment players after which no good markings are reachable to encode a bad marking. We obtain:

▶ **Theorem 11.** *For Petri games with good markings and at least three players, out of which one is an environment player and each of the other two changes between system and environment player, the question if the system players have a winning strategy is undecidable.*

7 Conclusion

We have investigated global winning conditions for the synthesis of asynchronous distributed reactive systems with causal memory. The general decidability or undecidability of the synthesis problem for these systems is a long-standing open question [30, 15]. We encode the synthesis problem for these systems by Petri games. For global winning conditions, we achieve a clear picture regarding decidability and undecidability.

From our decidability result and previous work [14], we obtain for bad markings as global winning condition that the question of whether the system players have a winning strategy is decidable for Petri games where the number of system players or the number of environment players is at most one and the number of players of the converse type can be bounded by some arbitrary number. For bad markings as global winning condition, this leaves the case of Petri games with two or more system players *and* two or more environment players open.

From our undecidability results, we obtain for good markings as global winning condition that the question of whether the system players have a winning strategy is undecidable for Petri games with two or more system players and three or more environment players. For good markings as global winning condition, this only leaves the corner case of Petri games with at most one system player and at most two environment players open.

Thus, for the synthesis of asynchronous distributed reactive systems with causal memory, global safety winning conditions are decidable for a large class of such systems, whereas global liveness winning conditions are undecidable for almost all classes of such systems. In the future, we plan to combine the decidability results for bad markings as global safety winning condition with local liveness specifications per player as in Flow-LTL [10, 11].

References

1 Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating asynchronous games for distributed synthesis. In *30th International Conference on Concurrency Theory, CONCUR*, volume 140 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.26.

2 Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *Computer Aided Verification – 24th International Conference, CAV*, volume 7358 of *Lecture Notes in Computer Science*. Springer, 2012. doi:10.1007/978-3-642-31424-7_45.

3 Krishnendu Chatterjee and Monika Henzinger. An $O(n^2)$ time algorithm for alternating Büchi games. In *Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. SIAM, 2012. doi:10.1137/1.9781611973099.109.

4 Rüdiger Ehlers. Unbeast: Symbolic bounded synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems – 17th International Conference, TACAS*, volume 6605 of *Lecture Notes in Computer Science*. Springer, 2011. doi:10.1007/978-3-642-19835-9_25.

5 Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6), 1991. doi:10.1007/BF01463946.

6 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.

7 Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of bounded synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems – 23rd International Conference, TACAS*, volume 10205 of *Lecture Notes in Computer Science*, 2017. doi:10.1007/978-3-662-54577-5_20.

8 Bernd Finkbeiner. Bounded synthesis for Petri games. In *Correct System Design – Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, volume 9360 of *Lecture Notes in Computer Science*. Springer, 2015. doi:10.1007/978-3-319-23506-6_15.

9 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Symbolic vs. bounded synthesis for Petri games. In *Sixth Workshop on Synthesis, SYNT@CAV*, volume 260 of *EPTCS*, 2017. doi:10.4204/EPTCS.260.5.

10 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Model checking data flows in concurrent network updates. In *Automated Technology for Verification and Analysis – 17th International Symposium, ATVA*, volume 11781 of *Lecture Notes in Computer Science*. Springer, 2019. doi:10.1007/978-3-030-31784-3_30.

11 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. AdamMC: A model checker for Petri nets with transits against Flow-LTL. In *Computer Aided Verification – 32nd International Conference, CAV*, volume 12225 of *Lecture Notes in Computer Science*. Springer, 2020. doi:10.1007/978-3-030-53291-8_5.

12 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Global winning conditions in synthesis of distributed systems with causal memory (Full Version). *CoRR*, abs/2107.09280, 2021. arXiv:2107.09280.

13 Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. Adam: Causality-based synthesis of distributed systems. In *Computer Aided Verification – 27th International Conference, CAV*, volume 9206 of *Lecture Notes in Computer Science*. Springer, 2015. doi:10.1007/978-3-319-21690-4_25.

14 Bernd Finkbeiner and Paul Gölz. Synthesis in distributed environments. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 93 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.28.

15 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Inf. Comput.*, 253, 2017. doi:10.1016/j.ic.2016.07.006.

16 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.

17 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP*, volume 7966 of *Lecture Notes in Computer Science*. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.

18 Manuel Gieseking, Jesko Hecking-Harbusch, and Ann Yanich. A web interface for Petri nets with transits and Petri games. In *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS*, volume 12652 of *Lecture Notes in Computer Science*. Springer, 2021. doi:10.1007/978-3-030-72013-1_22.

19 Hugo Gimbert. On the control of asynchronous automata. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 93 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.30.

20 Ursula Goltz and Wolfgang Reisig. The non-sequential behavior of Petri nets. *Inf. Control.*, 57(2/3), 1983. doi:10.1016/S0019-9958(83)80040-0.

21 Jesko Hecking-Harbusch and Niklas O. Metzger. Efficient trace encodings of bounded synthesis for asynchronous distributed systems. In *Automated Technology for Verification and Analysis – 17th International Symposium, ATVA*, volume 11781 of *Lecture Notes in Computer Science*. Springer, 2019. doi:10.1007/978-3-030-31784-3_22.

22 Jesko Hecking-Harbusch and Leander Tentrup. Solving QBF by abstraction. In *Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF*, volume 277 of *EPTCS*, 2018. doi:10.4204/EPTCS.277.7.

23 Swen Jacobs, Roderick Bloem, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Philipp J. Meyer, Thibaud Michaud, Mouhammad Sakr, Salomon Sickert, Leander Tentrup, and Adam Walker. The 5th reactive synthesis competition (SYNTCOMP 2018): Benchmarks, participants & results. *CoRR*, abs/1904.07736, 2019. arXiv:1904.07736.

24 Barbara Jobstmann, Stefan J. Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification, 19th International Conference, CAV*, volume 4590 of *Lecture Notes in Computer Science*. Springer, 2007. doi:10.1007/978-3-540-73368-3_29.

25 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, 57(1-2), 2020. doi:10.1007/s00236-019-00349-3.

26 P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *CONCUR – Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-45694-5_11.

27 P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *Lecture Notes in Computer Science*. Springer, 2005. doi:10.1007/11590156_16.

28 José Meseguer, Ugo Montanari, and Vladimiro Sassone. Process versus unfolding semantics for place/transition Petri nets. *Theor. Comput. Sci.*, 153(1&2), 1996. doi:10.1016/0304-3975(95)00121-2.

29 Thibaud Michaud and Maximilien Colange. Reactive synthesis from LTL specification with Spot. In *7th Workshop on Synthesis, SYNT@CAV*, 2018.

30 Anca Muscholl. Automated synthesis of distributed controllers. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP*, volume 9135 of *Lecture Notes in Computer Science*. Springer, 2015. doi:10.1007/978-3-662-47666-6_2.

31 Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, volume 29 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.639.

32 Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, Part I. *Theor. Comput. Sci.*, 13, 1981. doi:10.1016/0304-3975(81)90112-2.

33 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science FOCS*. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

34 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science FOCS*. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

35 Emil L. Post. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.*, 52(4), 1946.

36 John H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2), 1984. doi:10.1016/0022-0000(84)90034-5.

37 Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985. doi:10.1007/978-3-642-69968-9.

38 Sven Schewe. Distributed synthesis is simply undecidable. *Inf. Process. Lett.*, 114(4), 2014. doi:10.1016/j.ipl.2013.11.012.

39 Wieslaw Zielonka. Notes on finite asynchronous automata. *ITA*, 21(2), 1987. doi:10.1051/ita/1987210200991.

Inferring Symbolic Automata

Dana Fisman
Ben-Gurion University, Be'er Sheva, Israel

Hadar Frenkel
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Sandra Zilles
University of Regina, Canada

Abstract

We study the learnability of *symbolic finite state automata*, a model shown useful in many applications in software verification. The state-of-the-art literature on this topic follows the *query learning* paradigm, and so far all obtained results are positive. We provide a necessary condition for efficient learnability of SFAs in this paradigm, from which we obtain the first negative result. The main focus of our work lies in the learnability of SFAs under the paradigm of *identification in the limit using polynomial time and data*. We provide a necessary condition and a sufficient condition for efficient learnability of SFAs in this paradigm, from which we derive a positive and a negative result.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Formal languages and automata theory; Theory of computation → Models of computation; Computing methodologies → Machine learning

Keywords and phrases Symbolic Finite State Automata, Query Learning, Characteristic Sets

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.21

Related Version *Full Version*: `https://arxiv.org/abs/2112.14252`

Funding *Dana Fisman*: This research was partially supported by the Israel Science Foundation (ISF) grant 2507/21.

1 Introduction

Symbolic finite state automata, SFAs for short, are an automata model in which transitions between states correspond to predicates over a domain of concrete alphabet letters. Their purpose is to cope with situations where the domain of concrete alphabet letters is large or infinite. As an example for automata over finite large alphabets consider automata over the alphabet 2^{AP} where AP is a set of atomic propositions; these are used in model checking [21]. Another example, used in string sanitizer algorithms [32], are automata over predicates on the Unicode alphabet which consists of over a million symbols. An infinite alphabet is used for example in *event recording automata*, a determinizable class of timed automata [2] in which an alphabet letter consists of both a symbol from a finite alphabet, and a non-negative real number. Formally, the transition predicates in an SFA are defined wrt. an effective Boolean algebra as defined in §2.

SFAs have proven useful in many applications [23, 44, 10, 34, 45, 39] and consequently have been studied as a theoretical model of automata. Many algorithms for natural operations and decision problems regarding these automata already exist in the literature, in particular, Boolean operations, determinization, and emptiness [49]; minimization [22]; and language inclusion [35]. Recently the subject of learning automata in verification has also attracted attention, as it has been shown useful in many applications, see Vaandrager's survey [48].

There already exists substantial literature on learning restricted forms of SFAs [31, 36, 11, 37, 19], as well as general SFAs [25, 9], and even non-deterministic residual SFAs [20]. For other types of automata over infinite alphabets, [33] suggests learning abstractions, and [47] presents a learning algorithm for deterministic variable automata. All these works consider the query learning paradigm, and provide extensions to Angluin's $\mathbf{L}^*$ algorithm for learning DFAs using membership and equivalence queries [4]. Unique to these works is the work [9] which studies the learnability of SFAs taking as a parameter the learnability of the underlying algebras, providing positive results regarding specific Boolean algebras.

While Argyros and D'Antoni's work [9] is a major advancement towards a systematic way for obtaining results on learnability of SFAs, as it examines the learnability of the underlying algebra, the obtained result allows inferring only positive results, as it relies on a specific query learning algorithm, and does not provide means for obtaining a negative result regarding query learning of SFAs over certain algebras. We provide a necessary condition for efficient learnability of SFAs in the query learning paradigm. From this result we obtain a negative result regarding query learning of SFAs over the propositional algebra. This is, to the best of our knowledge, the first negative result on learning SFAs with membership and equivalence queries and thus gives useful insights into the limitations of the $\mathbf{L}^*$ framework in this context.

The main focus of our work lies on the learning paradigm of *identification in the limit using polynomial time and data*, or its strengthened version *efficient identifiability*. We provide a necessary condition a class of SFAs $\mathbb{M}$ should meet in order to be identified in the limit using polynomial time and data, and a sufficient condition a class of SFAs $\mathbb{M}$ should meet in order to be efficiently identifiable. These conditions are expressed in terms of the existence of certain efficiently computable functions, which we call $\mathsf{Generalize}_{\mathbb{M}}$, $\mathsf{Concretize}_{\mathbb{M}}$, and $\mathsf{Decontaminate}_{\mathbb{M}}$. We then provide positive and negative results regarding the learnability of specific classes of SFAs in this paradigm. In particular, we show that the class of SFAs over *any* monotonic algebras is efficiently identifiable.

2 Preliminaries

2.1 Effective Boolean Algebra

A Boolean Algebra $\mathcal{A}$ can be represented as a tuple $(\mathbb{D}, \mathbb{P}, [\![\cdot]\!], \bot, \top, \vee, \wedge, \neg)$ where $\mathbb{D}$ is a set of domain elements; $\mathbb{P}$ is a set of predicates closed under the Boolean connectives, where $\bot, \top \in \mathbb{P}$; the component $[\![\cdot]\!] : \mathbb{P} \to 2^{\mathbb{D}}$ is the so-called *semantics function*. It satisfies the following three requirements: (i) $[\![\bot]\!] = \emptyset$, (ii) $[\![\top]\!] = \mathbb{D}$, and (iii) for all $\varphi, \psi \in \mathbb{P}$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = \mathbb{D} \setminus [\![\psi]\!]$. A Boolean Algebra is *effective* if all the operations above, as well as satisfiability, are decidable. Henceforth, we implicitly assume Boolean algebras to be effective.

One way to define a Boolean algebra is by defining a set $\mathbb{P}_0$ of *atomic formulas* that includes $\top$ and $\bot$ and obtaining $\mathbb{P}$ by closing $\mathbb{P}_0$ for conjunction, disjunction and negation. For a predicate $\psi \in \mathbb{P}$ we say that ψ is *atomic* if $\psi \in \mathbb{P}_0$. We say that ψ is *basic* if ψ is a conjunction of atomic formulas.

We now introduce two Boolean algebras that are discussed extensively in the paper.

The Interval Algebra is the Boolean algebra in which the domain $\mathbb{D}$ is the set $\mathbb{Z} \cup \{-\infty, \infty\}$ of integers augmented with two special symbols with their standard semantics, and the set of atomic formulas $\mathbb{P}_0$ consists of intervals of the form $[a, b)$ where $a, b \in \mathbb{D}$ and $a \leq b$. The semantics associated with intervals is the natural one: $[\![[a, b)]\!] = \{z \in \mathbb{D} \mid a \leq z \text{ and } z < b\}$.

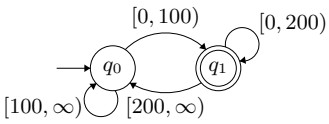


Figure 1 The SFA $\mathcal{M}$ over $\mathcal{A}_{\mathbb{N}}$.

The Propositional Algebra is defined wrt. a set $AP = \{p_1, p_2, \ldots, p_k\}$ of atomic propositions. The set of *atomic predicates* $\mathbb{P}_0$ consists of the atomic propositions and their negations as well as $\top$ and $\bot$. The domain $\mathbb{D}$ consists of all the possible valuations for these propositions, thus it is $\mathbb{B}^k$ where $\mathbb{B} = \{0, 1\}$. The semantics of an atomic predicate p is given by $[\![p_i]\!] = \{v \in \mathbb{B}^k \mid v[i] = 1\}$, and similarly $[\![\neg p_i]\!] = \{v \in \mathbb{B}^k \mid v[i] = 0\}$.[1]

2.2 Symbolic Automata

A *symbolic finite automaton* (SFA) is a tuple $\mathcal{M} = (\mathcal{A}, Q, q_\iota, F, \Delta)$ where $\mathcal{A}$ is a Boolean algebra, Q is a finite set of states, $q_\iota \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\Delta \subseteq Q \times \mathbb{P}_{\mathcal{A}} \times Q$ is a finite set of transitions, where $\mathbb{P}_{\mathcal{A}}$ is the set of predicates of $\mathcal{A}$.

We use the term *letters* for elements of $\mathbb{D}$ where $\mathbb{D}$ is the domain of $\mathcal{A}$ and the term *words* for elements of $\mathbb{D}^*$. A run of $\mathcal{M}$ on a word $a_1 a_2 \ldots a_n$ is a sequence of transitions $\langle q_0, \psi_1, q_1\rangle\langle q_1, \psi_2, q_2\rangle \ldots \langle q_{n-1}, \psi_n, q_n\rangle$ satisfying that $a_i \in [\![\psi_i]\!]$, that $\langle q_i, \psi_{i+1}, q_{i+1}\rangle \in \Delta$ and that $q_0 = q_\iota$. Such a run is said to be *accepting* if $q_n \in F$. A word $w = a_1 a_2 \ldots a_n$ is said to be *accepted* by $\mathcal{M}$ if there exists an accepting run of $\mathcal{M}$ on w. The set of words accepted by an SFA $\mathcal{M}$ is denoted $\mathcal{L}(\mathcal{M})$. We use $\hat{\mathcal{L}}(\mathcal{M})$ for the set $\{\langle w, 1\rangle \mid w \in \mathcal{L}(\mathcal{M})\} \cup \{\langle w, 0\rangle \mid w \notin \mathcal{L}(\mathcal{M})\}$.

An SFA is said to be *deterministic* if for every state $q \in Q$ and every letter $a \in \mathbb{D}$ we have that $|\{\langle q, \psi, q'\rangle \in \Delta \mid a \in [\![\psi]\!]\}| \leq 1$, namely from every state and every concrete letter there exists at most one transition. It is said to be *complete* if $|\{\langle q, \psi, q'\rangle \in \Delta \mid a \in [\![\psi]\!]\}| \geq 1$ for every $q \in Q$ and $a \in \mathbb{D}$, namely from every state and every concrete letter there exists at least one transition. It is not hard to see that, as is the case for finite automata (over concrete alphabets), non-determinism does not add expressive power but does add succinctness. When $\mathcal{A}$ is deterministic we use $\Delta(q, w)$ to denote the state $\mathcal{A}$ reaches on reading word w from state q. If $\Delta(q_\iota, w) = q$ then w is termed an *access word to state* q.

▶ **Example 1.** Consider the SFA $\mathcal{M}$ given in Fig.1. It is defined over the algebra $\mathcal{A}_{\mathbb{N}}$ which is the interval algebra restricted to the domain $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. The language of $\mathcal{M}$ is the set of all words over $\mathbb{D}$ of the form $w_1 \cdot d \cdot w_2$ where w_1 is some word over the domain $\mathbb{D}$, the letter d satisfies $0 \leq d < 100$ and all letters of the word w_2 are numbers smaller than 200.

3 Learning SFAs

In grammatical inference, loosely speaking, we are interested in learning a class of languages $\mathbb{L}$ over an alphabet Σ, from examples which are words over Σ. Examples for classes of languages can be the set of regular languages, the set of context-free languages, etc. A learning algorithm, aka a *learner*, is expected to output some concise representation of the language from a class of representations $\mathbb{R}$ for the class $\mathbb{C}$. For instance, in learning the class $\mathbb{L}_{reg}$ of regular languages one might consider the class $\mathbb{R}_{\text{DFA}}$ of DFAs, or the class

[1] In this case a basic formula is a *monomial*.

$\mathbb{R}_{\text{LIN}}$ of right linear grammars, since both are capable of expressing all regular languages.[2] We often say that a class of representations $\mathbb{R}$ is learnable (or not) when we mean that a class of languages $\mathbb{L}$ is learneable (or not) via the class of representations $\mathbb{R}$. Complexity of learning an unknown language $L \in \mathbb{L}$ via $\mathbb{R}$ is typically measured wrt. the size of the smallest representation $R_L \in \mathbb{R}$ for L. For instance, when learning $\mathbb{L}_{reg}$ via $\mathbb{R}_{\text{DFA}}$ a learner is expected to output a DFA for an unknown language in time that is polynomial in the number of states of the minimal DFA for L.

In our setting we are interested in learning regular languages using as a representation classes of SFAs over a certain algebra. To measure complexity we must agree on how to measure the size of an SFA. For DFAs, the number of states is a common measure of size, since the DFA can be fully described by a representation of size polynomial in the number of states. In the case of SFA the situation is different, as the size of the predicates labeling the transitions can vary greatly. In fact, if we measure the size of a predicate by the number of nodes in its parse DAG, then the size of a formula can grow unboundedly. The size and structure of the predicates influence the complexity of their satisfiability check, and thus the complexity of the corresponding algorithms. Another thing to note is that there might be a trade-off between the size of the transition predicates and the number of transitions; e.g. a predicate of the form $\psi_1 \vee \psi_2 \ldots \vee \psi_k$ can be replaced by k transitions, each one labeled by one ψ_i for $1 \leq i \leq k$.

The literature defines an SFA as *normalized* if for every two states q and q' there exists at most one transition from q to q'. This definition prefers fewer transitions over potentially complicated predicates. By contrast, preferring simple transitions at the cost of increasing the number of transitions, leads to *neat* SFAs. An SFA is termed *neat* if all transition predicates are basic predicates. In [27] we proposed to measure the size of an SFA by three parameters: the number of states (n), the maximal out-degree of a state (m) and the size of the most complex predicate (l); we then analyzed the complexity of the standard operations on SFAs, with particular attention to the mentioned special forms. Another important factor regarding size and canonical forms of SFAs, is the underlying algebra, specifically, whether it is monotonic or not.

Monotonicity. A Boolean algebra $\mathcal{A}$ over domain $\mathbb{D}$ is said to be *monotonic* if there exists a total order $<$ on the elements of $\mathbb{D}$, there exist two elements $d_{-\infty}, d_{\infty}$ such that $d_{-\infty} \leq d$ and $d \leq d_{\infty}$ for all $d \in \mathbb{D}$, and an atomic predicate $\psi \in \mathbb{P}_0$ can be associated with two concrete values a and b such that $[\![\psi]\!] = \{d \in \mathbb{D} \mid a \leq d < b\}$. The interval algebra (given in §2.1) is clearly monotonic, as is the similar algebra obtained using $\mathbb{R}$ (the real numbers) instead of $\mathbb{Z}$ (the integers). On the other hand, the propositional algebra is clearly non-monotonic.

Learning Paradigms. The exact definition regarding learnability of a class depends on the *learning paradigm*. In this work we consider two widely studied paradigms: *learning with membership and equivalence queries*, and *identification in the limit using polynomial time and data*. Their definitions are provided in the respective sections.

Non-Trivial Classes of SFAs. In the sequel we would like to prove results regarding non-trvial classes of SFAs, which are defined as follows.

[2] The class of regular languages was shown learnable via various representations including DFAs [4], NFAs [16], and AFAs (alternating finite automata) [7].

▶ **Definition 2.** *A class of SFAs $\mathbb{M}$ over a Boolean Algebra $\mathfrak{A}$ with a set of predicates $\mathbb{P}$ is termed* non-trivial *if for every predicate $\varphi \in \mathbb{P}$ the SFA $\mathcal{M}_\varphi = (\mathfrak{A}, \{q_\iota, q_{ac}, q_{rj}\}, q_\iota, \{q_{ac}\}, \Delta)$ where $\Delta = \{\langle q_\iota, \varphi, q_{ac}\rangle, \langle q_\iota, \neg\varphi, q_{rj}\rangle, \langle q_{rj}, \top, q_{rj}\rangle, \langle q_{ac}, \top, q_{rj}\rangle\}$ is in $\mathbb{M}$. Note that $\mathcal{M}_\varphi$ accepts only words of length one consisting of a concrete letter satisfying φ, and it is minimal among all complete deterministic SFAs accepting this language (minimal in both number of states and number of transitions).*

4 Efficient Identifiability

While in *active learning* (e.g. query learning) the learner can select any word and query about its membership in the unknown language, in *passive learning* the learner is given a set of words, and for each word w in the set, a label b_w indicating whether w is in the unknown language or not. Formally, a *sample* for a language L is a finite set $\mathcal{S}$ consisting of labeled examples, that is, pairs of the form $\langle w, b_w\rangle$ where w is a word and $b_w \in \{0, 1\}$ is its label, satisfying that $b_w = 1$ if and only if $w \in L$. The words that are labeled 1 are termed *positive* words, and those that are labeled 0 are termed *negative* words. Note that if L is recognized by $\mathcal{M}$, we have that $\mathcal{S} \subseteq \hat{\mathcal{L}}(\mathcal{M})$ (as defined in §.2.2). If $\mathcal{S}$ is a sample for L we often say that $\mathcal{S}$ *agrees with* L. Given two words w, w', we say that w and w' are not equivalent wrt. $\mathcal{S}$, denoted $w \not\sim_{\mathcal{S}} w'$, iff there exists z such that $\langle wz, b\rangle, \langle w'z, b'\rangle \in \mathcal{S}$ and $b \neq b'$. Otherwise we say that w and w' are equivalent wrt. $\mathcal{S}$, and write $w \sim_{\mathcal{S}} w'$.

Given a sample $\mathcal{S}$ for a language L over a concrete domain $\mathbb{D}$, it is possible to construct a DFA that agrees with $\mathcal{S}$ in polynomial time. Indeed one can create the *prefix-tree automaton*, a simple automaton that accepts all and only the positively labeled words in the sample. Clearly the constructed automaton may not be the minimal automaton that agrees with $\mathcal{S}$. There are several algorithms, in particular the popular RPNI [42], that minimize the prefix-tree automaton, and due to state merging may accept an infinite language. Obviously though, this procedure is not guaranteed to return an automaton for the unknown language, as the sample may not provide sufficient information. For instance if $L = aL_1 \cup bL_2$ and the sample contains only words starting with a, there is no way for the learner to infer L_2 and hence also L correctly. One may thus ask, given a language L, what should a sample contain in order for a passive learning algorithm to infer L correctly, and can such sample be of polynomial size with respect to a minimal representation (e.g., a DFA) for the language.

One approach to answer these questions is captured in the paradigm of *identification in the limit using polynomial time and data*. This model was proposed by Gold [28], who also showed that it admits learning of regular languages represented by DFAs. We follow de la Higuera's more general definition [24].[3] This definition requires that for any language L in a class of languages $\mathbb{L}$ represented by $\mathbb{R}$, there exists a sample $\mathcal{S}_L$ of size polynomial in the size of the smallest representation $R \in \mathbb{R}$ of L (e.g., the smallest DFA for L), such that a valid learner can infer the unknown language L from the information contained in $\mathcal{S}_L$. The set $\mathcal{S}_L$ is then termed a *characteristic sample*.[4] Here, a valid learner is an algorithm that learns the target language exactly and efficiently. In particular, a valid learner produces in polynomial time a representation that agrees with the provided sample. The learner also has

[3] This paradigm may seem related to conformance testing. The relation between conformance testing for Mealy machines and automata learning of DFAs has been explored in [14].

[4] De la Higuera's notion of characteristic sample is a core concept in grammatical inference, for various reasons. Firstly, it addresses shortcomings of several other attempts to formulate polynomial-time learning in the limit [5, 43]. Secondly, this notion has inspired the design of popular algorithms for learning formal languages such as, for example, the RPNI algorithm [42]. Thirdly, it was shown to bear strong relations to a classical notion of machine teaching [30]; models of the latter kind are currently experiencing increased attention in the machine learning community [50].

to correctly learn the unknown language L when given the characteristic sample $\mathcal{S}_L$ as input. Moreover, if the input sample $\mathcal{S}$ subsumes $\mathcal{S}_L$ yet is still consistent with L, the additional information in the sample should not "confuse" the learner; the latter still has to output a correct representation for L. (Intuitively, this requirement precludes situations in which the sample consists of some smart encoding of the representation that the learner simply deciphers. In particular, the learner will not be confused if an adversary "contaminates" the characteristic sample by adding labeled examples for the target language.) We provide the formal definition after the following informal example.

▶ **Example 3.** For the class of DFAs, let us consider the regular language $L = a^*$ over the alphabet $\{a, b\}$. Further, consider a sample set $\mathcal{S} = \{\langle\epsilon, 1\rangle, \langle a, 1\rangle, \langle b, 0\rangle, \langle bb, 0\rangle, \langle ba, 0\rangle\}$ for L. There is a valid learner for the class of all DFAs that uses the sample $\mathcal{S}$ as a characteristic sample for L. By definition, such a learner has to output a DFA for L when fed with $\mathcal{S}$, but also has to output equivalent DFAs whenever given any superset of $\mathcal{S}$ as input, as long as this superset agrees with L. Naturally, the sample $\mathcal{S}$ is also consistent with the regular language $L' = \{\epsilon, a\}$. However, this does not pose any problem, since the same learner can use a characteristic sample for L' that disagrees with L, for example, $\mathcal{S}' = \{\langle\epsilon, 1\rangle, \langle a, 1\rangle, \langle aa, 0\rangle\}$. When defining a system of characteristic samples like that, the core requirement is that the size of a sample be bounded from above by a function that is polynomial in the size of the smallest DFA for the respective target language.

▶ **Definition 4** (identification in the limit using polynomial time and data)**.** *A class of languages $\mathbb{L}$ is said to be* identified in the limit using polynomial time and data *via representations in a class $\mathbb{R}$ if there exists a learning algorithm $\boldsymbol{A}$ such that the following requirements are met.*

1. *Given a finite sample $\mathcal{S}$ of labeled examples, $\boldsymbol{A}$ returns a hypothesis $\mathcal{R} \in \mathbb{R}$ that agrees with $\mathcal{S}$ in polynomial time.*
2. *For every language $L \in \mathbb{L}$, there exists a sample $\mathcal{S}_L$, termed* a characteristic sample*, of size polynomial in the minimal representation $\mathcal{R} \in \mathbb{R}$ for L such that the algorithm $\boldsymbol{A}$ returns a correct hypothesis when run on any sample $\mathcal{S}$ for L that subsumes $\mathcal{S}_L$.*

Note that the first condition ensures polynomial time and the second polynomial data. However, the latter is not a worst-case measure; the algorithm may fail to return a correct hypothesis on arbitrarily large finite samples (if they do not subsume a characteristic set).

Note also that the definition does not require the existence of an efficient algorithm that constructs a characteristic sample for each language in the underlying class. When such an algorithm is also available we say that the class is *efficiently identifiable*. In the full version of the paper we provide an example of a class of languages that possesses polynomial-size characteristic sets, yet without the ability to construct such sets effectively. Since we are concerned with learning classes of automata we formulate the definition of *efficient identification* directly over classes of automata.

▶ **Definition 5** (efficient identification)**.** *A class of automata $\mathbb{M}$ over an alphabet Σ is said to be* efficiently identified *if the following two requirements are met.*

1. *There exists a polynomial time learning algorithm $\boldsymbol{Infer} : 2^{(\Sigma^* \times \{0,1\})} \to \mathbb{M}$ such that, for any sample $\mathcal{S}$, we have $\mathcal{S} \subseteq \hat{\mathcal{L}}(\boldsymbol{Infer}(\mathcal{S}))$.*
2. *There exists a polynomial time algorithm $\boldsymbol{Char} : \mathbb{M} \to 2^{(\Sigma^* \times \{0,1\})}$ such that, for every $\mathcal{M} \in \mathbb{M}$ and every sample $\mathcal{S}$ satisfying $\boldsymbol{Char}(\mathcal{M}) \subseteq \mathcal{S} \subseteq \hat{\mathcal{L}}(\mathcal{M})$, the automaton $\boldsymbol{Infer}(\mathcal{S})$ recognizes the same language as $\mathcal{M}$.*

When we apply this definition for a class of SFAs over a Boolean algebra $\mathcal{A}$ with domain $\mathbb{D}$ and predicates $\mathbb{P}$, the characteristic sample is defined over the concrete set of letters $\mathbb{D}$ rather than the set of predicates $\mathbb{P}$ because this is the alphabet of the words accepted by an SFA (inferring an SFA from a set of words labeled by predicates can be done using the methods for inferring DFAs, by considering the alphabet to be the set of predicates).

Throughout this section we study whether a class of SFAs $\mathbb{M}$ is efficiently identifiable. That is, we are interested in the existence of algorithms $\mathbf{Infer}_{\mathbb{M}}$ and $\mathbf{Char}_{\mathbb{M}}$ satisfying the requirements of Def.5. In §4.1 we provide a necessary condition for a non-trivial class of SFAs to be identified in the limit using polynomial time and data. In §4.2 we provide a sufficient condition for a non-trivial class of SFAs to be efficiently identifiable. On the positive side, we show in §4.3 that the class of SFAs over the interval algebra is efficiently identifiable. On the negative side, we show in §4.4 that SFAs over the general propositional algebra cannot be identified in the limit using polynomial time and data.

Efficient Identification of DFAs

Before investigating efficient identification of SFAs, it is worth noting that DFAs are efficiently identifiable. We state a result that provides more details about the nature of these algorithms, since we need it later, in §.4.3, to provide our positive result. Intuitively, it says that there exists a valid learner such that if $\mathcal{D}$ is a minimal DFA recognizing a certain language L then the learner can infer L from a characteristic sample consisting of access words to each state of $\mathcal{D}$ and their extensions with distinguishing words (words showing each pair of states cannot be merged) as well as one letter extensions of the access words that are required to retrieve the transition relation.

▶ **Theorem 6** ([42]). *I. The class of DFAs is efficiently identifiable via procedures* ***CharDFA*** *and* ***InferDFA****. II. Furthermore, these procedures satisfy that if $\mathcal{D}$ is a minimal and complete DFA and* $\mathbf{CharDFA}(\mathcal{D}) = \mathcal{S}_{\mathcal{D}}$ *then the following holds:*

1. *$\mathcal{S}_{\mathcal{D}}$ contains a prefix-closed set A of access words. Moreover, A can be chosen to contain only lex-access words, i.e., only the lexicographically smallest access word for each state.*
2. *For every $u_1, u_2 \in A$ it holds that $u_1 \not\sim_{\mathcal{S}_{\mathcal{D}}} u_2$.*
3. *For every $u, v \in A$ and $\sigma \in \Sigma$, if $\Delta(q_\iota, u\sigma) \neq \Delta(q_\iota, v)$ then $u\sigma \not\sim_{\mathcal{S}_{\mathcal{D}}} v$.*

We briefly describe **CharDFA** and **InferDFA**.

The algorithm **CharDFA** works as follows. It first creates a prefix-closed set of access words to states. This can be done by considering the graph of the automaton and running an algorithm for finding a spanning tree from the initial state. Choosing one of the letters on each edge, the access word for a state is obtained by concatenating the labels on the unique path of the obtained tree that reaches that state. If we wish to work with lex-access words, we can use a depth-first search algorithm that spans branches according to the order of letters in Σ, starting from the smallest. The labels on the paths of the spanning tree constructed this way will form the set of lex-access words. Let S be the set of access words (or lex-access words). Next the algorithm turns to find a distinguishing word $v_{i,j}$ for every pair of state $s_i, s_j \in S$ (where $s_i \neq s_j$). It holds that any pair of states of the minimal DFA has a distinguishing word of size quadratic in the size of the DFA. Let E be the set of all such distinguishing words. Then the algorithm returns the set $\mathcal{S}_{\mathcal{D}} = \{\langle w, \mathcal{D}(w)\rangle \mid w \in (S \cdot E) \cup (S \cdot \Sigma \cdot E)\}$ where $\mathcal{D}(w)$ is the label $\mathcal{D}$ gives w (i.e. 1 if it is accepted, and 0 otherwise). It is easy to see that $\mathcal{S}_{\mathcal{D}}$ satisfies the properties of Thm.6.

The algorithm **InferDFA**, given a sample of words $\mathcal{S}$, infers from it in polynomial time a DFA that agrees with $\mathcal{S}$. Moreover, if $\mathcal{S}$ subsumes the characteristic set $\mathcal{S}_{\mathcal{D}}$ of a DFA $\mathcal{D}$ then **InferDFA** returns a DFA that recognizes $\mathcal{D}$. Let W be the set of words in the given sample $\mathcal{S}$ (without their labels). Let R be the set of prefixes of W and C the set of suffixes of W. Note that $\epsilon \in R$ and $\epsilon \in C$. Let $r_0, r_1, \ldots$ be some enumeration of R and $c_0, c_1, \ldots$ some enumeration of C where $r_0 = c_0 = \epsilon$. The algorithm builds a matrix M of size $|R| \times |C|$ whose entries take values in $\{0, 1, ?\}$, and sets the value of entry (i, j) as follows. If $r_i c_j$ is

not in W, it is set to ?. Otherwise it is set to 1 iff the word $r_i c_j$ is labeled 1 in $\mathcal{S}$. We get that $r_i \sim_{\mathcal{S}} r_j$ iff for every k such that both $M(i,k)$ and $M(j,k)$ are different than ? we have that $M(i,k) = M(j,k)$. The algorithm sets $R_0 = \{\epsilon\}$. Once R_i is constructed, the algorithm tries to establish whether for $r \in R_i$ and $\sigma \in \Sigma$, $r\sigma$ is distinguished from all words in R_i. It does so by considering all other words $r' \in R_i$ and checking whether $r \sim_{\mathcal{S}} r'$. If $r\sigma$ is found to be distinct from all words in R_i, then R_{i+1} is set to $R_i \cup \{r\sigma\}$. The algorithm proceeds until no new words are distinguished. Let k be the iteration of convergence. If not all words in R_k are in W (that is $M(i,0) =?$ for some $r_i \in R_k$), the algorithm returns the prefix-tree automaton. Otherwise, the states of the constructed DFA are set to be the words in R_k. The initial state is ϵ and a state r_i is classified as accepting iff $M(i,0) = 1$ (recall that the entry $M(i,0)$ stands for the value of $r_i \cdot \epsilon$ in $\mathcal{S}$). To determine the transitions, for every $r \in R_k$ and $\sigma \in \Sigma$, recall that there exists at least one state $r' \in R$ that cannot be distinguished from $r\sigma$. The algorithm then adds a transition from r on σ to r'.

4.1 Necessary Condition

We make use of the following definitions. A sequence $\langle \Gamma_1, \ldots, \Gamma_m \rangle$ consisting of sets of concrete letters $\Gamma_i \subseteq \mathbb{D}$ is termed a *concrete partition* of $\mathbb{D}$ if the sets are pairwise disjoint (namely $\Gamma_i \cap \Gamma_j = \emptyset$ for every $i \neq j$). Note that we do not require that in addition $\bigcup_{1 \leq i \leq k} \Gamma_i = \mathbb{D}$. We use $\Pi_{\mathsf{conc}}(\mathbb{D}, m)$ to define the set of all concrete partitions of size m over $\mathbb{D}$. A sequence of predicates $\langle \psi_1, \ldots, \psi_m \rangle$ over a Boolean algebra $\mathcal{A}$ on a domain $\mathbb{D}$ is termed a *predicate partition* if $[\![\psi_i]\!] \cap [\![\psi_j]\!] = \emptyset$ for every $i \neq j$, and in addition $\bigcup_{\leq i \leq k} [\![\psi_i]\!] = \mathbb{D}$. That is, here we do require the assignments to the predicates cover the domain. We use $\Pi_{\mathsf{pred}}(\mathbb{P}, m)$ to define the set of all predicate partitions of size m over $\mathbb{P}$.

▶ **Definition 7.**

- *A function f_g from a concrete partition to a predicate partition is termed* generalizing *if $f_g(\langle \Gamma_1, \ldots, \Gamma_m \rangle) = \langle \psi_1, \ldots, \psi_k \rangle$ implies $k = m$ and $[\![\psi_i]\!] \supseteq \Gamma_i$ for all $1 \leq i \leq m$.*
- *A function f_c from a predicate partition to a concrete partition is termed* concretizing *if $f_c(\langle \psi_1, \ldots, \psi_m \rangle) = \langle \Gamma_1, \ldots, \Gamma_k \rangle$ implies $k = m$ and $\Gamma_i \subseteq [\![\psi_i]\!]$ for all $1 \leq i \leq m$.*

Note that f_g and f_c are *variadic* functions (i.e. can take any number of parameters). We can define their k-adic versions as those that work only on partitions of size k. In particular, their *dyadic* versions work only on partitions of size 2.

We say that f_g (resp. f_c) is *efficient* if it can be computed in polynomial time. Note that if f_c is efficient then the sets Γ_i in the constructed concrete partition are of polynomial size.

We are now ready to provide a necessary condition for identifiability in the limit using polynomial time and data.

▶ **Theorem 8.** *A necessary condition for a non-trivial class of SFAs $\mathbb{M}_{\mathcal{A}}$ over a Boolean algebra $\mathcal{A}$ to be identified in the limit using polynomial time and data is that there exist efficient dyadic concretizing and generalizing functions, $\mathsf{Concretize}_{\mathcal{A}} : \Pi_{pred}(\mathbb{P}, 2) \to \Pi_{conc}(\mathbb{D}, 2)$ and $\mathsf{Generalize}_{\mathcal{A}} : \Pi_{conc}(\mathbb{D}, 2) \to \Pi_{pred}(\mathbb{P}, 2)$, satisfying that*

$$\begin{gathered} \text{if } \mathsf{Concretize}_{\mathcal{A}}(\langle \psi_1, \psi_2 \rangle) = \langle \Gamma_1, \Gamma_2 \rangle \\ \text{and } \mathsf{Generalize}_{\mathcal{A}}(\langle \Gamma'_1, \Gamma'_2 \rangle) = \langle \varphi_1, \varphi_2 \rangle \\ \text{where } \Gamma_i \subseteq \Gamma'_i \text{ for every } 1 \leq i \leq 2 \\ \text{then } [\![\varphi_i]\!] = [\![\psi_i]\!] \text{ for every } 1 \leq i \leq 2. \end{gathered}$$

Proof. Assume that $\mathbb{M}_{\mathcal{A}}$ is identified in the limit using polynomial time and data. That is, there exist two algorithms $\mathsf{CharSFA} : \mathbb{M}_{\mathcal{A}} \to 2^{\mathbb{D}^* \times \{0,1\}}$ and $\mathsf{InferSFA} : 2^{\mathbb{D}^* \times \{0,1\}} \to \mathbb{M}_{\mathcal{A}}$ satisfying the requirements of Def.4. We show that efficient dyadic concretizing and generalizing functions do exist.

We start with the definition of $\mathsf{Concretize}_{\mathcal{A}}$. Let $\langle \varphi_1, \varphi_2 \rangle$ be the argument of $\mathsf{Concretize}_{\mathcal{A}}$. Note that $\varphi_2 = \neg\varphi_1$ by the definition of a predicate partition. The implementation of $\mathsf{Concretize}_{\mathcal{A}}$ invokes $\mathsf{CharSFA}$ on the SFA $\mathcal{M}_{\varphi_1}$ accepting all words of length one consisting of a concrete letter satisfying φ_1, as defined in Def.2. Let $\mathcal{S}$ be the returned sample. Let Γ_1 be the set of positively labeled words in the sample. Note that all such words are of size one, namely they are letters. Let Γ_2 be the set of letters that are first letters in a negative word in the sample. Then $\mathsf{Concretize}_{\mathcal{A}}$ returns $\langle \Gamma_1, \Gamma_2 \rangle$.

We turn to the definition of $\mathsf{Generalize}_{\mathcal{A}}$. Given $\langle \Gamma_1, \Gamma_2 \rangle$ the implementation of $\mathsf{Generalize}_{\mathcal{A}}$ invokes $\mathsf{InferSFA}$ on sample $\mathcal{S} = \{\langle \gamma, 1\rangle \mid \gamma \in \Gamma_1\} \cup \{\langle \gamma, 0\rangle \mid \gamma \in \Gamma_2\} \cup \{\langle \gamma\gamma', 0\rangle \mid \gamma, \gamma' \in \Gamma_1 \cup \Gamma_2\}$. That is, all one-letter words satisfying Γ_1 are positively labeled, all one-letter words satisfying Γ_2 are negatively labeled, and all words of length 2 using some of the given concrete letters, are negatively labeled. Let $\mathcal{M}$ be the returned SFA when given $\mathcal{S}' \supseteq \mathcal{S}$ as an input. Let Ψ_1 be the set of all predicates labeling some edge from the initial state to an accepting state, and let Ψ_2 be the set of all predicates labeling some edge from the initial state to a rejecting state. Let $\varphi = (\bigvee_{\psi \in \Psi_1} \psi) \wedge (\bigwedge_{\psi \in \Psi_2} \neg\psi)$. Then $\mathsf{Generalize}_{\mathcal{A}}$ returns $\langle \varphi, \neg\varphi \rangle$.

It is not hard to verify that the constructed methods $\mathsf{Generalize}_{\mathcal{A}}$ and $\mathsf{Concretize}_{\mathcal{A}}$ satisfy the requirements of the theorem. ◀

The following example shows that for some Boolean algebras, such functions exist, even for a generalization of the requirement for variadic versions of $\mathsf{Concretize}$ and $\mathsf{Generalize}$.

▶ **Example 9.** Consider the class $\mathbb{M}_{\mathcal{A}_\mathbb{N}}$ of SFAs over the algebra $\mathcal{A}_\mathbb{N}$ of Ex.1 and consider the functions $\mathsf{Concretize}_{\mathcal{A}_\mathbb{N}}(\langle [d_1, d_1'), [d_2, d_2'), \ldots, [d_m, d_m')\rangle) = \langle \{d_1\}, \ldots, \{d_m\}\rangle$ and $\mathsf{Generalize}_{\mathcal{A}_\mathbb{N}}(\langle \Gamma_1, \ldots, \Gamma_m\rangle) = \langle [\min \Gamma_{\pi(1)}, \min \Gamma_{\pi(2)}), [\min \Gamma_{\pi(2)}, \min \Gamma_{\pi(3)}), \ldots, [\min \Gamma_{\pi(m)}, \infty)\rangle$ where π is the permutation on $(1, \ldots, m)$ satisfying $\max \Gamma_{\pi(i)} < \min \Gamma_{\pi(i+1)}$ for every $1 \leq i < m$. Then, $\mathsf{Concretize}_{\mathcal{A}_\mathbb{N}}$ and $\mathsf{Generalize}_{\mathcal{A}_\mathbb{N}}$ satisfy the variadic generalization of the conditions of Thm.8.

We would like to relate the necessary condition on the learnability of a class of SFAs over a Boolean algebra $\mathcal{A}$ to the learnability of the Boolean algebra $\mathcal{A}$ itself. For this we need to first define efficient identifiability of a Boolean algebra $\mathcal{A}$. Since to learn an unknown predicate we need to supply two sets: one of negative examples and one of positive examples, it makes sense to say that a Boolean algebra $\mathcal{A}$ with predicates $\mathbb{P}$ over domain $\mathbb{D}$ is *efficiently identifiable* if there exist efficient dyadic concretizing and generalizing functions, $\mathsf{Concretize}_{\mathcal{A}} : \Pi_{\mathsf{pred}}(\mathbb{P}, 2) \to \Pi_{\mathsf{conc}}(\mathbb{D}, 2)$ and $\mathsf{Generalize}_{\mathcal{A}} : \Pi_{\mathsf{conc}}(\mathbb{D}, 2) \to \Pi_{\mathsf{pred}}(\mathbb{P}, 2)$ satisfying the criteria of Theorem 8. Using this terminology we can state the following corollary.

▶ **Corollary 10.** *Efficient identifiability of the Boolean algebra $\mathcal{A}$ is a necessary condition for identification in the limit using polynomial time and data of any non-trivial class of SFAs over $\mathcal{A}$.*

4.2 Sufficient Condition

We turn to discuss a sufficient condition for the efficient identifiability of a class of SFAs $\mathbb{M}_{\mathcal{A}}$ over a Boolean algebra $\mathcal{A}$. To prove that $\mathbb{M}_{\mathcal{A}}$ is efficiently identifiable, we need to supply two algorithms $\mathbf{CharSFA}_{\mathbb{M}_{\mathcal{A}}}$ and $\mathbf{InferSFA}_{\mathbb{M}_{\mathcal{A}}}$ as required in Def.5. The idea is to reduce the problem to efficient identifiablity of DFAs, namely to use the algorithms **CharDFA** and **InferDFA** provided in Thm.6. The implementation of **CharSFA**, given an SFA $\mathcal{M}$ will transform it into a DFA $\mathcal{D}_{\mathcal{M}}$ by applying $\mathsf{Concretize}_{\mathcal{A}}$ on the partitions induced by the states of the DFA. The resulting DFA $\mathcal{D}_{\mathcal{M}}$ will not be equivalent to the given SFA $\mathcal{M}$, but it may be used to create a sample of words $\mathcal{S}_{\mathcal{M}}$ that is a characteristic set for $\mathcal{M}$, see Fig.2.

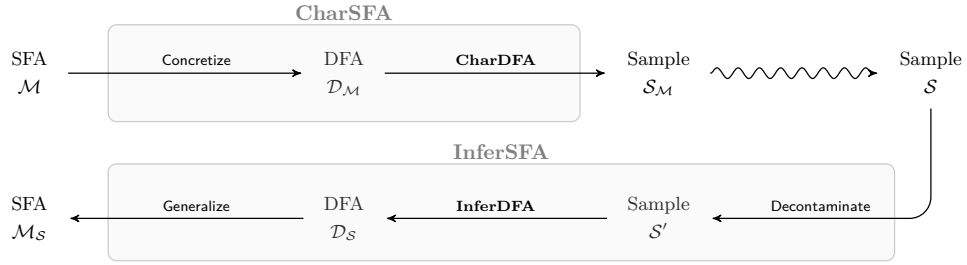


Figure 2 A schematic description of algorithms **CharSFA** and **InferSFA**.

Algorithm 1 $\mathsf{Concretize}_{\mathbb{M}_\mathcal{A}}(\mathcal{M})$.

Input: An SFA $\mathcal{M}$, function $\mathsf{Concretize}_\mathcal{A}$
Output: a DFA $\mathcal{D}_\mathcal{M}$

function $\textsc{Concretize}_{\mathbb{M}_\mathcal{A}}(\mathcal{M} = \langle \mathcal{A}, Q, q_\iota, F, \Delta \rangle)$
  $\Gamma_\mathcal{M} := \bigcup_{q \in Q} \mathsf{Concretize}_\mathcal{A}(\pi_q)$
  $\Delta_\mathcal{D} := \emptyset$
  for all $q, q' \in Q, \psi \in \pi_q, d \in \Gamma_\mathcal{M}$ **do**
    if $\langle q, \psi, q' \rangle \in \Delta$ and $d \in [\![\psi]\!]$ **then**
      $\Delta_\mathcal{D} := \Delta_\mathcal{D} \cup \langle q, d, q' \rangle$
  return $\mathcal{D}_\mathcal{M} := \langle \Gamma_\mathcal{M}, Q, q_\iota, F, \Delta_\mathcal{D} \rangle$

Algorithm 2 $\mathsf{Generalize}_{\mathbb{M}_\mathcal{A}}(\mathcal{M})$.

Input: A DFA $\mathcal{D}$, function $\mathsf{Generalize}_\mathcal{A}$
Output: An SFA $\mathcal{M}$

function $\textsc{Generalize}_{\mathbb{M}_\mathcal{A}}(\mathcal{D} = \langle \Sigma, Q, q_\iota, F, \Delta_\mathcal{D} \rangle)$
  $\Delta_\mathcal{M} := \emptyset$
  for all $q \in Q$ **do**
    for all $q_i \in Q$ **do** $\Gamma_i := \{\gamma \mid \langle q, \gamma, q_i \rangle \in \Delta_\mathcal{D}\}$
    $\langle \psi_1, \ldots, \psi_n \rangle := \mathsf{Generalize}_\mathcal{A}(\langle \Gamma_1, \ldots, \Gamma_n \rangle)$
    for all $q_i \in Q$ **do** $\Delta_\mathcal{M} := \Delta_\mathcal{M} \cup \langle q, \psi_i, q_i \rangle$
  return $\mathcal{M} := \langle \mathcal{A}, Q, q_\iota, F, \Delta_\mathcal{M} \rangle$

To implement **InferSFA** we would like to use **InferDFA** to obtain, as a first step, a DFA from the given sample, then at the second step, apply $\mathsf{Generalize}_\mathcal{A}$ on the concrete-partitions induced by the DFA states. A subtle issue that we need to cope with is that inference should succeed also on samples subsuming the characteristic sample. The fact that this holds for inference of the DFA does not suffice, since we are guaranteed that the inference of the DFA will not be confused if the sample contains more labeled words, as long as the new words are over the same alphabet. In our case the alphabet of the sample can be a strict subset of the concrete letters $\mathbb{D}$ (and if $\mathbb{D}$ is infinite, this surely will be the case).[5] So we need an additional step to remove words from the given sample if they are not over the alphabet of the characteristic sample. We call a method implementing this $\mathsf{Decontaminate}_{\mathbb{M}_\mathcal{A}}$.

Formally, we first define the extension of $\mathsf{Concretize}_\mathcal{A}$ and $\mathsf{Generalize}_\mathcal{A}$ to automata instead of partitions, which we term $\mathsf{Concretize}_{\mathbb{M}_\mathcal{A}}$ and $\mathsf{Generalize}_{\mathbb{M}_\mathcal{A}}$ (with $\mathbb{M}$ in the subscript).

- The formal definition of $\mathsf{Concretize}_{\mathbb{M}_\mathcal{A}}$ is given in Alg.1. Let $\mathcal{M} = (\mathcal{A}, Q, q_\iota, F, \Delta)$ be an SFA. Then $\mathsf{Concretize}_{\mathbb{M}_\mathcal{A}}(\mathcal{M})$ is the DFA $\mathcal{D}_\mathcal{M} = (\Sigma, Q, q_\iota, F, \Delta_\mathcal{D})$ where $\Delta_\mathcal{D}$ is defined as follows. For each state $q \in Q$ let $\pi_q = \langle \psi_1, \ldots, \psi_m \rangle$ be the predicate partition consisting of all predicates labeling a transition exiting q in $\mathcal{M}$. Intuitively, in $\mathcal{D}$, the outgoing transitions of each state q correspond to $\mathsf{Concretize}_\mathcal{A}(\pi_q)$. That is, let $\mathsf{Concretize}_\mathcal{A}(\pi_q) = \langle \Gamma_1, \ldots, \Gamma_m \rangle$. Then, if $\langle q, \psi_i, q' \rangle \in \Delta$, then $\langle q, \gamma, q' \rangle \in \Delta_\mathcal{D}$ for every $\gamma \in \Gamma_i$.
- The formal definition of $\mathsf{Generalize}_{\mathbb{M}_\mathcal{A}}$ is given in Alg.2. Let $\mathcal{D} = (\Sigma, Q, q_\iota, F, \Delta_\mathcal{D})$ be a DFA. We define $\mathsf{Generalize}_{\mathbb{M}_\mathcal{A}}(\mathcal{D})$ wrt. an algebra $\mathcal{A}$ as follows. Let $\mathcal{M} = (\mathcal{A}, Q, q_\iota, F, \Delta_\mathcal{M})$ where $\Delta_\mathcal{M}$ is defined as follows. For each state $q \in Q$ let $\langle \Gamma_1, \ldots, \Gamma_m \rangle$ be the concrete partition consisting of letters labeling outgoing transitions from q. Note that $\langle \Gamma_1, \ldots, \Gamma_m \rangle$ is a concrete partition, since $\mathcal{D}$ is a DFA. Let $\mathsf{Generalize}_\mathcal{A}(\langle \Gamma_1, \ldots, \Gamma_m \rangle) = \langle \psi_1, \ldots, \psi_m \rangle$. Then, $\langle q, \psi_i, q' \rangle \in \Delta_\mathcal{M}$ if Γ_i is the set of letters labeling transitions from q to q' in $\mathcal{D}$.

[5] In the full version of this paper we provide an example illustrating this problem for the class of SFAs over a monotonic algebra $\mathcal{A}_m$, for which respective methods $\mathsf{Concretize}_{\mathcal{A}_m}$ and $\mathsf{Generalize}_{\mathcal{A}_m}$ exist.

We are now ready to define the conditions the *decontaminating* function has to satisfy.

▶ **Definition 11.** *A function* $f_d : 2^{(\mathbb{D}^* \times \{0,1\})} \to 2^{(\mathbb{D}^* \times \{0,1\})}$ *is called* decontaminating *for a class of SFAs* $\mathbb{M}$ *and a respective* $\mathsf{Concretize}_{\mathbb{M}}$ *function if the following holds. Let* $\mathcal{M} \in \mathbb{M}$ *be an SFA, and* $\mathcal{D} = \mathsf{Concretize}_{\mathbb{M}}(\mathcal{M})$*. Let* $\mathcal{S}_{\mathcal{D}} = \textbf{\textit{CharDFA}}(\mathcal{D})$*. Then, for every* $\mathcal{S}' \supseteq \mathcal{S}_{\mathcal{D}}$ *s.t.* $\mathcal{S}'$ *agrees with* $\mathcal{M}$*, it holds that* $\mathcal{S}_{\mathcal{D}} \subseteq f_d(\mathcal{S}') \subseteq (\mathcal{S}' \cap \Gamma_{\mathcal{D}})$*, where* $\Gamma_{\mathcal{D}}$ *is the alphabet of* $\mathcal{S}_{\mathcal{D}}$*.*

As before we say that f_d is *efficient* if it can be computed in polynomial time. We can now provide the sufficient condition.

▶ **Theorem 12.** *Let* $\mathbb{M}_{\mathcal{A}}$ *be a class of SFAs over a Boolean algebra* $\mathcal{A}$*. If there exist efficient functions* $\mathsf{Concretize}_{\mathcal{A}}$ *and* $\mathsf{Generalize}_{\mathcal{A}}$ *satisfying that*

$$\begin{gathered}\textit{if } \mathsf{Concretize}_{\mathcal{A}}(\langle \psi_1, \ldots, \psi_m \rangle) = \langle \Gamma_1, \ldots, \Gamma_m \rangle \\ \textit{and } \mathsf{Generalize}_{\mathcal{A}}(\langle \Gamma'_1, \ldots, \Gamma'_m \rangle) = \langle \varphi_1, \ldots, \varphi_m \rangle \\ \textit{where } \Gamma_i \subseteq \Gamma'_i \textit{ for every } 1 \leq i \leq m \\ \textit{then } [\![\varphi_i]\!] = [\![\psi_i]\!] \textit{ for every } 1 \leq i \leq m\end{gathered}$$

and in addition there exists an efficient decontaminating function $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}}}$*, then the class* $\mathbb{M}_{\mathcal{A}}$ *is efficiently identifiable.*

Given functions $\mathsf{Concretize}_{\mathcal{A}}$, $\mathsf{Generalize}_{\mathcal{A}}$ and $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}}}$ for a class $\mathbb{M}_{\mathcal{A}}$ of SFAs over a Boolean algebra $\mathcal{A}$ meeting the criteria of Thm.12, we show that $\mathbb{M}_{\mathcal{A}}$ can be efficiently identified by providing two algorithms **CharSFA** and **InferSFA**, described bellow. These algorithms make use of the respective algorithms **CharDFA** and **InferDFA** guaranteed in Thm.6.I., as well as the methods provided by the theorem.

We briefly describe these two algorithms, and then turn to prove Thm.12. The algorithm **CharSFA** receives an SFA $\mathcal{M} \in \mathbb{M}$, and returns a characteristic sample for it. It does so by applying $\mathsf{Concretize}_{\mathbb{M}_{\mathcal{A}}}(\mathcal{M})$ (Alg.1) to construct a DFA $\mathcal{D}_{\mathcal{M}}$ and generating the sample $\mathcal{S}_{\mathcal{M}}$ using the algorithm **CharDFA** applied on the DFA $\mathcal{D}_{\mathcal{M}}$.

Algorithm **InferSFA**, given a sample set $\mathcal{S}$, if $\mathcal{S}$ subsumes a characteristic set of an SFA $\mathcal{M}$, returns an equivalent SFA. Otherwise it suffices with returning an SFA that agrees with the sample. First, it applies $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}}}$ to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ over the alphabet of the subsumed characteristic sample, if such a subsumed sample exists. Then it uses $\mathcal{S}'$ to construct a DFA by applying the inference algorithm **InferDFA** on $\mathcal{S}'$. From this DFA it constructs an SFA, $\mathcal{M}_{\mathcal{S}}$, by applying $\mathsf{Generalize}_{\mathbb{M}_{\mathcal{A}}}$ (Alg.2). If the resulting automaton disagrees with the given sample it resorts to returning the prefix-tree automaton. In brief,

- $\textbf{CharSFA}(\mathcal{M}) = \textbf{CharDFA}(\mathsf{Concretize}_{\mathbb{M}_{\mathcal{A}}}(\mathcal{M}))$
- $\textbf{InferSFA}(\mathcal{S}) = \begin{cases} \mathcal{M}_{\mathcal{S}} := \mathsf{Generalize}_{\mathbb{M}_{\mathcal{A}}}(\textbf{InferDFA}(\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}}}(\mathcal{S}))) & \text{if } \mathcal{S} \subseteq \hat{\mathcal{L}}(\mathcal{M}_{\mathcal{S}}) \\ \text{The prefix-tree automaton of } \mathcal{S} & \text{otherwise} \end{cases}$

In §4.3 we provide methods $\mathsf{Concretize}_{\mathcal{A}}$, $\mathsf{Generalize}_{\mathcal{A}}$ and $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}}}$ for SFAs over monotonic algebras, deriving their identification in the limit result. We now prove Thm.12.

Proof of Thm.12. Given functions $\mathsf{Concretize}_{\mathcal{A}}$, $\mathsf{Generalize}_{\mathcal{A}}$, and $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}}}$, we show that the algorithms **CharSFA** and **InferSFA** satisfy the requirements of Def.5.

For the first condition, given that **CharDFA**, $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}}}$ and $\mathsf{Generalize}_{\mathcal{A}}$ run in polynomial time, and that the prefix-tree automaton can be constructed in polynomial time, it is clear that so does **InferSFA**. In addition, the test performed in the definition of **InferSFA** ensures the output agrees with the sample.

For the second condition, note that the sample generated by **CharSFA** is polynomial in the size of $\mathcal{D}_{\mathcal{M}}$, from the correctness of **CharDFA**. In addition, since $\mathsf{Concretize}_{\mathcal{A}}$ is efficient, $\mathcal{D}_{\mathcal{M}}$ is polynomial in the size of $\mathcal{M}$, and thus $\mathcal{S}_{\mathcal{M}}$ generated by **CharSFA** is polynomial in

$\mathcal{M}$ as well. It is left to show that given $\mathcal{S}_\mathcal{M}$ is the concrete sample produced by **CharSFA** when running on an SFA $\mathcal{M}$, then when **InferSFA** runs on any sample $\mathcal{S} \supseteq \mathcal{S}_\mathcal{M}$ it returns an SFA for $\mathcal{L}(\mathcal{M})$. Since $\mathsf{Decontaminate}_{\mathbb{M}_\mathcal{A}}$ is a decontaminating function, and $\mathcal{S} \supseteq \mathcal{S}_\mathcal{M}$, the set $\mathcal{S}' = \mathsf{Decontaminate}_{\mathbb{M}_\mathcal{A}}(\mathcal{S})$ satisfies $\mathcal{S}' \supseteq \mathcal{S}_\mathcal{M}$ and is only over the alphabet $\Gamma_\mathcal{M}$, which is the alphabet of the DFA $\mathcal{D}_\mathcal{M}$ generated in Alg.1.

From the correctness of **InferDFA**, given $\mathcal{S}' \supseteq \mathcal{S}_\mathcal{M}$, applying **InferDFA** on the output $\mathcal{S}'$ of $\mathsf{Decontaminate}_{\mathbb{M}_\mathcal{A}}$ results in a DFA $\mathcal{D}$ that is equivalent to $\mathcal{D}_\mathcal{M}$ constructed in Alg.1. Since $\mathcal{D}_\mathcal{M}$ is complete wrt. its alphabet $\Gamma_\mathcal{M}$, for state q of $\mathcal{D}$, the concrete partition $\langle \Gamma_1, \ldots, \Gamma_n \rangle$ generated in Alg.2 line 4, covers $\Gamma_\mathcal{M}$ and subsumes the output of $\mathsf{Concretize}_{\mathbb{M}_\mathcal{A}}$ on π_q (Alg.1, line 2). Thus, since $\mathsf{Generalize}_\mathcal{A}$ and $\mathsf{Concretize}_\mathcal{A}$ satisfy the criteria of Thm.12, it holds that the constructed predicates agree with the original predicates. In addition, since $\mathcal{S}$, and therefore $\mathcal{S}'$, agrees with $\mathcal{M}$, the test performed in the definition of **InferSFA** fails and the returned SFA is equivalent to $\mathcal{M}$. ◀

4.3 Positive Result

We present the following positive result regarding monotonic algebras.

▶ **Theorem 13.** *Let $\mathbb{M}_{\mathcal{A}_m}$ be the set of SFAs over a monotonic Boolean algebra $\mathcal{A}_m$. Then $\mathbb{M}_{\mathcal{A}_m}$ is efficiently identifiable.*

In order to prove Thm.13, we show that the sufficient condition holds for the case of monotonic algebras. In the full version we provide an example that demonstrates how to apply **CharSFA** and **InferSFA** in order to learn an SFA over the algebra $\mathcal{A}_\mathbb{N}$.

▶ **Proposition 14.** *There exist functions $\mathsf{Concretize}_{\mathcal{A}_m}$ and $\mathsf{Generalize}_{\mathcal{A}_m}$ for a monotonic Boolean algebra $\mathcal{A}_m$, satisfying the criteria of Thm.12.*

Proof. Let $\mathbb{D}$ be the domain of $\mathcal{A}_m$. We provide the functions $\mathsf{Concretize}_{\mathcal{A}_m}$ and $\mathsf{Generalize}_{\mathcal{A}_m}$ and prove that the criteria of Thm.12 hold for them. For ease of presentation, for the function $\mathsf{Concretize}$ we consider basic predicates. Note that for monotonic algebras, basic predicates are in fact intervals, as a conjunction of intervals is an interval. We can assume all predicates are basic since, as we show in [27, Lemma 3], for monotonic algebras the transformation from a general formula to a DNF formula of basic predicates is linear. Then, each basic predicate in the formula corresponds to a different predicate in the predicate partition. The definitions of $\mathsf{Concretize}_{\mathcal{A}_m}$ and $\mathsf{Generalize}_{\mathcal{A}_m}$ are generalizations of the functions $\mathsf{Concretize}_{\mathcal{A}_\mathbb{N}}$ and $\mathsf{Generalize}_{\mathcal{A}_\mathbb{N}}$ given in Ex.9. We define $\mathsf{Concretize}_{\mathcal{A}_m}(\langle \psi_1, \ldots \psi_m \rangle) = \langle \Gamma_1, \ldots, \Gamma_m \rangle$ where we set $\Gamma_i = \{ \min\{d \in \mathbb{D} \mid d \in [\![\psi_i]\!]\} \}$ for $1 \leq i \leq m$. Since $\mathcal{A}_m$ is monotonic, Γ_i is well defined and contains a single element, thus $\mathsf{Concretize}_{\mathcal{A}_m}$ is an efficient concretizing function.

We define $\mathsf{Generalize}_{\mathcal{A}_m}(\langle \Gamma_1, \ldots, \Gamma_m \rangle) = \langle \psi_1, \ldots, \psi_m \rangle$, where ψ_i is defined as follows. Let $\Gamma = \bigcup_{1 \leq i \leq m} \Gamma_i$. First, for all $1 \leq i \leq m$ we set $\psi_i = \bot$. Then, we iteratively look for the minimal element $\gamma \in \Gamma$. Let i be such that $\gamma \in \Gamma_i$, and let γ' be the minimal element in Γ satisfying $\gamma' \notin \Gamma_i$. We then set $\psi_i = \psi_i \vee [\gamma, \gamma')$, and remove all elements $\gamma \leq \gamma'' < \gamma'$ from Γ. We repeat the process until for the found $\gamma \in \Gamma_j$, there is no $\gamma' > \gamma$ such that $\gamma' \notin \Gamma_j$. In that case, we define $\psi_j = \psi_j \vee [\gamma, d_\infty)$. Then, $\Gamma_i \subseteq [\![\psi_i]\!]$ and the predicates are disjoint, thus $\mathsf{Generalize}_{\mathcal{A}_m}$ is an efficient generalizing function.

Now, let $\langle \Gamma_1, \ldots, \Gamma_m \rangle$ be the concrete partition obtained from $\mathsf{Concretize}_{\mathcal{A}_m}$ when applied on the predicate partition $\langle \psi_1, \ldots, \psi_m \rangle$. Assume further that the predicate partition $\langle \Gamma'_1, \ldots, \Gamma'_m \rangle$ satisfies $\Gamma_i \subseteq \Gamma'_i \subseteq [\![\psi_i]\!]$ for $1 \leq i \leq m$. In particular, $\min(\Gamma'_i) = \min(\Gamma_i)$, since Γ_i contains the minimal elements in $[\![\psi_i]\!]$, and $\Gamma_i \subseteq \Gamma'_i \subseteq [\![\psi_i]\!]$. Thus applying $\mathsf{Generalize}_{\mathcal{A}_m}$ will result in the same interval, satisfying the criterion of Thm.12. ◀

Algorithm 3 $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}$ – finding the necessary letters for a characteristic sample.

Input: set $\mathcal{S}$ over alphabet Σ
Output: set $\mathcal{S}'$ over alphabet Σ'

function $\textsc{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}(\mathcal{S})$
  $A_w := \{\epsilon\}$, $\Sigma' := \{d_{inf}\}$, $\sigma_{max} := d_{inf}$
  repeat
    for all $u \in A_w$, by lexicographic order **do**
      for all $\sigma \in \Sigma$, by lexicographic order **do**
        if $\sigma > \sigma_{max}$ and $u\sigma \not\sim_{\mathcal{S}} u\sigma_{max}$ **then**
          if $\forall \sigma'.\ \sigma_{max} < \sigma' < \sigma :\ u\sigma' \sim_{\mathcal{S}} u\sigma_{max}$ **then**
            $\Sigma' := \Sigma' \cup \{\sigma\}$
            if $\forall u' \in A_w.\ u\sigma \not\sim_{\mathcal{S}} u'$ **then** $A_w := A_w \cup \{u\sigma\}$
        $\sigma_{max} := \sigma$
    $\sigma_{max} := d_{inf}$
  until Σ' is remained unchanged
  return $\mathcal{S}' := \mathcal{S} \cap \Sigma'^*$

▶ **Example 15.** Let $\Gamma_1 = \{0, 100, 400, 500\}$ and $\Gamma_2 = \{150, 200\}$ over the algebra $\mathcal{A}_{\mathbb{N}}$ with domain $\mathbb{N} \cup \{\infty\}$. Then, $\mathsf{Generalize}_{\mathcal{A}_{\mathbb{N}}}$ sets $\Gamma = \{0, 100, 150, 200, 400, 500\}$, and finds the minimal element in γ which is 0. Since $0 \in \Gamma_1$, it then looks for the minimal element $\gamma \in \Gamma$ such that $\gamma \notin \Gamma_1$, and finds $150 \in \Gamma_2$. Therefore $\psi_1 = [0, 150)$ and Γ is updated to be $\Gamma = \{150, 200, 400, 500\}$. Next, it finds the minimal element, which is 150 and is in Γ_2, and the minimal element that is not in Γ_2 is 400. Then, ψ_2 is set to be $\psi_2 = [150, 400)$ and $\Gamma = \{400, 500\}$. Last, $\psi_1 = [0, 150) \vee [400, \infty)$ since $400 \in \Gamma_1$ and there is no greater element that is not in Γ_1.

To show that any class of SFAs $\mathbb{M}_{\mathcal{A}_m}$ over a monotonic algebra $\mathcal{A}_m$ is efficiently identifiable, we define in Alg.3 an algorithm that implements a decontaminating function $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}$, fulfilling the requirements of Thm.12. Loosely speaking, the idea of the algorithm is to simultaneously collect elements into two sets A_w and Σ' s.t. A_w will consist of the minimal representative according to the lexicographic order of each equivalence class in $\sim_{\mathcal{S}}$ and Σ' will consist of minimal letters aiding to distinguishing these words. When this process terminates the algorithm returns the subset of words in the sample that consist of only letters in Σ'.

▶ **Lemma 16.** *Assume the input to* $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}$ *is* $\mathcal{S}$ *with* $\mathcal{S} \supseteq \mathcal{S}_{\mathcal{M}}$ *for some* $\mathcal{M} \in \mathbb{M}_{\mathcal{A}_m}$ *s.t.* $\mathcal{S}_{\mathcal{M}} = \boldsymbol{CharDFA}(\mathsf{Concretize}_{\mathbb{M}_{\mathcal{A}_m}}(\mathcal{M}))$*, and* $\mathcal{D}_{\mathcal{M}} = \mathsf{Concretize}_{\mathbb{M}_{\mathcal{A}_m}}(\mathcal{M})$ *is over alphabet* $\Gamma_{\mathcal{M}}$*. Then for* Σ' *constructed by* $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}$ *(Alg.3) it holds that* $\Sigma' = \Gamma_{\mathcal{M}}$*.*

Proof sketch. Let $\mathcal{M} = (\mathcal{A}, Q, q_\iota, F, \Delta_{\mathcal{M}})$, $\mathcal{D}_{\mathcal{M}} = \mathsf{Concretize}_{\mathbb{M}_{\mathcal{A}_m}}(\mathcal{M})$ where $\mathcal{D}_{\mathcal{M}} = (\Gamma_{\mathcal{M}}, Q, q_\iota, F, \Delta_{\mathcal{D}})$, and $\mathcal{S}_{\mathcal{M}} = \mathbf{CharDFA}(\mathcal{D}_{\mathcal{M}})$. We inductively show that for $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}$ given in Alg.3, if its input $\mathcal{S}$ satisfies $\mathcal{S} \supseteq \mathcal{S}_{\mathcal{M}}$ then the set A_w is exactly the set of all lex-access words of states in $\mathcal{D}_{\mathcal{M}}$ and that $\Sigma' = \Gamma_{\mathcal{M}}$ (where $\Gamma_{\mathcal{M}}$ is the alphabet of $\mathcal{D}_{\mathcal{M}}$).

First, we show that every $u \in A_w$ is a lex-access word and that $\Sigma' \subseteq \Gamma_{\mathcal{M}}$. For the base case, we have $A_w = \{\epsilon\}$ and $\Sigma' = \{d_{-\infty}\}$. Since ϵ is the minimal element in the lexicographic order, it holds that $\epsilon \in A_w$ is indeed a lex-access word (of the state q_ι). For $d_{-\infty} \in \Sigma'$, since $\mathsf{Concretize}_{\mathcal{A}_m}$ returns the minimal element of each interval, it holds that $d_{-\infty} \in \Gamma_{\mathcal{M}}$.

For the induction step, assume that A_w contains only lex-access words and that the current Σ' is a subset of $\Gamma_{\mathcal{M}}$. Then, when considering $u \in A_w$ in line 4, it holds that u is a lex-access word of some state q. Then, if σ is added to Σ' it must be a minimal element of some interval labeling an outgoing transition from q, thus it is in $\Gamma_{\mathcal{M}}$, and hence $\Sigma' \subseteq \Gamma_{\mathcal{M}}$. Let $u\sigma$ be a word added to A_w in line 9. Thus, for all $u' \in A_w$ it holds that $u\sigma \not\sim_{\mathcal{S}} u'$.

▷ Claim. In this setting, $u\sigma \not\sim_{\mathcal{S}} u'$ implies $u\sigma \not\sim_{\mathcal{S}_{\mathcal{M}}} u'$.

See the full version for a detailed proof of the lemma, and in particular, a proof of this claim.

Then, for all $u' \in A_w$ we have $\Delta_{\mathcal{D}}(q_\iota, u\sigma) \neq \Delta_{\mathcal{D}}(q_\iota, u')$ where $\Delta_{\mathcal{D}}$ is the transition relation of $\mathcal{D}_{\mathcal{M}}$. Since u is a lex-access word and σ is minimal, $u\sigma$ is a lex-access word for $\Delta_{\mathcal{D}}(q_\iota, u\sigma)$. This concludes the first direction.

For the second direction, we show that every lex-access word is in A_w and that $\Gamma_{\mathcal{M}} \subseteq \Sigma'$. The lex-access word ϵ is in A_w. Let $u\sigma$ be a lex-access word. For all lex-access words u' found in previous iterations it holds that $u\sigma \not\sim_{\mathcal{S}_{\mathcal{M}}} u'$ from item 2 of Thm.6.II, and thus $u\sigma \not\sim_{\mathcal{S}} u'$ since $\mathcal{S}_{\mathcal{M}} \subseteq \mathcal{S}$. Thus, $u\sigma$ satisfies the condition of line 9 in Alg.3 and is added to A_w. For $\Gamma_{\mathcal{M}} \subseteq \Sigma'$, let $\sigma \in \Gamma_{\mathcal{M}}$. From the construction of $\mathsf{Concretize}_{\mathcal{A}_m}$ it holds that σ is the left endpoint of some interval that is an outgoing transition from q_ι. Then, indeed σ is found in the first iteration of line 4. Inductively, since A_w contains all lex-access words, for every state q, the outgoing transitions of q will be considered in some following iteration. Thus, all minimal letters indicating new intervals are added to Σ' and we have that $\Gamma_{\mathcal{M}} \subseteq \Sigma'$. ◀

▶ **Proposition 17.** *The sufficient condition of Thm.12 holds for the class $\mathbb{M}_{\mathcal{A}_m}$ of SFAs over a monotonic Boolean algebra $\mathcal{A}_m$.*

Proof. In Prop.14 we have shown that there exist functions $\mathsf{Concretize}_{\mathcal{A}_m}$ and $\mathsf{Generalize}_{\mathcal{A}_m}$ for a monotonic Boolean algebra $\mathcal{A}_m$, satisfying the criteria of Thm.12. It is left to show that $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}$ is an efficient decontaminating function. Assume that $\mathcal{S} \supseteq \mathcal{S}_{\mathcal{M}}$ where $\mathcal{S}_{\mathcal{M}} = \mathbf{CharDFA}(\mathsf{Concretize}_{\mathbb{M}_{\mathcal{A}_m}}(\mathcal{M}))$, and $\mathsf{Concretize}_{\mathbb{M}_{\mathcal{A}_m}}(\mathcal{M})$ is over alphabet $\Gamma_{\mathcal{M}}$. In Lemma 16 we showed that under these assumptions it holds that the alphabet Σ' of the returned sample $\mathcal{S}'$ is $\Gamma_{\mathcal{M}}$. Then, for the set $\mathcal{S}'$ returned in line 13 (Alg.3) it holds that $\mathcal{S}' = \mathcal{S} \cap \Gamma^*_{\mathcal{M}}$. Since $\mathcal{S} \supseteq \mathcal{S}_{\mathcal{M}}$ and $\Gamma^*_{\mathcal{M}} \supseteq \mathcal{S}_{\mathcal{M}}$, it holds that $\mathcal{S}' \supseteq \mathcal{S}_{\mathcal{M}}$ and $\mathcal{S}'$ is defined over the alphabet $\Gamma_{\mathcal{M}}$. Therefore, $\mathsf{Decontaminate}_{\mathbb{M}_{\mathcal{A}_m}}$ is a decontaminating function. In addition, it runs in time polynomial in the size of $\mathcal{S}$, thus the conditions of Thm.12 are met. ◀

4.4 Negative Result

The result of Thm.13 does not extend to the non-monotonic case, as stated in Thm.18 regarding SFAs over the general propositional algebra. Let $\mathbb{D}_{\mathbb{B}} = \{\mathbb{B}^k\}_{k\in\mathbb{N}}$. Let $\mathbb{P}_{\mathbb{B}} = \{\mathbb{P}_{\mathbb{B}_k}\}_{k\in\mathbb{N}}$ where $\mathbb{P}_{\mathbb{B}_k}$ is the set of predicates over at most k variables. Let $\mathcal{A}_{\mathbb{B}}$ be the Boolean algebra defined over the discrete domain $\mathbb{D}_{\mathbb{B}}$ and the set of predicates $\mathbb{P}_{\mathbb{B}}$, and the usual operators $\vee$, $\wedge$ and $\neg$. Let $\mathbb{M}_{\mathcal{A}_{\mathbb{B}}}$ be the class of SFAs over the Boolean algebra $\mathcal{A}_{\mathbb{B}}$. We show that unless $P = NP$, this class of SFAs is not efficiently identifiable.

▶ **Theorem 18.** *The class $\mathbb{M}_{\mathcal{A}_{\mathbb{B}}}$ is not efficiently identifiable unless $P = NP$.*

Proof. We show that there is no pair of efficient concretizing and generalizing functions $f_c : \Pi_{\mathsf{pred}}(\mathbb{P}_{\mathbb{B}}, 2) \to \Pi_{\mathsf{conc}}(\mathbb{D}_{\mathbb{B}}, 2)$ and $f_g : \Pi_{\mathsf{conc}}(\mathbb{D}_{\mathbb{B}}, 2) \to \Pi_{\mathsf{pred}}(\mathbb{P}_{\mathbb{B}}, 2)$ unless $P = NP$. From Thm.8 it follows that $\mathbb{M}_{\mathbb{B}}$ is not efficiently identifiable unless $P = NP$.

Assume towards contradiction that such a pair of functions exist. We provide a polynomial time algorithm $\mathsf{A}_{\mathsf{SAT}}$ for SAT. On predicate φ, the algorithm $\mathsf{A}_{\mathsf{SAT}}$ invokes $f_c(\langle\varphi, \neg\varphi\rangle)$. Suppose the returned concrete partition is $\langle\Gamma_1, \Gamma_2\rangle$. Then $\mathsf{A}_{\mathsf{SAT}}$ returns "true" if and only

if $\Gamma_1 \neq \emptyset$. Correctness follows from the fact that if there exists a system of characteristic samples for $\mathbb{P}_{\mathbb{B}}$ then the set of positive examples associated with a satisfiable predicate φ must be non-empty, as otherwise f_g cannot distinguish φ from $\bot$. ◀

5 Query Learning

The paradigm of *query learning* stipulates that the *learner* can interact with an *oracle* (*teacher*) by asking it several types of allowed queries. Angluin showed, on the negative side, that regular languages cannot be learned (in the exact model) from only *membership queries* (MQ) [3] or only *equivalence queries* (EQ) [6]. On the positive side, she showed that regular languages, represented as DFAs, can be learned using both MQ and EQ [4]. The celebrated algorithm, termed $\mathbf{L}^*$, was extended to learning many other classes of languages and representations, e.g. [46, 15, 1, 16, 7, 38, 8, 41], see the survey [26] for more references.

In particular, an extension of $\mathbf{L}^*$, termed $\mathbf{MAT}^*$, to learn SFAs was provided in [9] which proved that SFAs over an algebra $\mathcal{A}$ can be efficiently learned using $\mathbf{MAT}^*$ if and only if the underlying algebra is efficiently learnable, and the size of disjunctions of k predicates doesn't grow exponentially in k.[6] From this it was concluded that SFAs over the following underlying algebras are efficiently learnable: Boolean algebras over finite domains, equality algebra, tree automata algebra, and SFAs algebra. Efficient learning of SFAs over a monotonic algebra using MQ and EQ was established in [19], which improved the results of [36, 37] by using a binary search instead of a helpful teacher.

The result of [9] provides means to establish new positive results on learning classes of SFAs using MQ and EQ, but it does not provide means for obtaining negative results for query learning of SFAs using MQ and EQ. We strengthen this result by providing a learnability result that is independent of the use of a specific learning algorithm. In particular, we show that efficient learnability of a Boolean algebra $\mathcal{A}$ using MQ and EQ is a necessary condition for the learnability of the class of SFAs over $\mathcal{A}$, as we state in Thm. 19.

▶ **Theorem 19.** *A non-trivial class of SFAs $\mathbb{M}$ over a Boolean algebra $\mathcal{A}$ is polynomially learnable using MQ and EQ, only if $\mathcal{A}$ is polynomially learnable using MQ and EQ.*

Proof. Assume that $\mathbb{M}$ is polynomially learnable using MQ and EQ, using an algorithm $\mathbf{Q}_{\mathbb{M}}$. We show that there exists a polynomial learning algorithm $\mathbf{Q}_{\mathcal{A}}$ for the algebra $\mathcal{A}$ using MQ and EQ. The algorithm $\mathbf{Q}_{\mathcal{A}}$ uses $\mathbf{Q}_{\mathbb{M}}$ as a subroutine, and behaves as a teacher for $\mathbf{Q}_{\mathbb{M}}$. Whenever $\mathbf{Q}_{\mathbb{M}}$ asks a $\mathbb{M}$-MQ on word $\gamma_1 \dots \gamma_k$, if $k > 1$ then $\mathbf{Q}_{\mathcal{A}}$ answers "no". If $k = 1$ then the $\mathbb{M}$-MQ is essentially an $\mathcal{A}$-MQ, thus $\mathbf{Q}_{\mathcal{A}}$ issues this query and passes the answer to $\mathbf{Q}_{\mathbb{M}}$. Whenever $\mathbf{Q}_{\mathbb{M}}$ asks a $\mathbb{M}$-EQ on SFA $\mathcal{M}$, if $\mathcal{M}$ is of the form $\mathcal{M}_\psi$ for some ψ (as defined in Def.2) then $\mathbf{Q}_{\mathcal{A}}$ answers "no" to the $\mathbb{M}$-EQ and returns some word $w \in \mathcal{L}(\mathcal{M})$ s.t. $|w| > 1$ and w was not provided before, as a counterexample. Otherwise (if the SFA is of the form $\mathcal{M}_\psi$ for some ψ) $\mathbf{Q}_{\mathcal{A}}$ asks an $\mathcal{A}$-EQ on ψ. If the answer is "yes" then $\mathbf{Q}_{\mathcal{A}}$ terminates and returns ψ as the result of the learning algorithm; if the answer to the $\mathcal{A}$-EQ on ψ is "no", then the provided counterexample $\langle \gamma, b_\gamma \rangle$ is passed back to $\mathbf{Q}_{\mathbb{M}}$ together with the answer "no" to the $\mathbb{M}$-EQ. It is easy to verify that $\mathbf{Q}_{\mathcal{A}}$ terminates correctly in polynomial time. ◀

From Thm. 19 we derive what we believe to be the first negative result on learning SFAs from MQ and EQ, as we show that SFAs over the propositional algebra over k variables $\mathcal{A}_{\mathbb{B}_k}$ are not polynomially learnable using MQ and EQ. Polynomiality is measured with respect

[6] As is the case, for instance, in the OBDD (Ordered Binary Decisions Diagrams) algebra [17].

to the parameters $\langle n, m, l \rangle$ representing the size of the SFA and the number k of atomic propositions. Note that the algebra $\mathcal{A}_{\mathbb{B}_k}$ is a restriction of the algebra $\mathcal{A}_{\mathbb{B}}$ considered in §.4.4 and therefore implies a negative result also with regard to the algebra $\mathcal{A}_{\mathbb{B}}$ considered there.

We achieve this by showing that no learning algorithm $\mathbf{A}$ for the propositional algebra using MQ and EQ can do better than asking 2^k MQ/EQ, where k is the number of atomic propositions.[7] We assume the learning algorithm is *sound*, that is, if S_i^+ and S_i^- are the sets of positive and negative examples observed by the algorithm up to stage i, then at stage $i+1$ the algorithm will not ask a MQ for a word in $S_i^+ \cup S_i^-$ or an EQ for an automaton that rejects a word in S_i^+ or accepts a word in S_i^-.

▶ **Proposition 20.** *Let $\boldsymbol{A}$ be a sound learning algorithm for the propositional algebra over $\mathbb{B}^k$. There exists a target predicate ψ of size k, for which $\boldsymbol{A}$ will be forced to ask at least $2^k - 1$ queries (either MQ or EQ).*

Proof. Since $\mathbf{A}$ is sound, at stage $i+1$ we have $S_{i+1}^+ \supseteq S_i^+$ and $S_{i+1}^- \supseteq S_i^-$ and at least one inclusion is strict. Since the size of the concrete alphabet is 2^k, for every round $i < 2^k$, an adversarial teacher can answer both MQ and EQ negatively. In the case of EQ there must be an element in $\mathbb{B}^k \setminus (S_i^- \cup S_i^+)$ with which the provided automaton disagrees. The adversary will return one such element as a counterexample. This forces $\mathbf{A}$ to ask at least 2^k–1 queries. Note that for any element v in $\mathbb{B}^k$ there exists a predicate φ_v of size k such that $[\![\varphi_v]\!] = \{v\}$. ◀

▶ **Corollary 21.** *SFAs over the propositional algebra $\mathcal{A}_{\mathbb{B}_k}$ with k propositions cannot be learned in poly(k) time using MQ and EQ.*

The propositional algebra $\mathcal{A}_{\mathbb{B}_k}$ is a special case of the n-dimensional boxes algebra. Learning n-dimensional boxes was studied using MQ and EQ [29, 18, 12], as well as in the PAC setting [13]. The algorithms presented in [29, 18, 12, 13] are mostly exponential in n. Alternatively, [29, 18] suggest algorithms that are exponential in the number of boxes in the union. In [12] a linear query learning algorithm for unions of disjoint boxes is presented. Since n-dimensional boxes subsume the propositional algebra, Corollary 21 implies the following.

▶ **Corollary 22.** *The class of SFAs over the n-dimensional boxes algebra cannot be learned in poly(n) time using MQ and EQ.*

6 Discussion

We examine the question of learnability of a class of SFAs over certain algebras where the main focus of our study is on passive learning. We provide a necessary condition for identification of SFAs in the limit using polynomial time and data, as well as a necessary condition for efficient learning of SFAs using MQ and EQ. We note that a positive result on learning SFAs using MQ and EQ implies a positive result for identification of SFAs in the limit using polynomial time and data. The latter follows because a systematic set of characteristic samples $\{S_L\}_{L \in \mathbb{L}}$ for a class of languages $\mathbb{L}$ may be obtained by collecting the words observed by the query learner when learning L. However, it does not imply a positive result regarding the stronger notion of *efficient identifiability*, as the latter requires the set to be also constructed efficiently. We thus provide a sufficient condition for *efficient identification* of a class of SFAs, and show that the class of SFAs over any monotonic algebra satisfies these conditions.

[7] In [40] Boolean formulas represented using OBDDs are claimed to be polynomially learnable with MQ and EQ. However, [40] measures the size of an OBDD by its number of nodes, which can be exponential in the number of propositions.

We hope that these sufficient or necessary conditions will help to obtain more positive and negative results for learning of SFAs, and spark an interest in investigating characteristic samples in other automata models used in verification.

References

1 F. Aarts and F. Vaandrager. Learning I/O automata. In *Concurrency Theory, 21th Int. Conf., CONCUR 2010*, pages 71–85, 2010.

2 R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.

3 D. Angluin. A note on the number of queries needed to identify regular languages. *Inf. Control.*, 51(1):76–87, 1981.

4 D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

5 D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

6 D. Angluin. Negative results for equivalence queries. *Mach. Learn.*, 5:121–150, 1990.

7 D. Angluin, S. Eisenstat, and D. Fisman. Learning regular languages via alternating automata. In *Proc. of the Twenty-Fourth Int. Joint Conf. on Artificial Intelligence, IJCAI*, pages 3308–3314, 2015.

8 D. Angluin and D. Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016.

9 G. Argyros and L. D'Antoni. The learnability of symbolic automata. In *Computer Aided Verification – 30th Int. Conf., CAV*, volume 10981 of *LNCS*, pages 427–445. Springer, 2018.

10 G. Argyros, I. Stais, S. Jana, A. D. Keromytis, and A. Kiayias. Sfadiff: Automated evasion attacks and fingerprinting using black-box differential automata learning. In *Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security*, pages 1690–1701. ACM, 2016.

11 G. Argyros, I. Stais, A. Kiayias, and A. D. Keromytis. Back in black: Towards formal, black box analysis of sanitizers and filters. In *IEEE Symposium on Security and Privacy, SP*, pages 91–109, 2016.

12 A. Beimel and E. Kushilevitz. Learning boxes in high dimension. *Algorithmica*, 22(1/2):76–90, 1998.

13 A. Beimel and E. Kushilevitz. Learning unions of high-dimensional boxes over the reals. *Inf. Process. Lett.*, 73(5-6):213–220, 2000.

14 T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen. On the correspondence between conformance testing and regular inference. In *Fundamental Approaches to Software Engineering, 8th Int. Conf., FASE*, volume 3442, pages 175–189. Springer, 2005.

15 F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, 1996.

16 B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In *IJCAI*, pages 1004–1009, 2009.

17 R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

18 N. H. Bshouty, P. W. Goldberg, S. A. Goldman, and H. D. Mathias. Exact learning of discretized geometric concepts. *SIAM J. Comput.*, 28(2):674–699, 1998.

19 K. Chubachi, Diptarama, R. Yoshinaka, and A. Shinohara. Query learning of regular languages over large ordered alphabets. In *1st Workshop on Learning and Automata (LearnAut)*, 2017.

20 K. Chubachi, D. Hendrian, R. Yoshinaka, and A. Shinohara. Query learning algorithm for residual symbolic finite automata. In *Proc. Tenth Int. Symposium on Games, Automata, Logics, and Formal Verification, GandALF*, pages 140–153, 2019.

21 E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2001.

22 L. D'Antoni and M. Veanes. Minimization of symbolic tree automata. In *Proc. of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 873–882. ACM, 2016.

23 L. D'Antoni, M. Veanes, B. Livshits, and D. Molnar. Fast: a transducer-based language for tree manipulation. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI*, pages 384–394. ACM, 2014.

24 C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

25 S. Drews and L. D'Antoni. Learning symbolic automata. In *Tools and Algorithms for the Construction and Analysis of Systems – 23rd Int. Conf., TACAS*, pages 173–189, 2017.

26 D. Fisman. Inferring regular languages and ω-languages. *J. Log. Algebraic Methods Program.*, 98:27–49, 2018.

27 D. Fisman, H. Frenkel, and S. Zilles. On the complexity of symbolic finite-state automata, 2021. `arXiv:2011.05389`.

28 E. M. Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978.

29 P. W. Goldberg, S. A. Goldman, and H. D. Mathias. Learning unions of boxes with membership and equivalence queries. In *Proc. of the Seventh Annual ACM Conf. on Computational Learning Theory, COLT*, pages 198–207. ACM, 1994.

30 S. A. Goldman and H. D. Mathias. Teaching a smarter learner. *J. Comput. Syst. Sci.*, 52(2):255–267, 1996.

31 O. Grinchtein, B. Jonsson, and M. Leucker. Learning of event-recording automata. *Theor. Comput. Sci.*, 411(47):4029–4054, 2010.

32 P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes. Fast and precise sanitizer analysis with BEK. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proc.* USENIX Association, 2011.

33 F. Howar, B. Steffen, and M. Merten. Automata learning with automated alphabet abstraction refinement. In *Verification, Model Checking, and Abstract Interpretation – 12th Int. Conf., VMCAI*, volume 6538 of *LNCS*, pages 263–277. Springer, 2011.

34 Q. Hu and L. D'Antoni. Automatic program inversion using symbolic transducers. In *Proc. of the 38th ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI*, pages 376–389. ACM, 2017.

35 M. Keil and P. Thiemann. Symbolic solving of extended regular expression inequalities. In *34th Int. Conf. on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, pages 175–186, 2014.

36 O. Maler and I. Mens. Learning regular languages over large alphabets. In *Tools and Algorithms for the Construction and Analysis of Systems – 20th Int. Conf., TACAS*, volume 8413 of *LNCS*, pages 485–499. Springer, 2014.

37 O. Maler and I. Mens. A generic algorithm for learning symbolic automata from membership queries. In *Models, Algorithms, Logics and Tools – Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, pages 146–169, 2017.

38 O. Maler and A. Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995.

39 K. Mamouras, M. Raghothaman, R. Alur, Z. G. Ives, and S. Khanna. StreamQRE: modular specification and efficient evaluation of quantitative queries over streaming data. In *Proc. of the 38th ACM SIGPLAN Conf. on Prog. Lang. Design and Impl., PLDI*, pages 693–708, 2017.

40 A. Nakamura. Query learning of bounded-width obdds. *Theor. Comput. Sci.*, 241(1-2):83–114, 2000.

41 D. Nitay, D. Fisman, and M. Ziv-Ukelson. Learning of structurally unambiguous probabilistic grammars. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 9170–9178, 2021. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/17107`.

42 J. Oncina and P. García. Inferring regular languages in polynomial update time. *World Scientific*, January 1992. doi:10.1142/9789812797902_0004.

43 L. Pitt. Inductive inference, DFAs, and computational complexity. In *Proc. Int. Workshop on Analogical and Inductive Inference*, pages 18–44, 1989.

44 M. D. Preda, R. Giacobazzi, A. Lakhotia, and I. Mastroeni. Abstract symbolic automata: Mixed syntactic/semantic similarity analysis of executables. In *Proc. of the 42nd Annual ACM SIGPLAN-SIGACT Symp. on Princ. of Prog. Lang., POPL*, pages 329–341, 2015.

45 O. Saarikivi and M. Veanes. Translating c# to branching symbolic transducers. In *IWIL@LPAR 2017 Workshop and LPAR-21 Short Presentations*, volume 1 of *Kalpa Publications in Computing*. EasyChair, 2017.

46 Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theor. Comput. Sci.*, 76(2-3):223–242, 1990.

47 S. Sheinvald. Learning deterministic variable automata over infinite alphabets. In *Formal Methods – The Next 30 Years – Third World Congress, FM*, volume 11800 of *LNCS*, pages 633–650. Springer, 2019.

48 Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017. doi:10.1145/2967606.

49 M. Veanes, P. de Halleux, and N. Tillmann. Rex: Symbolic regular expression explorer. In *Third Int. Comf, on Software Testing, Verification and Validation, ICST*, pages 498–507. IEEE Computer Society, 2010.

50 X. Zhu, A. Singla, S. Zilles, and A. N. Rafferty. An overview of machine teaching. *CoRR ArXiv*, abs/1801.05927, 2018. arXiv:1801.05927.

Differential Games, Locality, and Model Checking for FO Logic of Graphs

Jakub Gajarský
University of Warsaw, Poland

Maximilian Gorsky
TU Berlin, Germany

Stephan Kreutzer
TU Berlin, Germany

Abstract

We introduce differential games for FO logic of graphs, a variant of Ehrenfeucht-Fraïssé games in which the game is played on only one graph and the moves of both players are restricted. We prove that these games are strong enough to capture essential information about graphs from graph classes which are interpretable in nowhere dense graph classes. This, together with the newly introduced notion of differential locality and the fact that the restriction of possible moves by the players makes it easy to decide the winner of the game in some cases, leads to a new approach to the FO model checking problem which can be used on various graph classes interpretable in classes of sparse graphs.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Fixed parameter tractability

Keywords and phrases FO model checking, locality, Gaifman's theorem, EF games

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.22

Related Version *Full Version*: `https://arxiv.org/pdf/2007.11345.pdf`

Funding *Jakub Gajarský*: The research reported in this paper is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement No. 683080).
Maximilian Gorsky: The research reported in this paper has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).
Stephan Kreutzer: The research reported in this paper has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).

1 Introduction

The first-order (FO) model checking problem asks, given a graph G and a sentence φ as input, whether $G \models \varphi$. It is known that this problem is PSPACE-complete in general [26, 27], but one can obtain efficient parameterised algorithms on many structurally restricted classes of graphs. Here by efficient parameterised algorithms we mean algorithms with runtime $f(|\varphi|) \cdot n^{O(1)}$; these are known as fpt algorithms.

There has been a long line of research studying this problem on sparse graphs and the existence of fpt algorithms was established for graphs of bounded degree [25], graphs with locally bounded treewidth [13], graphs with a locally excluded minor [6], bounded expansion graph classes [7] and nowhere dense graph classes [19]. The positive results on non-sparse graphs fall into two categories. The first category are formed by somewhat isolated results such as [18, 15, 21, 10] and the recent important and general result of [1]. The second

category are positive results about graph classes which can be obtained from sparse graph classes by means of interpretations [16, 17] (although [10] can also be put into this category).

One of the reasons why the research into the FO model checking has been so successful is that Gaifman's theorem [14] – an important result which essentially states that FO logic is local – is particularly useful in the case of sparse graphs. Informally, Gaifman's theorem allows us to reduce the problem of determining whether a given FO formula φ holds on a given graph G to the problem of evaluating a formula $\psi(x)$ in the r-neighbourhood of each vertex of G. In case G is a graph in which each vertex has a simple neighbourhood, one can evaluate φ on G efficiently. This idea leads to efficient algorithms for evaluating FO formulas on classes of graphs of bounded degree, planar graphs, and graphs with locally bounded treewidth.

One shortcoming of using Gaifman's theorem for evaluating FO formulas is that if a graph has an (almost) universal vertex, then for $r \geq 2$ the r-neighbourhood of any vertex is (almost) the whole graph, and therefore evaluating formulas locally on the r-neighbourhoods is essentially the same as evaluating them on the whole graph. Even worse, on complements of bounded degree graphs, it holds for every vertex v that almost the whole graph is in the 1-neighbourhood of v. In such cases, one cannot use the locality-based approach directly, but has to complement the input graph G to get the graph $\bar{G}$ first, use the locality of $\bar{G}$, and then translate the results back to G. In many cases when dealing with non-sparse graphs, there seems to be no good way of using Gaifman's theorem at all, and either one uses a notion of locality tailor-made to the given situation (such as in [15] or [1]) or does not use locality at all (for example the dynamic programming algorithm for FO (and even MSO) logic on graph classes of bounded treewidth).

In this paper we initiate a relativised approach to FO model checking, which is aimed to work on graph classes interpretable in nowhere dense graph classes and which avoids some of the issues mentioned above. Instead of focusing on the absolute question "What is the r-local q-type of a vertex v?" (which is essentially what we do when we apply Gaifman's theorem to obtain algorithmic results), our approach is based on the relative question that, for a pair u, v of vertices, asks "Is the q-type of u and v the same?". It is not difficult to show that being able to answer this question efficiently leads to an efficient algorithm for the FO model checking problem.

The advantage of the relativised approach stems from the fact that one may be able to determine whether q-types of u and v are the same or not without actually determining their q-types themselves – for example if u and v are twins, then they necessarily will have the same q-type, and if one of them has two neighbours and the other has three neighbours, then they cannot have the same q-type (for $q \geq 3$). Moreover, as the examples just given suggest, it is true that if the q-types of u and v are different, then it is possible to exhibit some difference between them in $D(u, v)$ – the symmetric difference of neighbourhoods of u and v. To make this precise, we introduce the *differential game*, a newly defined version of Ehrenfeucht-Fraïssé game, which is played between two vertices u, v of a graph G (the whole game is played on one graph only) and in which the moves of the players are restricted – the first move takes place in $D(u, v)$, the second move in $D(u, v) \cup D(u_1, v_1)$ (here u_1 and v_1 are the vertices played in the first move) and so on. The potential algorithmic advantage offered by our relativized approach comes from the fact that the set $D(u, v)$ can be significantly smaller than sets $N(u)$ and $N(v)$ – for example in complements of graphs of degree at most d, the set $D(u, v)$ has size at most $2d + 2$ while $N(u)$ and $N(v)$ are large.

Our contributions can be briefly summarised as follows:

1. We show that FO model checking can be reduced to deciding whether two vertices u, v of a graph G have the same q-type, i.e. whether $u \equiv_q v$. Note that this is not entirely

trivial – even if we have access to the q-equivalence relation on $V(G)$, it is not clear which equivalence class corresponds to which q-type.

2. We introduce *differential games* which are aimed at distinguishing vertices of different q-types. We prove that for every q there exists r such that the relation $u \cong_r^D v$ defined by "Duplicator wins the r-round differential game between u and v" suitably approximates $\equiv_q$ on graph classes interpretable in nowhere dense graph classes. This leads to the following theorem:

   ▶ **Theorem 6.3.** *Let $\mathcal{C}$ be a class of labelled graphs interpretable in a nowhere dense class of graphs such that we can decide the winner of the r-round differential game in fpt runtime with respect to the parameter r. Then the FO model checking problem is solvable in fpt runtime on $\mathcal{C}$.*

3. As a simple application of our methods, we reprove the result that FO model checking is in FPT for graph classes interpretable in classes of graphs of bounded degree. The important aspect of our new proof is that our algorithm does not involve computing any decomposition of the input graph (in particular, it does not rely on computing a sparse pre-image of the input graph).
4. We define the *differential r-neighborhood* $DN_r(u, v)$ of two vertices, which is an extension of $D(u, v)$ to a "larger radius" and which has the property that the whole r-round differential game played between u and v is played in $DN_r(u, v)$. We use differential neighborhoods to define *differentially simple* graph classes, which are graph classes on which it is possible to decide the winner of the r-round differential game efficiently.

Our results do not immediately lead to new strong FO model checking results – at this point our main contribution is mainly conceptual. We believe that the tools and ideas presented here may lead to new insights and algorithms for graph classes interpretable in classes of sparse graphs. In particular, the idea of performing model checking directly on the input graph, without computing any decomposition or sparsification, seems interesting and probably deserves further attention.

Organization. We give an overview of our ideas in the next section. Section 4 is devoted to relativized model checking, and Sections 5 and 6 are devoted to differential games. Applications are discussed in sections 7 and 8.

2 Overview of our approach

As mentioned in the introduction, our relativised approach to FO model checking is based on determining whether two vertices u, v of G *differ* from each other – i.e. whether $u \not\equiv_q v$, which is the case whenever there is a formula $\psi(x)$ of quantifier rank q such that $G \models \psi(u)$ but $G \not\models \psi(v)$. In Section 4, we show that if we can solve this problem efficiently, then we can solve the FO model checking problem efficiently as well. To be more precise, we show that if we can efficiently compute a relation $\sim_q$ on $V(G)$ such that the transitive closure of $\sim_q$ refines $\equiv_q$ and does not have too many classes, then we can construct an *evaluation tree* of size bounded in terms of q. This evaluation tree then allows us to determine whether $G \models \varphi$ for every sentence φ in prenex normal form with q quantifiers.

From this point on, we focus on efficiently determining whether two vertices u, v differ. Our approach relies on the intuition that if they differ, then there should be a way to exhibit the difference through $D(u, v)$. To capture this intuition formally, we introduce *semi-differential games* in Section 5, which are a variant of the well-known Ehrenfeucht-Fraïssé

(EF) games. The semi-differential game is played in one graph only and if we are given two vertices a_0, b_0 as a starting position, we play the game with the Spoiler's moves being restricted in the following way: In his first move the Spoiler plays a vertex in $D(a_0, b_0)$ and declares the picked vertex to be either a_1 or b_1. The Duplicator picks her reply anywhere in G and the picked vertex becomes b_1 or a_1 – the "opposite" of the Spoiler's choice. More generally, in the i-th move, after the vertices $a_1, \ldots, a_{i-1}$ and $b_1, \ldots, b_{i-1}$ have been played, the Spoiler picks $j \in \{0, \ldots, i-1\}$ and a vertex in $D(a_j, b_j)$ and calls it a_i or b_i. Again, the Duplicator replies by picking a vertex anywhere in G. The winner of the game is decided as in the usual EF game, by comparing the graphs induced by $(a_0, a_1, \ldots, a_m)$ and $(b_0, b_1, \ldots, b_m)$.

As our first crucial result, we show (Lemma 5.2) that there exists a function $l : \mathbb{N} \to \mathbb{N}$ such that if the Spoiler wins the standard r-round Ehrenfeucht-Fraïssé game on a graph G starting from a_0 and b_0, then he wins the $l(r)$-round semi-differential game starting from the same position. This is the aforementioned formalization of the fact that the difference between two vertices can be exhibited through $D(u, v)$. If we denote by $u \cong_r^{SD} v$ the relation "Duplicator wins the r-round semi-differential game starting from u and v", then the above result also tells us that $\cong_{l(q)}^{SD}$ refines $\equiv_q$, and so we can use $\cong_{l(q)}^{SD}$ as the relation $\sim_q$ from the first paragraph of this section (it is not difficult to show that the transitive closure of $\cong_{l(q)}^{SD}$ has bounded number of classes, see Lemma 5.3).

While the relation $\cong_{l(q)}^{SD}$ could serve as a suitable refinement $\sim_q$ of $\equiv_q$, the fact that the Duplicator's moves are not restricted in any way makes it hard to use it algorithmically. To alleviate this, we introduce *differential games* in Section 6 in which the Duplicator's moves are restricted to $D(a_j, b_j)$ as well. For every graph G and every r we define the relation on $V(G)$ by setting $u \cong_r^D v$ if and only if Duplicator wins the r-round differential game starting from u and v. Note that $\cong_r^D$ is a subset of $\cong_r^{SD}$, because whenever Duplicator wins the differential game, she also wins the semi-differential game. This means that the relation $\cong_{l(q)}^D$ also refines $\equiv_q$, as required. Unfortunately, the transitive closure of relation $\cong_r^D$ does not have bounded number of classes, in terms of r, in general. An example of this are ladders – bipartite graphs on vertex set $\{v_1, \ldots, v_{2n}\}$ where the two parts are formed by even and odd numbered vertices and in which there is an edge between v_i and v_j where i is odd and j is even if $i < j$. The reason is that each side in a ladder contains n vertices with nested neighbourhoods, and for any u, v with $N(u) \subsetneq N(v)$ the Spoiler wins the 1-round differential game. There are, however, very rich classes of graphs which exclude arbitrarily large ladders in a very strong sense – such classes of graphs (and more general structures) are known in model theory as *stable* classes of graphs (structures). A prominent example are nowhere dense graph classes introduced by Nešetřil and Ossona de Mendez [22, 23]. On these classes of graphs we can show that the graph of the relation $\cong_m^D$ has a bounded number of connected components. Moreover, we can also show this for graph classes interpretable in nowhere dense graph classes (Theorem 6.2). This leads us to the conclusion that if we can decide the winner of the r-round differential game on any class of graphs which is interpretable in a nowhere dense graph class, then we can perform FO model checking efficiently.

We now turn our attention to classes of graphs on which the winner of the differential game can be found efficiently. First we describe the main idea behind the notion of *differential r-neighbourhood*. In its simplest form it is defined as follows: $DN_1(u, v)$ is just $D(u, v)$ and for any $i > 1$, the differential i-neighbourhood $DN_i(u, v)$ is $DN_{i-1}(u, v)$ together with the union of all $D(a, b)$, with $a, b \in DN_{i-1}(u, v)$. It is easy to see that the whole r-round differential game starting from u, v takes place in $DN_r(u, v)$. If the subgraph of G induced by $DN_r(u, v)$ is simple (say, it has small treewidth), we can decide the winner of differential game efficiently – one can just write a formula $\xi_r(x, y)$ saying "Duplicator wins the r-round differential game

between x and y" and evaluate it on $DN_r(u, v)$ using Courcelle's theorem in our example. This is the essential idea behind *differentially simple* graph classes, in which for all graphs G and all pairs of vertices u, v it holds that $DN_r(u, v)$ induces a subgraph of G which is simple (comes from a class of graphs with efficient FO model checking algorithm). It is easily seen that the "usual suspects" – graphs of bounded degree and of locally bounded treewidth are differentially simple, and so are the classes of complements of these graphs.

We remark that the actual definition of differential neighbourhoods given in Section 8 is slightly more involved and uses colourings. On one hand this makes the notion more general (interpretations of classes of locally bounded treewidth are differentially simple, see Lemma 8.4), on the other hand computing the required colourings may be difficult – we were unable to find a polynomial-time algorithm which computes them. This is similar to the results of [17] and [1], in which the existence of a model checking algorithm is proven, provided that a suitable decomposition of the input graph is given.

3 Preliminaries

We use standard notation from graph theory. All graphs in this paper are finite, undirected, simple, and without loops. The depth of a rooted tree T is the largest number of edges on any leaf-to-root path in T and we say that a node p is at depth i in T if the distance of p from the root of T is i.

By $A \Delta B$ we denote the symmetric difference of two sets A and B defined by $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

3.1 Logic

We assume familiarity with FO logic. We refer to [8] or any standard logic textbook for precise definitions. Since in the paper we only work with finite, simple, undirected graphs, to simplify the exposition we define the notions from logic and model theory for the vocabulary $\sigma = \{E, \{L_a\}_{a \in Lab}\}$ of labelled graphs. Here E is a binary relation symbol, Lab is a finite set of labels and each L_a is a unary predicate symbol.

We say that two graphs G and H are m-equivalent, denoted by $G \equiv_m H$, if they satisfy the same FO sentences of quantifier rank m. It is well known that for every m the relation $\equiv_m$ is an equivalence with finitely many classes.

The FO q-type of a tuple of vertices $\bar{a} = (a_1, \ldots, a_k) \in V(G)^k$, for a given (labelled) graph G, is defined as the set of formulas $\mathrm{tp}_q^G(\bar{a}) := \{\psi(x_1, \ldots, x_k) \in \mathrm{FO}[\sigma] \mid G \models \psi(a_1, \ldots, a_k)$ and ψ has quantifier rank $q\}$, where $\sigma = \{E\}$, or $\sigma = \{E, \{L_a\}_{a \in Lab}\}$, if G is labelled with elements of Lab.

Using the notion of q-types, we can more generally define, for the tuples $\bar{v} := (v_1, \ldots, v_k)$ and $\bar{u} := (u_1, \ldots, u_k)$, consisting of vertices from G, and respectively from H, that $(G, \bar{v}) \equiv_q^k (H, \bar{u})$ if and only if $\mathrm{tp}_q^G(\bar{v}) = \mathrm{tp}_q^H(\bar{u})$. We will mostly be interested in the case when $G = H$; whenever we write $\bar{v} \equiv_q^k \bar{u}$, it is understood that $\bar{v}$ and $\bar{u}$ come from the same graph G, which is clear from the context and should we want to refer to the relation itself and need to note the graph it is based upon, we will add the graph as an index, as in $\equiv_q^{k,G}$. Note that up to equivalence there exist only a finite number of formulas with a given quantifier rank and number of free variables. Therefore there also only exist a finite number of q-types for any given number of free variables. Thus the graph of the relation $\equiv_q^{k,G}$ has a number of components (cliques) bounded by a number depending only on q and k.

For a graph G and a tuple $\bar{v} = (v_1, \ldots, v_k)$ of vertices of G, we define the relation $\equiv_q^{\bar{v}}$ on $V(G)$ by setting $u \equiv_q^{\bar{v}} w$ if and only if $(v_1, \ldots, v_k, u) \equiv_q (v_1, \ldots, v_k, w)$.

3.2 Games

Let G and H be two graphs, and $m \in \mathbb{N}$. The m-round *Ehrenfeucht-Fraïssé game* [11, 12, 9] (or EF game for short), denoted by $\mathcal{G}_m(G, H)$, is played by two players called the *Spoiler* and the *Duplicator*. Each player has to make m moves in the course of play. The players take turns, with the Spoiler going first in each round. In his i-th move, the Spoiler first selects a graph, G or H, and a vertex in this graph. If the Spoiler chooses v_i in G then the Duplicator in her i-th move must choose an element u_i in H. If the Spoiler chooses u_i in H then Duplicator in her i-th move must choose an element v_i in G. The Duplicator wins if $\iota(v_i) = u_i$ is a label preserving isomorphism from $G[\{v_1, \ldots, v_m\}]$ to $H[\{u_1, \ldots, u_m\}]$. Otherwise the Spoiler wins. We say that a player has a *winning strategy*, or in short that they *win* $\mathcal{G}_m(G, H)$, if it is possible for them to win each play regardless of the choices made by their opponent. We denote the fact that the Duplicator wins the m-round EF game between graphs G and H by $G \cong_m H$. The relation $\cong_m$ is an equivalence with finitely many classes for every m. The following theorem connects EF games and m-equivalence.

▶ **Theorem 3.1** (Corollary 2.2.9 in [8]). *Let G and H be graphs and $m \in \mathbb{N}$. Then $G \equiv_m H$ if and only if $G \cong_m H$.*

A *position* in $\mathcal{G}_m(G, H)$ is a tuple $((v_1, \ldots, v_k), (u_1, \ldots, u_{k'}))$, where each v_i is from $V(G)$, each u_i is from $V(H)$, and it holds that $k, k' \leq m$ and $|k - k'| \leq 1$. If $|k - k'| = 0$, then it is the Spoiler's move, otherwise it is the Duplicator's move.

Let G and H be graphs, $(v_1, \ldots, v_k)$ a tuple of vertices of G and $(u_1, \ldots, u_k)$ a tuple of vertices of H. For every m we can play the m-round EF game between $(v_1, \ldots, v_k)$ and $(u_1, \ldots, u_k)$, denoted as $\mathcal{G}_m((G, v_1, \ldots, v_k), (H, u_1, \ldots, u_k))$, by considering the $(k+m)$-round EF game between G and H in which the position $((v_1, \ldots, v_k), (u_1, \ldots, u_k))$ has been reached and starting the play from this position. If the Duplicator wins the m-round game between $(v_1, \ldots, v_k)$ and $(u_1, \ldots, u_k)$, we denote this by $(G, v_1, \ldots, v_k) \cong^k_m (H, u_1, \ldots, u_k)$. The following more general version of Theorem 3.1 connects relations $\equiv^k_m$ and $\cong^k_m$.

▶ **Theorem 3.2** (Theorem 2.2.8 in [8]). *Let G and H be graphs, $(v_1, \ldots, v_k)$ a tuple of vertices of G, $(u_1, \ldots, u_k)$ a tuple of vertices of H, and m a non-negative integer. Then $(G, v_1, \ldots, v_k) \equiv^k_m (H, u_1, \ldots, u_k)$ if and only if $(G, v_1, \ldots, v_k) \cong^k_m (H, u_1, \ldots, u_k)$.*

Again we will be mostly interested in the case when $G = H$; whenever we write $(v_1, \ldots, v_k) \cong^k_m (u_1, \ldots, u_k)$ it is understood that $(v_1, \ldots, v_k)$ and $(u_1, \ldots, u_k)$ come from the same graph G which is clear from the context. When comparing two concrete tuples, we will write $\cong_m$ instead of $\cong^k_m$, since k can be inferred from the context and we will apply the same rationale to $\equiv^k_m$ as well.

3.3 Interpretations

Let $\psi(x, y)$ be an FO formula with two free variables over the language of (possibly labelled) graphs such that for any graph and any u, v it holds that $G \models \psi(u, v) \Leftrightarrow G \models \psi(v, u)$ and $G \not\models \psi(u, u)$, i.e. the relation on $V(G)$ defined by the formula is symmetric and irreflexive. From now on we will assume that formulas with two free variables are symmetric and irreflexive (which can easily be enforced). Given a graph G, the formula $\psi(x, y)$ maps G to a graph $H = I_\psi(G)$ defined by $V(H) = V(G)$ and $E(H) = \{\{u, v\} \mid G \models \psi(u, v)\}$. We then say that the graph H is *interpreted* in G.

In case G is labelled, H inherits labels from G. This way, whenever we need graph H to be labelled, it is enough to consider appropriately labelled G. In case we do not need certain labels from G in H we can simply ignore or drop them when necessary.

The notion of interpretation can be extended to graph classes as well. To a graph class $\mathcal{C}$ the formula $\psi(x,y)$ assigns the graph class $\mathcal{D} = I_\psi(\mathcal{C}) = \{H \mid H = I_\psi(G), G \in \mathcal{C}\}$. We say that a graph class $\mathcal{D}$ is *interpretable* in a graph class $\mathcal{C}$ if there exists formula $\psi(x,y)$ such that $\mathcal{D} \subseteq I_\psi(\mathcal{C})$. Note that we do not require $\mathcal{D} = I_\psi(\mathcal{C})$, as we just want every graph from $\mathcal{D}$ to have a preimage in $\mathcal{C}$.

3.4 Gaifman's theorem

An FO formula $\varphi(x_1,\dots,x_l)$ is *r-local* if for every graph G and all $v_1,\dots,v_l \in V(G)$ it holds that $G \models \varphi(v_1,\dots,v_l) \iff \bigcup_{1\le i\le l} N_r^G(v_i) \models \varphi(v_1,\dots,v_l)$, where $N_r^G(v)$ is the subgraph of G induced by v and all vertices of distance at most r from v.

▶ **Theorem 3.3** (Gaifman's theorem, [14]). *Every first-order formula with free variables $x_1,\dots,x_l$ is equivalent to a Boolean combination of the following:*

- *Local formulas $\phi^{(r)}(x_1,\dots,x_l)$ around $x_1,\dots,x_l$, and*
- *Basic local sentences, i.e. sentences of the form*

$$\exists x_1 \dots \exists x_k \left(\bigwedge_{1\le i<j\le k} dist(x_i,x_j) > 2r \wedge \bigwedge_{1\le i\le k} \phi^{(r)}(x_i) \right)$$

where each $\phi^{(r)}(x_i)$ is a r-local formula

We will need the following simple corollary of Gaifman's theorem, in which we denote by $tp_q^r(v)$ the r-local q-type of v, i.e. the set of all r-local formulas $\psi(x)$ of quantifier rank q such that $G \models \psi(v)$.

▶ **Corollary 3.4.** *For every formula $\psi(x,y)$ there exist numbers r and q such that for every graph G the following holds: If u and v are two vertices of G such that the distance between them is more than $2r$, then whether $G \models \psi(u,v)$ depends only on $tp_q^r(u)$ and $tp_q^r(v)$.*

3.5 Graph classes

We assume familiarity with the notions of treewidth and of clique-width. We will need the following results about the latter concept.

▶ **Theorem 3.5** ([2]). *Let $\mathcal{C}$ be a class of graphs which is interpretable in a graph class of bounded treewidth. Then $\mathcal{C}$ is of bounded clique-width.*

▶ **Theorem 3.6** ([2]). *The FO model checking problem is solvable in fpt runtime on classes of graphs of bounded clique-width.*

We remark that Theorem 3.6 assumes that a clique-width decomposition of the input graph G is provided together with G and it is not known how to efficiently compute an optimal clique-width decomposition. However, one can approximate clique-width using the notion of rankwidth [24], and rankwidth decompositions can be efficiently computed [20].

Nowhere dense graph classes were introduced by Nešetřil and Ossona de Mendez [23]. Instead of working with the original definition, we will be working with an equivalent notion of *uniform quasi-wideness*. Informally, the definition says that we can obtain a large r-scattered set in any sufficiently large set $A \subseteq V(G)$ by removing a few vertices from G.

▶ **Definition 3.7** (Uniform quasi-wideness [4, 5]). *A class $\mathcal{C}$ of graphs is uniformly quasi-wide if for each $r \in \mathbb{N}$ there is a function $N : \mathbb{N} \to \mathbb{N}$ and a constant $s \in \mathbb{N}$ such that for every $k \in \mathbb{N}$, graph $G \in \mathcal{C}$ and subset A of $V(G)$ with $|A| \ge N(k)$, there is a set S of size $|S| \le s$ such that in $A \setminus S$ there are at least k vertices with pairwise distance more than r in $G \setminus S$.*

▶ **Theorem 3.8** ([22]). *A class $\mathcal{C}$ of graphs is uniformly quasi-wide if and only if $\mathcal{C}$ is nowhere dense.*

3.6 Parameterized complexity

We refer to [3] for an indepth introduction to parameterized complexity and only briefly recall the concepts from parameterized complexity needed below. A *parameterized problem* P is essentially a classical problem but in addition to the normal input instance w we are given an integer k, the so-called *parameter*. The problem P is called *fixed-parameter tractable*, or in the complexity class FPT, if there is a computable function f and a constant c such that the problem can be solved by an algorithm whose running time on input (w, k) is bounded by $f(k) \cdot |w|^c$. The class FPT can be seen as the parameterized equivalent to the classical complexity class P as abstraction of efficiently solvable problems.

4 Differential model checking

In this section, we show that in order to efficiently decide whether $G \models \varphi$, it is enough to efficiently solve the following problem: Given a labelled graph G, two of its vertices u and v, and a number q, decide whether there exists a formula $\psi(x)$ with quantifier rank q such that $G \models \psi(u)$ and $G \not\models \psi(u)$, i.e. whether $u \equiv^1_q v$. Moreover, it is enough to compute a relation $\sim$ (not necessarily an equivalence) such that the transitive closure of $\sim$ is a refinement of $\equiv^1_q$ with the number of classes bounded in terms of q.

Due to the space restrictions, we only sketch the idea behind the main result of this section and refer to the full version for details.

First note that, if we can decide for two vertices u, v of G whether $u \equiv^1_q v$ efficiently, then we can compute the whole relation $\equiv^1_q$ on $V(G)$ with quadratic overhead and pick a representative for each class of $\equiv^1_q$. Let $\varphi = Q_1x_1Q_2x_2\ldots, Q_qx_q\psi(x_1, x_2\ldots, x_q)$ be a sentence in prenex normal form and let $v_1, \ldots, v_m$ be representatives of all classes of $\equiv^1_{q-1}$ on $V(G)$. Then the following is true in case $Q_1 = \exists$ we have: $G \models \varphi$ if and only if among $v_1, \ldots, v_m$ there is a v such that $G \models Q_2x_2\ldots, Q_qx_q\psi(v, x_2\ldots, x_q)$. In case $Q_1 = \forall$ the following holds: $G \models \varphi$ if and only if for every v among $v_1, \ldots, v_m$ we have $G \models Q_2x_2\ldots, Q_qx_q\psi(v, x_2\ldots, x_q)$. Thus, instead of going through every $v \in V(G)$ and evaluating $Q_2x_2\ldots, Q_qx_q\psi(v, x_2\ldots, x_q)$ on G (the naive evaluation algorithm), we only need to evaluate $Q_2x_2\ldots, Q_qx_q\psi(v, x_2\ldots, x_q)$ on every vertex v from $v_1, \ldots, v_m$, and m is bounded in terms of q. To evaluate $Q_2x_2\ldots, Q_qx_q\psi(v, x_2\ldots, x_q)$ for any fixed v from $v_1, \ldots, v_m$, we proceed as follows: Mark v in G by label l_1 and all its neighbours by label l'_1 to obtain graph G'. A simple argument shows that we can transform $\psi(x_1, x_2, \ldots, x_q)$ into $\psi'(x_2, \ldots, x_q)$ such that it holds that $G \models Q_2x_2\ldots, Q_qx_q\psi(v, x_2\ldots, x_q)$ if and only if $G' \models Q_2x_2\ldots, Q_qx_q\psi'(x_2\ldots, x_q)$. Thus, we are again left with a problem of evaluating a sentence $\varphi' = Q_2x_2\ldots, Q_qx_q\psi'(x_2\ldots, x_q)$ on G', but this time the sentence has one less quantifier and the graph is labelled. If we can find representatives of all classes of $\equiv'_{q-2}$, where $\equiv'_{q-2}$ is taken over the original vocabulary extended by the two labels l_1 and l'_1, we can continue in this fashion until we eliminate all quantifiers and evaluate the original sentence φ on G.

It is easily seen that the above approach also works if we can just compute a subrelation $\sim_q$ of $\equiv^1_q$, provided that the graph of $\sim_q$ has bounded number of connected components (in terms of q). Finally, if the graph of $\sim_q$ has bounded independence number, one can greedily compute the representatives of classes in the transitive closure of $\sim_q$ with linear overhead instead of quadratic one. This leads to the following corollary.

▶ **Corollary 4.1.** *Let $\mathcal{C}$ be a class of graphs with a function p such that for every t, every $G \in \mathcal{C}$ with at most t labels and every q there is a symmetric and reflexive relation $\sim_{q,t}$ on $V(G)$ such that*

1. *Every class of $\equiv_q^{1,G}$ (over the vocabulary extended with t labels) is a union of connected components of the graph of $\sim_{q,t}$ (in other words the transitive closure of $\sim_{q,t}$ is a refinement of $\equiv_q^{1,G}$),*
2. *The maximum size of any independent set in the graph of $\sim_{q,t}$ is bounded by $p(q,t)$, and*
3. *We can decide whether $u \sim_{q,t} v$ in time $|V(G)|^c \cdot h(q,t)$ for some function h.*

Then one can perform model checking on $\mathcal{C}$ for any sentence φ in prenex normal form, with quantifier rank q, in time $|V(G)|^{c+1} \cdot g(q)$ for some function g.

In Section 6, we show that for graph classes interpretable in nowhere dense graph classes it makes sense to consider as $\sim_q$ the relation "Duplicator wins the $l(q)$-round differential game between vertices u and v of G", for some function l, as this relation satisfies items 1 and 2 above. The situations when it also satisfies item 3 are discussed in sections 7 and 8.

5 Semi-differential EF game

Based on the results from Section 4, to perform model checking efficiently it is enough to be able to determine whether two vertices u and v of a graph G are of the same q-type. To this end we introduce *semi-differential EF games* (this section) and *differential EF games* (next section). The main differences compared to the standard EF game are that the (semi-)differential game is played only on one graph and the moves of the Spoiler (and also the Duplicator in the differential game) are restricted.

For two vertices u, v of a graph G we denote by $D(u,v)$ the symmetric difference of their neighbourhoods, which we will call their *differential neighbourhood*, i.e.

$$D(u,v) := N(u) \Delta N(v).$$

If the graph G in which we want to take the differential neighbourhood is not clear from the context, we will add the relevant graph as an index, as in $D^G(u,v)$.

▶ **Definition 5.1.** *The semi-differential EF game $\mathcal{G}_m^{SD}(G, a_1, \ldots, a_k, b_1, \ldots, b_k)$ with $m \in \mathbb{N}$ rounds is defined in the same way as the standard EF game with the following differences:*

1. *The game is played only on one graph G and $a_1, \ldots, a_k, b_1, \ldots, b_k$ all stem from $V(G)$.*[1]
2. *The starting position is $((a_1, \ldots, a_k), (b_1, \ldots, b_k))$.*
3. *In the j-th round the Spoiler is only allowed to make a move on a vertex $v \in D(a_i, b_i)$ for some $i < j + k$. The Spoiler decides whether this move defines a_j or b_j, i.e. whether the position after his move is $((a_1, \ldots, a_k, v), (b_1, \ldots, b_k))$ or $((a_1, \ldots, a_k), (b_1, \ldots, b_k, v))$. In case no such v exists, the Duplicator wins.*
4. *Duplicator's moves are unrestricted and her reply becomes b_j, if Spoiler decided that the vertex he chose becomes a_j, or a_j, otherwise.*

If $\mathcal{G}_m^{SD}(G, a_1, \ldots, a_k, b_1, \ldots, b_k)$ is won by the Duplicator, we write $a_1, \ldots, a_k \cong_m^{k,SD} b_1, \ldots, b_k$. We apply the same notational conventions to $\cong_m^{k,SD}$ as we did to $\cong_m^k$. If a vertex is played to append the tuple $(a_1, \ldots, a_k)$, we call it an *a-move*, otherwise we call it a *b-move*.

[1] One can also think of the game being played on two copies G_1 and G_2 of graph G, with $a_1, \ldots, a_k \in V(G_1)$ and $b_1, \ldots, b_k \in V(G_2)$. However, we need to be able to refer to $D(a_i, b_i)$, and this is more convenient if both a_i and b_i are in the same graph.

We will from now on refer to the semi-differential EF game as the *semi-differential game*. Let $\cong^{SD}_{m,G}$ denote the relation "Duplicator wins the m round differential game between u and v on the graph G". As usual, we will drop the index G if the graph is clear from the context.

We note that, due to the distinction between a-moves and b-moves, it is also possible that the subgraphs induced by these tuples are not connected. Consider for example a semi-differential game on P_4, with $V(P_4) = [4]$ and $E(P_4) = \{\{i, i+1\} \mid i \in [3]\}$, starting on the position $((a_1 = 1), (b_1 = 4))$. The Spoiler can now play $a_2 = 3$, since $D(a_1, b_1) = \{2, 3\}$, and the graph $P_4[\{a_1, a_2\}]$ is not connected.

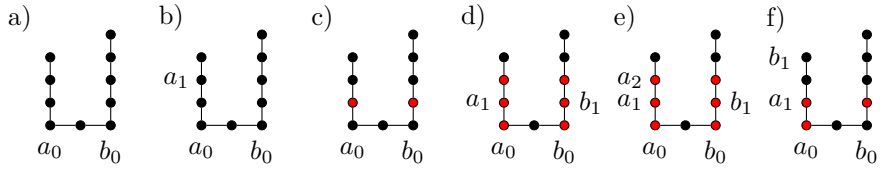


Figure 1 Pictures a) and b): A regular EF game on one graph. The starting position is depicted in a), and the position after Spoiler's first move in b). No matter where the Duplicator replies, she will lose in at most two more moves. Pictures c) - f): The semi-differential EF game. The starting position is depicted in c); all vertices into which the Spoiler can move are marked in red. In this case the Spoiler cannot play the same first move as in the regular EF game. However, he can still play this vertex in two moves (example play in d) and e)) and win from there. The Duplicator can prevent this by playing b_1 as in f) (her moves are unrestricted.), but this loses immediately, since $a_0a_1 \in E$ but $b_0b_1 \notin E$.

Semi-differential games have a direct relation to regular EF games. The rest of the section is devoted to proving that, at the cost of playing more moves, we can play a semi-differential game instead of a regular EF-game to distinguish two vertices. The proof idea is also partially illustrated in Figure 1.

▶ **Lemma 5.2.** *For every $m \in \mathbb{N}$ there exists $l = l(m) \in \mathbb{N}$ such that for every graph G it holds that if $\bar{a} \not\cong_m \bar{b}$, then $\bar{a} \not\cong^{SD}_{l(m)} \bar{b}$.*

Proof. We set $l(0) := 0$ and $l(i+1) := 2l(i) + 1$ and prove the claim by induction on m. For $m = 0$ there is nothing to prove. For the induction step, we assume that the claim holds for m and prove it for $m + 1$. Let $\bar{a} = (a_1, \ldots, a_k)$ and $\bar{b} = (b_1, \ldots, b_k)$ be the starting position. Since by our assumptions the Spoiler has a winning strategy, there exists $v \in V(G)$ such that for every $u \in V(G)$ the Spoiler has a winning strategy in the m-round EF game from position $((\bar{a}, v), (\bar{b}, u))$. In particular, there exists a winning strategy for the Spoiler if $v = u$. By our induction hypothesis, the Spoiler wins the $l(m)$-round semi-differential game starting from $((\bar{a}, v), (\bar{b}, v))$. We fix the Spoiler's winning strategy S for this semi-differential game and apply it to the position $(\bar{a}, \bar{b})$. This is possible because $D(v, v) = \emptyset$. Let $((\bar{a}, a_{k+1}, \ldots, a_{k+l(m)}), (\bar{b}, b_{k+1}, \ldots, b_{k+l(m)}))$ be the position after $l(m)$ rounds. If the subgraphs of G induced by $(\bar{a}, a_{k+1}, \ldots, a_{k+l(m)})$ and $(\bar{a}, b_{k+1}, \ldots, b_{k+l(m)})$ are not isomorphic, then the Spoiler has already won. If they are isomorphic, then it has to hold that $v \in D(a_i, b_i)$, for some $i \in \{k+1, \ldots, k+l(m)\}$, as otherwise the Duplicator's moves would beat the Spoiler's strategy S in the m-round semi-differential game starting from the position $((\bar{a}, v), (\bar{b}, v))$.

Since $v \in D(a_i, b_i)$, for some $i \in \{k+1, \ldots, k+l(m)\}$, the Spoiler can play v in the next round. Let u be the Duplicator's reply. By our assumptions, the Spoiler wins the m-round EF game from the position $((\bar{a}, v), (\bar{b}, u))$. Therefore, according to the induction hypothesis, there exists a winning strategy for the Spoiler in the semi-differential game with $l(m)$ rounds

from the position $((\bar{a}, v), (\bar{b}, u))$; let S' be this strategy. Since we only restrict the Spoiler's moves in the semi-differential game, applying S' to the position $((\bar{a}, a_{k+1}, \ldots, a_{k+l(m)}, v), (\bar{b}, b_{k+1}, \ldots, b_{k+l(m)}, u))$ will not change the outcome and thus the Spoiler wins. ◀

Even though we are not able to establish that the relation $\cong^{SD}_{m,G}$ is an equivalence, we get the following.

▶ **Lemma 5.3.** *For every $m \in \mathbb{N}$ and every graph G, the number of connected components of the graph of the relation $\cong^{SD}_{l(m)}$ is bounded by a function of m, and each equivalence class of $\cong_m$ is a union of connected components of the graph of $\cong^{SD}_{l(m)}$.*

6 Differential game

In the semi-differential game we restrict Spoilers moves to $D(a_i, b_i)$ but the Duplicator's moves are unrestricted. For the application we have in mind, we will restrict Duplicators moves to $D(a_i, b_i)$ as well, using the same i picked by the Spoiler, and call the resulting game the *differential game*. For every graph G, we define the relation $\cong^{D}_{m}$ on $V(G)$ by setting $u \cong^{D}_{m} v$ if and only if Duplicator wins the m-round differential game starting from u and v. We extend this notation to tuples of vertices in the same way as we did with the previous two game definitions.

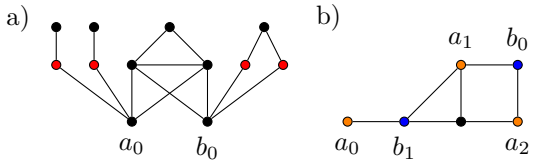


Figure 2 Example positions/plays of the differential game. Notice that in a) from the starting position no vertex of the upper middle triangle can be played by either player. Meanwhile in b), we started from a_0 and b_0, then played a_1, forcing the Duplicator's choice of b_1. Subsequently, the choice of a_2 produces an independent set of size three, which the Duplicator cannot replicate. It can therefore be useful to play a sequence of disconnected vertices even in the differential game.

We summarize the basic properties of differential games used in this section in the following lemma. We refer to the full version for the proof.

▶ **Lemma 6.1** (Properties of differential games). *Let l be the function from Lemma 5.2.*

1. *Every class of $\cong_m$ is a union of connected components of the graph of $\cong^{D}_{l(m)}$.*
2. *Let u and v be two vertices of a graph G such that $u \cong_m v$ and the distance between u and v is more than $2m$. Then the Duplicator wins the m round differential game between u and v.*
3. *For every $m \in \mathbb{N}$ and every graph G, if $\bar{a} \ncong_m \bar{b}$, then $\bar{a} \ncong^{D}_{l(m)} \bar{b}$.*
4. *Let G be a graph and let $\bar{a} := a_1, \ldots, a_k$, $\bar{b} := b_1, \ldots, b_k$ and w be vertices of G such that $\bar{a}w \ncong_m \bar{b}w$. Then the Spoiler has a strategy in $\mathcal{G}^{D}_{l(m)+1}(G, \bar{a}, \bar{b})$ such that he can play w at some point or he wins $\mathcal{G}^{D}_{l(m)+1}(G, \bar{a}, \bar{b})$.*
5. *Let $\psi(x, y)$ be an interpretation formula of quantifier rank q and let G and H be graphs such that $H = I_\psi(G)$. Let a and b be two vertices of G such that $a \cong^{D}_{(m+1)(l(q)+1)} b$ in G. Then $a \cong^{D}_{m} b$ in H.*

We now show that for any first order interpretation $\psi(x,y)$ and any nowhere dense (uniformly quasi-wide) class $\mathcal{C}$ of graphs the number of components of $\cong_m^D$ in any $G \in I_\psi(\mathcal{C})$ is bounded.

▶ **Theorem 6.2.** *Let $\mathcal{C}$ be a uniformly quasi-wide class of labelled graphs and let $\psi(x,y)$ be a first order interpretation formula. Then for each m there exists p such that for every $H \in I_\psi(\mathcal{C})$ the maximum size of any independent set in the graph of $\cong_m^{D,H}$ is at most p.*

Proof. Suppose that there exists an m such that for every p there is a graph $H \in I_\psi(\mathcal{C})$ for which there is an independent set of size more than p in the graph of $\cong_m^{D,H}$.

In what follows we assume that the graph H we work with is as large as necessary and the maximum size of an independent set in the graph of $\cong_m^D$ is as large as we need in our argumentation. We set A to be an independent set of maximum size in the graph of $\cong_m^D$. We therefore have $|A| > p$, and since we can choose p to be arbitrarily large, we can ensure that $|A|$ is as large as we want.

Let q be the quantifier rank of ψ and set $d \coloneqq (m+1)(l(q)+1)$, where l is the function from Lemma 5.2. Since $H \in I_\psi(\mathcal{C})$, there exists at least one $G \in \mathcal{C}$ such that $I_\psi(G) = H$. Because $\mathcal{C}$ is uniformly quasi-wide, we know (by applying the definition to $r = 2d+1$) that there exists a constant s and a function M such that for any number k and any set A of size at least $M(k)$, it is possible to remove s vertices from G such that there are k vertices $v_1, \ldots, v_k$ at pairwise distance at least r in $G \setminus S$. We create the graph G' from $G \setminus S$ by putting the vertices from S back (but without any edges) and labelling them each with a different colour from $[s]$. Additionally, we label the neighbourhood in G of each vertex with colour i with the label l_i. In G' the vertices $v_1, \ldots, v_k$ remain pairwise at distance more than $2m$. We can recover G from G' with a quantifier-free interpretation $\delta(x,y)$. By concatenating δ and ψ, we obtain an interpretation formula $\psi'(x,y)$, with quantifier rank q, such that $H = I_{\psi'}(G')$.

We choose A to be large enough so that among $v_1, \ldots, v_k$ there exists a pair of distinct vertices v_a and v_b with $v_a \cong_d^{G'} v_b$. Note that k only has to be larger than the number of classes of relation $\cong_r^1$ with respect to the vocabulary of $\mathcal{C}$ enriched by $2s$ labels, and so k does not depend on the particular graph G. Since v_a and v_b lie at distance r in G' and $r > 2d$, we can use part 2 of Lemma 6.1 to conclude that $v_a \cong_d^{D,G'} v_b$ is true as well. We can now use part 5 of Lemma 6.1 to conclude that $v_a \cong_m^{D,H} v_b$, which contradicts our assumptions because v_a and v_b come from an independent set of the graph of $\cong_m^{D,H}$. ◀

Combining part 1 of Lemma 6.1 and Theorem 6.2 with Corollary 4.1 we get the following theorem.

▶ **Theorem 6.3.** *Let $\mathcal{C}$ be a class of labelled graphs interpretable in a nowhere dense class of graphs such that we can decide the winner of the m-round differential game in fpt runtime with respect to the parameter m. Then the FO model checking problem is solvable in fpt runtime on $\mathcal{C}$.*

Proof. Let $\mathcal{C}$ be a class of graphs with labels from the set $\{1, \ldots, t\}$ and properties assumed in the statement of the theorem. From part 1 of Lemma 6.1 it follows that for every $G \in \mathcal{C}$ it holds that the closure of $\cong_{l(q)}^{D,G}$ refines $\cong_q^{1,G}$ and by Theorem 3.2 the relation $\cong_q^1$ is the same as $\equiv_q^1$. It follows that the closure of $\cong_{l(q)}^{D,G}$ refines $\equiv_q^{1,G}$. By Theorem 6.2 we know that the maximum size of and independent set in the graph of $\equiv_{l(q)}^{D,G}$ is bounded in terms of $l(q)$. It follows that we can use $\cong_{l(q)}^{D,G}$ as $\sim_{q,t}$ in Corollary 4.1. ◀

7 Applications

Theorem 6.3 from the previous section immediately implies the well-known result that FO model checking can be solved in fpt runtime on classes of graphs of bounded degree. In fact, one can easily use it to establish the (already known) existence of FO model checking algorithms for locally simple graph classes, such as graph classes of locally bounded treewidth – in this case the whole m-round game between any two vertices u, v will take place in $N_m(u) \cup N_m(v)$, and $G[N_m(u) \cup N_m(v)]$ has bounded treewidth, so one can decide the winner there efficiently. The more complicated case of graph classes interpretable in graphs with locally bounded treewidth is analysed in Section 8.

Returning to the case of graphs of bounded degree, the following theorem was proven in [16].

▶ **Theorem 7.1.** *Let $\mathcal{C}$ be a class of graphs of maximum degree d, ψ an interpretation formula and $\mathcal{D} = I_\psi(\mathcal{C})$. Then we can perform FO model checking on $\mathcal{D}$ in fpt runtime.*

The idea behind the proof of Theorem 7.1 in [16] was to compute an approximate "reversal" of the interpretation: Given $H \in \mathcal{D}$, one computes in fpt runtime a graph G of degree d' and formula $\psi'(x, y)$ such that $H = I_{\psi'}(G)$ and such that both d' and ψ' depend only on d and ψ. From this one can then establish Theorem 7.1 by standard considerations.

One can consider a more general version of the approximate interpretation reversal mentioned above. Let $\mathcal{D}$ be a class of graphs interpretable in a nowhere dense class of graphs $\mathcal{C}$ using formula $\psi(x, y)$. The task is to find a nowhere dense class $\mathcal{C}'$ of graphs and a polynomial time algorithm which, given $H \in \mathcal{D}$ as input, computes in a graph $G \in \mathcal{C}'$ and formula $\psi'(x, y)$ such that $H = I_{\psi'}(G)$ and such that ψ' depends only on ψ. A general solution to this problem would imply that there is an fpt FO model checking algorithm for every class of graphs interpretable in a nowhere dense graph class. Unfortunately, the problem of efficiently computing interpretation reversals seems to be currently out of reach, and only small progress has been made so far.

Our techniques suggest that in some cases it may be possible to avoid computing approximate interpretation reversals and efficiently model check formulas directly on the class of graphs interpreted in a class of sparse graphs. To illustrate this, we now reprove Theorem 7.1 in this fashion. We will need the following definition.

▶ **Definition 7.2.** *A pair u, v of vertices of a graph G is* (m, k)-good *if the Duplicator has a winning strategy in the m-round differential game starting from $a_0 := u, b_0 := v$ such that, for any pair a_i, b_i played in the course of the game, it holds that $|D(a_i, b_i)| \leq k$*

Intuitively, the notion of (m, k)-goodness captures the situation when the Duplicator can win the m-round Differential game between u and v in such a way that the "arena" of admissible moves never gets too big. This is useful because one can easily check in fpt runtime whether two vertices are (m, k)-good by playing the differential game by brute-force and whenever for some a_i, b_i it holds that $|D(a_i, b_i)| > k$, declare the current branch as lost for Duplicator.

▶ **Proposition 7.3.** *Given two vertices u, v of a graph G, one can check in time $f(m, k) \cdot |V(G)|^2$ whether they are (m, k)-good.*

With the notion of (m, k)-goodness one can prove Theorem 7.1 easily by showing that the relation $\simeq_{m,k}$ defined by "$u \simeq_{m,k} v$ if and only if u and v are (m, k)-good" can be taken as $\sim_{q,t}$ in Corollary 4.1 for suitable values of m and k (depending on q, t, ψ, d). We refer to the full version for details.

8 Differentially simple graph classes

Based on the results from the previous sections, in order to evaluate FO sentences in prenex normal form on a graph class $\mathcal{C}$ interpretable in a nowhere dense graph class, it is enough to be able to determine the winner of the differential game on pairs of vertices of a graph from $\mathcal{C}$. This can, however, be too difficult – for example consider the case in which $\mathcal{C}$ is an interpretation of planar graphs. In this case for many pairs of vertices u, v of G from $\mathcal{C}$ it can happen that $D(u, v)$ contains most of, or even the entire, graph. We can sidestep this problem by giving such vertices u and v different labels. This essentially means that whenever the Duplicator would play u as a reply to v (or vice versa), she would already have lost from that point on, and so $D(u, v)$ would be irrelevant. Extending these ideas to more than one round leads to the following definitions.

▶ **Definition 8.1** (Differential neighbourhoods). *Let G be a coloured graph and let $c(a)$ denote the colour of a vertex a of G.*

- *The differential 1-neighbourhood $DN_1(u, v)$ of vertices u, v with $c(u) = c(v)$ is the set $D(u, v)$.*
- *For $r \in \mathbb{N}$ with $r > 1$,*

$$DN_r(u, v) := \bigcup_{\substack{a,b \in DN_{r-1}(u,v) \\ c(a)=c(b)}} D(a, b) \cup DN_{r-1}(u, v).$$

- *For $r \in \mathbb{N}$, the* closed *differential r-neighbourhood is defined as $DN_r[u, v] := DN_r(u, v) \cup \{u, v\}$.*

▶ **Definition 8.2.** *We say that class $\mathcal{C}$ is* differentially simple *if for every r there exists $m_r \in \mathbb{N}$ and a graph class $\mathcal{D}_r$ with an efficient FO model checking algorithm such that it is possible to colour every $G \in \mathcal{C}$ with m_r colours such that for every pair u, v of vertices of the same colour it holds that $G[DN_r[u, v]]$ is a graph from $\mathcal{D}_r$.*

Note that if $\mathcal{C}$ is interpretable in a nowhere dense class of graphs, then adding at most m labels to each graph from $\mathcal{C}$ does not change the fact that the maximum size of an independent set in the graph of $\cong_r^D$ for each G from $\mathcal{C}$ is bounded, because Theorem 6.2 works with labelled graphs. We can thus focus on determining the winner of the differential game on $G[DN_r[u, v]]$, which is from $\mathcal{D}_r$, instead of on G which is from $\mathcal{C}$. In case $\mathcal{D}_r$ is a class of graphs with efficient model checking algorithm, we can use this to determine the winner of the game. We will use the FO formula $\xi_r(x, y)$, which expresses that Duplicator wins the r-round differential game between x and y (it is easy to construct such formula for each r), and evaluate it on $G[DN_r[u, v]]$ using the model checking algorithm for $\mathcal{D}_r$. The following theorem summarises this.

▶ **Theorem 8.3.** *Let $\mathcal{C}$ be a differentially simple class of graphs such that*

- *$\mathcal{C}$ is interpretable in a nowhere dense class of graphs, and*
- *There exists an fpt algorithm (with respect to the parameter r) which computes the colouring from Definition 8.2 for every $G \in \mathcal{C}$.*

Then the FO model checking problem is in FPT on $\mathcal{C}$.

To show that differentially simple graph classes can be useful, we prove that classes of graphs interpretable in graph classes with locally bounded treewidth are differentially simple, where each graph class $\mathcal{D}_r$ is a class of graphs of bounded clique-width. We note that in this case there is one m-colouring which works for every value of r and which satisfies the requirements of Definition 8.2.

▶ **Lemma 8.4.** *Let $\mathcal{C}$ be a class of graphs which is interpretable in a class of graphs of locally bounded treewidth. Then $\mathcal{C}$ is differentially simple.*

Proof. Let $\mathcal{E}$ be a class of graphs of locally bounded treewidth and $\psi(x,y)$ an interpretation formula such that $\mathcal{C} = I_\psi(\mathcal{E})$. From Gaifman's theorem applied to $\psi(x,y)$ it follows (by Corollary 3.4) that there exist d and q such that the following holds for any $G \in \mathcal{E}$ and $H \in \mathcal{C}$ such that $H = I_\psi(G)$. If u, v are two vertices of the same d-local q-type in $G \in \mathcal{E}$ and the vertex w is at distance more than $2d$ from both u and v in G, then $G \models \psi(u,w)$ iff $G \models \psi(v,w)$, which in turn means that $uw \in E(H)$ iff $vw \in E(H)$. It follows that if u, v are two vertices of H such that in G these vertices have the same d-local q-types, then every vertex in $D^H(u,v)$ has to be in the $2d$-neighbourhood of u or v in G.

We define the colouring of any $H \in \mathcal{C}$ as follows. Let $G \in \mathcal{E}$ be such that $H = I_\psi(G)$. We colour every vertex v of H by its d-local q-type in G. By the above considerations for any two vertices $u, v \in V(H)$ of the same colour it has to hold that every vertex in $D^H(u,v)$ has to come from $N^G_{2d}(u) \cup N^G_{2d}(v)$. If we consider any two vertices $u', v' \in D^H(u,v)$ of the same colour, the same argumentation applies – every vertex w in $D^H(u',v')$ has to come from $N^G_{2d}(u') \cup N^G_{2d}(v')$ and thus has to be at distance at most $2d$ from u or v in G, which means $w \in N^G_{2d}(u) \cup N^G_{2d}(v)$. It follows by an easy inductive argument that $DN^H_r(u,v)$ in H is a subset of $N^G_{2dr}(u) \cup N^G_{2dr}(v)$ for any positive integer r. Since $\mathcal{E}$ is a class of graphs of locally bounded treewidth, the subgraph of G induced by $N^G_{(r+1)2d}[u] \cup N^G_{(r+1)2d}[v]$ has treewidth bounded in terms of $(r+1)2d$ and an easy argument shows that $H[DN^H_r[u,v]]$ is an induced subgraph of $I_\psi(G[N^G_{(r+1)2d}[u] \cup N^G_{(r+1)2d}[v]])$ which has bounded clique-width. ◀

Lemma 8.4 implies that if we are able to efficiently compute the colourings from Definition 8.2, then we obtain an efficient FO model checking algorithm for classes of graphs interpretable in graph classes of locally bounded treewidth by means of Theorem 8.3. However, the existence of such a colouring algorithm is unknown.

9 Discussion and open problems

We have introduced the notions of differential games and differential locality, which can lead to efficient model checking algorithms and which seem to be more "interpretation friendly" than Gaifman's theorem. We believe that the ideas outlined in this paper can lead to improved understanding of the structure of graphs interpretable in sparse graphs, and perhaps also lead to some insights into stable graphs (if Theorem 6.2 gets strengthened to stable graph classes).

9.1 Complement-simple graph classes

Regarding our application to the model checking problem for graph classes interpretable in classes of graphs with locally bounded treewidth, it has to be noted that there exists a simpler approach based on colourings and on Gaifman's theorem, which avoids differential techniques altogether.

▶ **Definition 9.1.** *We say that a class $\mathcal{C}$ of graphs is* complement-simple *if for every r there exists m_r and graph class $\mathcal{D}_r$ with efficient FO model checking algorithm such that every $G \in \mathcal{C}$ has a m_r-colouring such that complementing edges between some pairs of colours results in a graph G' in which for every $v \in V(G')$ it holds that $N^G_r[v] \in \mathcal{D}_r$.*

If $\mathcal{C}$ is a complement-simple graph class such that we can compute colourings from Definition 9.1 efficiently, then we can perform FO model checking on graphs from $\mathcal{C}$ efficiently. To do this, note that we can interpret G in G' and that we can solve the model checking problem on G' efficiently by using Gaifman's theorem.

Coming back to graph classes interpretable in graph classes with locally bounded treewidth, using the colouring used in the proof of Lemma 8.4 one can show that every such graph class $\mathcal{C}$ is complement-simple and again one can use the same colouring for all values of r.

▶ **Proposition 9.2.** *Let $\mathcal{C}$ be a class of graphs which is interpretable in a class of graphs of locally bounded treewidth. Then $\mathcal{C}$ is complement-simple.*

Proof sketch. We will use the same colouring as was used in the proof of Lemma 8.4 and show that complementing the edges in G between some pairs of colours in this colouring leads to a graph G' with locally bounded clique-width. Let $H \in \mathcal{C}$ and let G be such that $H = I_\psi(G)$ and colour each vertex of H by its d-local q-type, where d and q come from Gaifman's theorem applied to $\psi(x, y)$. We say that an edge uv in $E(H)$ is *long* if $dist_G(u, v) > 2d$. Let $t_1 := tp^d_q(u)$ and $t_2 := tp^d_q(v)$. By Corollary 3.4, if there is a long edge in H between any two vertices of types t_1 and t_2, then there exists an edge between all pairs of vertices of type t_1 and t_2 which are at distance more than $2d$ in G. In this case we say that types t_1 and t_2 *induce long edges*. By complementing the edges between any pair of types (colours) in H which induce long edges we remove all long edges in H and obtain graph H'. It is easily shown that H' is interpretable in G (equipped with colours) by an interpretation which acts only locally and thus H' has locally bounded clique-width. ◀

9.2 Open problems

We conclude with several open problems and possible directions for future research.

1. Is it true that for any stable class $\mathcal{C}$ of graphs and any q there exists p such that every independent set in the graph of the relation $\cong^D_q$ has size at most p?
2. Is it possible to extend the ideas from the proof of Theorem 7.1 from Section 7 to more general graph classes? In particular, is it possible to use an approach based on differential games to design efficient algorithms on interpretations of sparse graphs that do not rely on computing approximate reversals of interpretations?
3. Let $\mathcal{C}$ be a class of graphs interpretable in graph classes of locally bounded treewidth. Is there a polynomial time algorithm that, for every $G \in \mathcal{C}$, computes a colouring such that for every r and every $u, v \in V(G)$ the graph $G[DN_r[u, v]]$ has small clique-width (depending on r)? If not, is there an fpt algorithm which computes such a colouring for every r?
4. Is it possible to use an approach based on differential games to give simpler/different algorithms for the FO model checking problem on graph classes of bounded expansion or nowhere dense graph classes than the algorithms presented in [7] and [19]? If yes, is it possible to use it to extend these results to interpretations of nowhere dense graph classes?
5. More generally, if the answer to Question 1 is yes, is it possible to use our methods to attack the FO model checking problem on stable graph classes?
6. Is there a useful normal form for FO formulas (say, similar to Gaifman normal form) based on differential neighbourhoods and the formulas ξ_r?

References

1 É. Bonnet, E. J. Kim, S. Thomassé, and R. Watrigant. Twin-width i: tractable fo model checking. In *FOCS'20*, pages 601–612. IEEE Computer Society, 2020. doi:10.1109/FOCS46700.2020.00062.

2 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

4 A. Dawar. Finite model theory on tame classes of structures. In *International Symposium on Mathematical Foundations of Computer Science*, pages 2–12. Springer, 2007. doi:10.1007/978-3-540-74456-6_2.

5 A. Dawar. Homomorphism preservation on quasi-wide classes. *Journal of Computer and System Sciences*, 76(5):324–332, 2010. doi:10.1016/j.jcss.2009.10.005.

6 A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *LICS'07*, pages 270–279. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.31.

7 Z. Dvorák, D. Kráľ, and R. Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

8 H. Ebbinghaus and J. Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

9 A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49(129-141):13, 1961.

10 K. Eickmeyer and K. Kawarabayashi. FO model checking on map graphs. In *FCT 2017*, volume 10472 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 2017. doi:10.1007/978-3-662-55751-8_17.

11 R. Fraïssé. Sur une nouvelle classification des systemes de relations. *Comptes Rendus Hebdomadaires Des Seances De L' Academie Des Sciences*, 230(11):1022–1024, 1950.

12 R. Fraïssé. *Sur quelques classifications des systemes de relations*. PhD thesis, Université de Paris, 1955.

13 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

14 H. Gaifman. On local and non-local properties. In *Proceedings of the Herbrand Symposium*, volume 107 of *Stud. Logic Found. Math.*, pages 105–135. Elsevier, 1982.

15 J. Gajarský, P. Hliněný, D. Lokshtanov, J. Obdržálek, S. Ordyniak, M. S. Ramanujan, and S. Saurabh. FO model checking on posets of bounded width. In *FOCS'15*, pages 963–974. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.63.

16 J. Gajarský, P. Hliněný, J. Obdržálek, D. Lokshtanov, and M. S. Ramanujan. A new perspective on FO model checking of dense graph classes. In *LICS '16*, pages 176–184. ACM, 2016. doi:10.1145/2933575.2935314.

17 J. Gajarský, S. Kreutzer, J. Nešetřil, P. Ossona de Mendez, M. Pilipczuk, S. Siebertz, and S. Torunczyk. First-order interpretations of bounded expansion classes. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 126:1–126:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.126.

18 R. Ganian, P. Hliněný, D. Kráľ, J. Obdržálek, J. Schwartz, and J. Teska. FO model checking of interval graphs. *Log. Methods Comput. Sci.*, 11(4:11):1–20, 2015.

19 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

20 P. Hliněný and S. Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

21 P. Hliněný, F. Pokrývka, and B. Roy. FO model checking on geometric graphs. *Comput. Geom.*, 78:1–19, 2019. doi:10.1016/j.comgeo.2018.10.001.

22 J. Nešetřil and P. Ossona de Mendez. First order properties on nowhere dense structures. *J. Symb. Log.*, 75(3):868–887, 2010. doi:10.2178/jsl/1278682204.

23 J. Nešetřil and P. Ossona de Mendez. On nowhere dense graphs. *Eur. J. Comb.*, 32(4):600–617, 2011. doi:10.1016/j.ejc.2011.01.006.

24 S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

25 D. Seese. Linear time computable problems and first-order descriptions. *Math. Structures Comput. Sci.*, 6(6):505–526, 1996.

26 L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

27 M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.

Cyclic Proofs for Transfinite Expressions

Emile Hazard
LIP, ENS Lyon, France

Denis Kuperberg
CNRS, LIP, ENS Lyon, France

Abstract

We introduce a cyclic proof system for proving inclusions of transfinite expressions, describing languages of words of ordinal length. We show that recognising valid cyclic proofs is decidable, that our system is sound and complete, and well-behaved with respect to cuts. Moreover, cyclic proofs can be effectively computed from expressions inclusions. We show how to use this to obtain a PSPACE algorithm for transfinite expression inclusion.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Proof theory; Theory of computation → Logic and verification

Keywords and phrases transfinite expressions, transfinite automata, cyclic proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.23

Related Version *Full Version*: `https://perso.ens-lyon.fr/denis.kuperberg/papers/CSL22long.pdf`

1 Introduction

Language inclusion. Deciding inclusion of regular languages is a fundamental problem in verification. For instance if a program and a specification are modelled by regular languages P and S respectively, the correctness of the program is expressed by the inclusion $P \subseteq S$.

The most standard approach to deciding regular language inclusion is via automata, and this field of research is still active, see for instance [4] for well-performing non-deterministic automata inclusion algorithms using coinduction techniques. Language inclusion is especially important in the framework of infinite words. Indeed, the standard way to model possible behaviours of a system is via ω-regular languages. For instance, Linear Temporal Logic (LTL), which is a practical way to describe some ω-regular languages, is heavily used for expressing specifications. Inclusion of ω-regular languages is still being investigated, with recent works giving refined algorithms [1]. Finally, generalising further, some models of automata and expressions defining languages of transfinite words (i.e. words of ordinal length) were studied in [8, 3]. Transfinite expressions allow any nesting of Kleene star and ω-power. Such expressions define languages of transfinite words, for instance the expression $(a^+b^\omega)^\omega$ describes a language of words of length ω^2. This more general setting of transfinite words can be used for instance to model phenomena with Zeno-type behaviours, such as a ball bouncing at smaller and smaller heights, and after infinitely many bounces it is considered stabilised and can perform some other action.

Proofs systems. The above algorithms give only a yes/no answer, but in some cases the user is interested in having a certificate witnessing inclusion, that he can check independently. This justifies the use of formal proof systems, where proofs can be easily communicated. On finite words, the seminal work [17] gives a complete axiomatic system for regular expression inclusion. Complete axiomatisations for ω-regular expressions were given as well [10].

Cyclic proofs systems. A proof is usually a finite tree with axioms as leaves, built using certain logical rules, and having the conclusion to prove as root. However, under certain conditions, we can consider that infinite trees form valid proofs. Such proofs are called non-well-founded, and can naturally express for instance reasoning by infinite descent. Many proof systems based on non-well-founded proofs were shown to be sound and complete in various frameworks, so these special proofs should be considered as a perfectly valid way of establishing a result. Such proof systems often require a validity condition on their infinite proofs, for instance of the form "on any infinite branch, such a rule must be used infinitely many times". Such a validity condition is often necessary to impose some kind of progress along the branches of the proof, in order to avoid proving false formulas by circular reasoning. These non-well-founded proofs have been studied in several contexts, such as arithmetic [20], first-order logic [5], modal μ-calculus [13, 2, 15], LTL formulas [16], and others. Non-well-founded proofs are especially suited to reason about objects defined via fixed points. Since a non-well-founded proof is a priori an infinite object, it is often relevant to consider the special case of *cyclic* (or *regular*) proofs: those are the proofs obtained as the unfolding of finite graphs, so they are finitely describable.

One of the main advantages of moving to non-well-founded proofs is that in many cases it removes the need to guess invariants (or auxiliary lemmas). See for instance [6], where a cut-free completeness result is proved for a non-well-founded proof system. This makes the system more amenable to proof search: in most non-well-founded systems, we can prove any true formula φ using only formulas that are (in some sense) sub-formulas of φ. In a context where automated proof assistants such as Coq are becoming standard tools, this motivates the current growing interest in cyclic proofs.

Cyclic proofs for regular languages. Here, we aim at exploring the problem of language inclusion in the framework of cyclic proofs. Notice that the Kleene star is a least fixed point operator, and the ω power is a greatest fixed point, so we expect cyclic proofs to be well-suited to deal with regular expressions using these operators.

Das and Pous [12] explored this question in the context of finite words, with standard regular expressions whose only fixed point operator is the Kleene star. They exhibit a cyclic proof system for regular expression inclusion, that they prove sound and complete, even in its cut-free variant. To our knowledge, the cyclic proof approach to inclusion of regular expressions was not explored in the case of infinite and transfinite words, and this is the purpose of the present work.

Contributions. We design a non-well-founded proof system for the inclusion of transfinite expressions. The notion of proof tree is replaced by a proof forest, whose branches can be of ordinal length. We show that our system is sound (in its most general version) and complete (even for cut-free cyclic proofs). We also show that the validity criterion for cyclic proofs is decidable. In the case of infinite words, our system is similar to systems for linear μ-calculus as introduced in [13], except that we use hypersequents as in [12].

The main new difficulty when jumping from finite to infinite or transfinite words is the explosion in the number of non-deterministic choices one is faced with when trying to match a word to an expression. This explains the use of hypersequents, and leads to a slightly more intricate system than [12]. In the transfinite case, the branches of the proof tree become transfinite as well, thereby requiring additional care in the study of the system.

In order to prove the completeness of our system, we show that cyclic proofs can be effectively built from the expressions for which we want to prove inclusion. To show that the resulting proofs are correct, we use a model of automata (close to the one from [8]) recognising these transfinite languages. This allows us to show that cut-free, finitely representable proofs are enough to prove any true inclusion, and that these proofs can be computed.

The cyclic cut-free completeness of our system allows us to obtain a PSPACE algorithm for inclusion of transfinite expressions. This matches the known lower bound: inclusion of regular expressions is PSPACE-hard already for finite words. PSPACE membership is folklore for inclusion of ω-regular expressions as well, but to our knowledge, this upper bound is a new result for transfinite expressions. Let us note however that since automata models were already defined for transfinite expressions [8, 3], it is plausible, that a PSPACE algorithm can also be obtained more directly through these models. This PSPACE-completeness result can be compared with the result from [14], stating PSPACE-completeness of LTL satisfiability on transfinite words.

Related works. In addition to related works that were already mentioned, let us comment on the link between our results and the recent paper [9], which studies cyclic proofs for first-order logic extended with least and greatest fixed points. The validity criterion in [9] is very similar to ours, and as they note, their general framework allows to embed reasonings on infinite words as a special case. One advantage of our system for ω-regular expressions is that although it is less general, it is much more convenient to manipulate ω-regular languages. Moreover, the use of hypersequents allows us to obtain cut-free regular completeness, which is not the case in [9]. On the other hand, our work on transfinite expressions is orthogonal to [9], as in such expressions, the ω operator is no longer a greatest fixed point.

Outline. We will start by describing the system for infinite words in Section 2, and first prove our results in this restricted case. We then show in Section 3 how the system can be modified to accommodate transfinite words, and how the results can be lifted to this setting.

2 The case of ω-regular expressions

In this part, we do not yet look at truly transfinite expressions such as $(a^+b^\omega)^\omega$, but only at ω-regular ones, which are the ones describing languages of words of length at most ω.More formally, these expressions can be described by the following grammar.

- Regular expressions: $e, f ::= a \mid e + f \mid e \cdot f \mid e^+$
- ω-regular expressions: $g, h ::= e \mid e^\omega \mid e \cdot g \mid g + h$, where e ranges over regular expressions.

To associate a language $\mathcal{L}(g)$ of finite or infinite words to an ω-regular expression g, it suffices to interpret each constructor on languages in the standard way:

$\mathcal{L}(0) = \emptyset$	$L(e + f) = \mathcal{L}(e) \cup \mathcal{L}(f)$	$\mathcal{L}(e \cdot f) = \mathcal{L}(e) \cdot \mathcal{L}(f) = \{uv \mid u \in \mathcal{L}(e), v \in \mathcal{L}(f)\}$
$L(a) = \{a\}$	$\mathcal{L}(e^+) = \bigcup_{n>0} \mathcal{L}(e)^n$	$\mathcal{L}(e^\omega) = \mathcal{L}(e)^\omega = \{u_1 u_2 \cdots \mid \forall i, u_i \in \mathcal{L}(e)\}$

We avoid the use of ε, and we use e^+ instead of e^*, to guarantee that an expression e^ω only accepts infinite words.

We design a proof system S_ω that will provide a certificate for any inclusion between the languages of two such expressions. Starting with the special case of ω-regular expressions allows us to introduce most proof techniques, while staying in a more familiar framework. We also claim that already in this case, such a proof system can bring new insights, as it can offer interesting trade-offs compared to automata models (see Conclusion).

2.1 The proof system S_ω

The proof system described in this section is strongly inspired from [11], the novelty being the introduction of ω.

2.1.1 Rules for building preproofs

We will first describe the sequents of the system S_ω, i.e. the shape of any label of a node in a proof tree. These are identical to the ones we use later, in the proof system for generalised expressions.

▶ **Definition 1** (Sequent). *We call* sequent *a pair* (Γ, B)*, noted* $\Gamma \to B$*, where* Γ *is a list of expressions and* B *is a nonempty finite set of such lists. In the rest of the paper, upper case Greek letters will be used for lists of expressions, and upper case Latin letters for sets of lists.* Γ *will be called the* left side *of the sequent and* B *its* right side*. Their contents will be denoted as follows, with brackets isolating each list in* B*:*

$$\Gamma = e_1, \dots, e_n \qquad\qquad B = \langle f_1^1, \dots, f_1^{k_1}\rangle; \dots; \langle f_m^1, \dots, f_m^{k_m}\rangle$$

Languages are associated to such lists and sets of lists in the following way:

$$\mathcal{L}(\Gamma) = \mathcal{L}(e_1 \cdot \ldots \cdot e_n) \qquad\qquad \mathcal{L}(B) = \mathcal{L}(f_1^1 \cdot \ldots \cdot f_1^{k_1} + \ldots + f_m^1 \cdot \ldots \cdot f_m^{k_m})$$

The sequent $\Gamma \to B$ *is called* sound *if the inclusion* $\mathcal{L}(\Gamma) \subseteq \mathcal{L}(B)$ *holds.*

To describe our proof system, we now need to define the notion of proof tree. These are usually finite objects, but in our setting we allow infinite trees.

A *tree* is a non-empty, prefix-closed subset of $\{0,1\}^*$. We typically represent it with the root ε at the bottom, and the sons $v0$ and $v1$ of a node v (if they exist) are represented above v, respectively on the left and on the right.

A *branch* of a tree $T \subseteq \{0,1\}^*$ is a prefix-closed subset of T that do not contain two words of the same length, i.e. two nodes at the same depth of the tree. A branch of T is *maximal* if it is not strictly contained in another branch of T.

A *preproof* is given by a tree and a labelling π of its nodes by sequents in such a way that for any node v with children $v_1, \dots, v_n$ (with $n \in \{0,1,2\}$), the expression $\dfrac{\pi(v_1) \quad \cdots \quad \pi(v_n)}{\pi(v)}$ is an instance of a rule from Figure 1.

A preproof is called *cyclic* or *regular* if it has finitely many distinct subtrees. Such a proof can be represented using a finite tree, where each leaf x not closed with an id rule is equipped with a pointer to a node y below x, indicating that the infinite trees rooted in x and y are identical. Examples of this representation can be found in Figure 2.

2.1.2 Threads and validity condition

Some preproofs satisfying the conditions described above actually prove wrong inclusions, meaning that we can build such a tree with an unsound sequent at its root. An example of such a preproof can be found in Figure 2. This illustrates the need for a validity condition that will rule out such unsound preproofs. We need a few more definitions before we can state this validity condition.

Occurrences. When talking about "expression" in a preproof, we will actually be talking about particular occurrences of an expression in the preproof, see the long version for details on this. If S is a sequent, we will note $pos(S)$ the set of expression positions in S.

$$\frac{}{\to \langle\rangle}\ \text{id} \qquad \frac{e, \Gamma \to B \quad f, \Gamma \to B}{e + f, \Gamma \to B}\ +\text{-l} \qquad \frac{\Gamma \to \langle e, \Lambda\rangle; \langle f, \Lambda\rangle; B}{\Gamma \to \langle e + f, \Lambda\rangle; B}\ +\text{-r}$$

$$\frac{\Gamma \to B}{\Gamma \to B; C}\ \text{wkn} \qquad \frac{\Gamma, e, f, \Lambda \to B}{\Gamma, e \cdot f, \Lambda \to B}\ \cdot\text{-l} \qquad \frac{\Gamma \to \langle \Lambda, e, f, \Theta\rangle; B}{\Gamma \to \langle \Lambda, e \cdot f, \Theta\rangle; B}\ \cdot\text{-r}$$

$$\frac{\Lambda \to \langle \Theta_1\rangle; \ldots; \langle \Theta_n\rangle}{\Gamma, \Lambda \to \langle \Gamma, \Theta_1\rangle; \ldots; \langle \Gamma, \Theta_n\rangle}\ \text{match} \qquad \frac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}\ *\text{-l} \qquad \frac{\Gamma \to \langle e, \Lambda\rangle; \langle e, e^+, \Lambda\rangle; B}{\Gamma \to \langle e^+, \Lambda\rangle; B}\ *\text{-r}$$

$$\left(\frac{\Lambda \to \langle e\rangle \quad \Gamma, e, \Theta \to B}{\Gamma, \Lambda, \Theta \to B}\ \text{cut}\right) \qquad \frac{e, e^\omega \to B}{e^\omega \to B}\ \omega\text{-l} \qquad \frac{\Gamma \to \langle e, e^\omega\rangle; B}{\Gamma \to \langle e^\omega\rangle; B}\ \omega\text{-r}$$

Figure 1 The rules of the system S_ω for ω-regular expressions.
Γ, Λ, Θ are lists of expressions; B, C are sets of such lists; e, f are ω-regular expressions.
Rules wkn, match, cut will sometimes be abbreviated w, m, c.

Principal expression. In a sequent of a preproof where a rule r is applied, an expression is called *principal* for r if it is the one corresponding to the lower case expression in the lower side of the rule r from Figure 1. Note that there is no principal expression when the rule is id, wkn, cut or match, since these rules do not contain lower case letters in the lower sequent.

Ancestors. Given an expression e in the lower part of a rule, its *immediate ancestors* are:
- if e is principal: the lower case expressions in the upper sequents of the rule
- if e is in a list Γ or a set of list B: its copies in the same position in each copy of Γ (resp. B) on the upper sequents.

Note that an expression can have between 0 (expression in C in the wkn rule) and 3 (e^+ in any $*$ rule) immediate ancestors.

Threads. A *thread* is a path in the graph of immediate ancestry (also called the logical flow graph [7]). We say that a thread witnesses a v-unfolding if the current expression is principal for either a $*$-l rule or an ω-r rule. As in Figure 2, threads will be represented by colored lines, with bullets to mark v-unfoldings.

Note that we purposely talk about the "graph" of immediate ancestry, and not the "tree". Since the right part of a sequent is a set, it does not keep track of multiplicity, and two threads can merge when going upwards. For instance, if we apply the rule $\dfrac{\Gamma \to \langle e, e^\omega\rangle}{\Gamma \to \langle e^\omega\rangle; \langle e, e^\omega\rangle}\ \omega\text{-r}$, the red and blue threads are merged. We need to allow that phenomenon in order to be able to build finitely representable proofs.

We can now define the *validity condition*, that makes a preproof into an actual proof.

▶ **Definition 2** (Validity condition). *A thread is* validating *if it witnesses infinitely many v-unfoldings. A preproof is* valid*, and is then called a* proof*, if all its infinite branches contain a validating thread.*

We will call *$*$-l thread* (resp. *ω-r thread*) a validating thread on the left side (resp. right side) of sequents, as it witnesses infinitely many $*$-l (resp. ω-r) rules.

Let us give an intuition for this validity condition. A proof has to guarantee that any word generated by the left side expression can be parsed in the right side one. Branches with a $*$-l thread do not correspond to a word on the left side, so there is nothing to verify and the branch can be accepted. On the other hand, when a legitimate infinite word from the left side has to be parsed on the right side, it must involve an expression e^ω where e is matched to infinitely many factors. This corresponds to an ω-r thread.

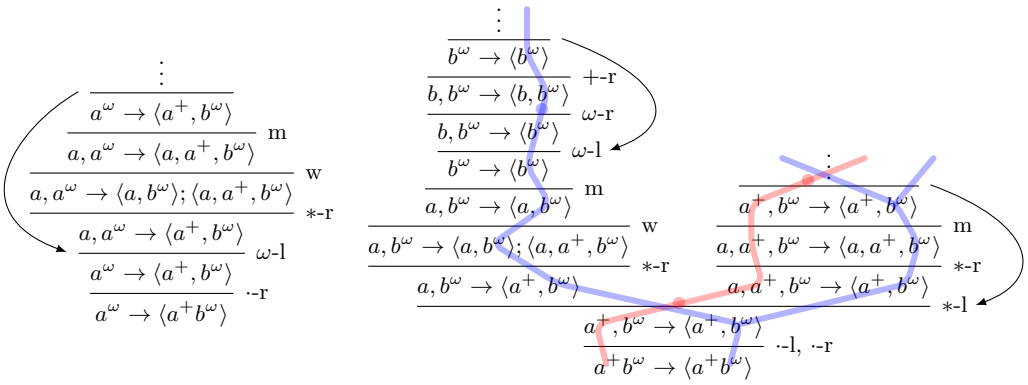


Figure 2 An invalid preproof (left) and a valid one (right).

We give two examples of preproofs in Figure 2. The left one is an invalid preproof of a wrong inclusion. The validity condition is not satisfied, since there are no $*$-l or ω-r rules.

The right one is an actual proof. It is comb-shaped, with a "main" branch always going to the right. We can get a validating thread for any branch of that preproof, by taking the red thread on the rightmost branch, and a blue thread on all other branches.

For an example of a non-trivial inclusion, see the long version for a cyclic proof showing that $\mathcal{L}((a+b)^\omega) \subseteq \mathcal{L}((b^*a)^\omega + (a+b)^*b^\omega)$, i.e. any infinite word on alphabet $\{a, b\}$ has either infinitely or finitely many a's.

▶ **Definition 3** (Soundness and completeness). *A proof system is sound if the conclusions of all of its valid proofs are sound. It is complete if for any sound sequent $\Gamma \to B$, there is a proof with conclusion $\Gamma \to B$.*

2.2 Soundness of the system S_ω

In this part, we want to prove that any proof (i.e. any valid preproof) derives a sound sequent. We will do that without any assumption of regularity, since we want every proof from the S_ω system to be correct, and not just the regular fragment. We will show soundness of the system with cuts, since this is more general and it allows to write proofs more conveniently. Notice that incorporating the cuts significantly increases the difficulty: unlike what happens in a finitary proof system, it is not enough here to prove that the cut rule is locally sound. Since the cuts can be used infinitely many times along a branch, it calls for a careful argument. Missing details and proofs can be found in the long version. The following first result is an easy consequence of the local soundness of our rules:

▶ **Lemma 4.** *Any finite preproof derives a sound sequent.*

To prove the general case, we take any valid proof tree P in S_ω, with a root sequent $\Gamma_0 \to B_0$. We take an arbitrary $w \in \mathcal{L}(\Gamma_0)$, and we show that $w \in \mathcal{L}(B_0)$.

We create a tree $P(w)$ that will be a subtree of the original one, with additional information labelling its nodes. The purpose of the tree $P(w)$ is to prove the membership of w in $\mathcal{L}(B_0)$.

The sequents of $P(w)$ are similar to the ones of a preproof, but we additionally label each expression e on the left side of sequents with a word u. Given a list of expressions Γ, we will denote Γ' a labelling of its expressions with words, represented as a list of pairs (expression,

word). If $\Gamma' = (e_1, u_1), \dots, (e_k, u_k)$, we say that (the labelling of) Γ' is *correct* if for each $i \in [1, k]$, we have $u_i \in \mathcal{L}(e_i)$. We define $\mathrm{concat}(\Gamma')$ as the word $u_1 \dots u_k$. We additionally say that a sequent $\Gamma' \to B$ is *label-sound* if $\mathrm{concat}(\Gamma') \in \mathcal{L}(B)$.

We will build $P(w)$ by transforming the initial proof by induction from the root. First we take a correct labelling Γ'_0 of Γ_0 such that $\mathrm{concat}(\Gamma'_0) = w$. Then we move upwards while replacing each rule by the corresponding one in the table below, while satisfying the condition specified in the table (if possible). This tree will be a subtree of the initial one since we only keep one successor at rules $+$-l and $*$-l.

Rule	New rule	Condition
$\dfrac{\Gamma, e, f, \Lambda \to B}{\Gamma, e \cdot f, \Lambda \to B}$ $\cdot$-l	$\dfrac{\Gamma', (e, u), (f, v), \Lambda' \to B}{\Gamma', (e \cdot f, uv), \Lambda' \to B}$ $\cdot$-l	$(u, v) \in \mathcal{L}(e) \times \mathcal{L}(f)$
$\dfrac{\Gamma, e_1, \Lambda \to B \quad \Gamma, e_2, \Lambda \to B}{\Gamma, e_1 + e_2, \Lambda \to B}$ $+$-l	$\dfrac{\Gamma', (e_i, u), \Lambda' \to B}{\Gamma', (e_1 + e_2, u), \Lambda' \to B}$ $+$-l	$u \in \mathcal{L}(e_i)$
$\dfrac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}$ $*$-l	$\dfrac{(e, u), \Gamma \to B}{(e^+, u), \Gamma \to B}$ $*$-l	$u \in \mathcal{L}(e)$
$\dfrac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}$ $*$-l	$\dfrac{(e, u), (e^+, v), \Gamma' \to B}{(e^+, uv), \Gamma' \to B}$ $*$-l	$(u, v) \in \mathcal{L}(e) \times \mathcal{L}(e^+)$
$\dfrac{\Gamma, e, e^\omega \to B}{\Gamma, e^\omega \to B}$ ω-l	$\dfrac{\Gamma', (e, u), (e^\omega, v) \to B}{\Gamma', (e^\omega, uv) \to B}$ $*$-l	$(u, v) \in \mathcal{L}(e) \times \mathcal{L}(e^\omega)$
$\dfrac{\Lambda \to \langle e \rangle \quad \Gamma, e, \Theta \to B}{\Gamma, \Lambda, \Theta \to B}$ c	$\dfrac{\Lambda' \to \langle e \rangle \quad \Gamma', (e, u), \Theta' \to B}{\Gamma', \Lambda', \Theta' \to B}$ c	$u = \mathrm{concat}(\Lambda')$

Notice that if the labelling of the bottom sequent is correct, this guarantees us that we can choose a correct labelling for the upper sequent as well, while satisfying the condition in the table. It is only because of the cut rule that we cannot simply propagate correctness of labellings from the root. Moreover, all these rules are label-sound, in the sense that their lower sequents are label-sound whenever the upper ones are.

If at some point the condition cannot be met, we stop there and call the current node a dead leaf. This is only important for the sake of a complete definition, since we will prove that this actually never occurs if the initial preproof is valid (Lemma 8).

We deal with the remaining rules (right rules, wkn and match) by simply copying the pairs (expression, word) from bottom to top on the left side.

▶ **Lemma 5.** *In $P(w)$, any infinite branch has an ω-r thread.*

Proof. This follows from the fact that any expression e^+ on the left of a sequent in $P(w)$ is associated with a finite word, and therefore can only be principal for finitely many $*$-l rules, each one decreasing the size of that word (notice that we use an infinite descent argument here). The validity condition then ensures the result. ◀

▶ **Lemma 6.** *In $P(w)$, no branch goes infinitely many times to the left at cut rules.*

Proof. Let us consider an infinite branch β of $P(w)$. By Lemma 5, the branch β has an ω-r thread. Since the right side of a sequent is not preserved when going to the left at a cut rule, it can only happen finitely many times in β. ◀

If β_1, β_2 are maximal branches, we note $\beta_1 \leq \beta_2$ if β_1 is to the left of β_2. The following Lemma is proved in the long version.

▶ **Lemma 7.** *The maximal branches of $P(w)$ are well-ordered by $\leq$.*

The main Lemma for the soundness proof is Lemma 8, which also ensures that there is no dead leaf in $P(w)$.

▶ **Lemma 8.** *In $P(w)$, all labellings are correct and all sequents are label-sound.*

Proof. (Sketch) We just give the main idea here, a detailed proof can be found in the long version. We proceed by well-founded induction on the maximal branches, using the left-to-right order from Lemma 7. The only interesting case it when dealing with cuts.

When encountering a cut rule of the form $\dfrac{\Lambda' \to \langle e \rangle \qquad \Gamma', (e,u), \Theta' \to B}{\Gamma', \Lambda', \Theta' \to B}$ cut , we need to show that the label (e,u) is correct, i.e. provided $u \in \mathcal{L}(\Lambda')$, we have $u \in \mathcal{L}(e)$. This will be obtained thanks to the induction hypothesis, guaranteeing that the sequent $\Lambda' \to \langle e \rangle$ on the left of the cut rule is label-sound. ◀

▶ **Theorem 9.** *Any valid proof from S_ω is sound.*

Proof. For any word w in $\mathcal{L}(\Gamma_0)$, there is a way to correctly label Γ_0 into a Γ'_0 with $\mathrm{concat}(\Gamma'_0) = w$. By Lemma 8, this correct labelling can be propagated through $P(w)$, and we obtain that the root sequent $\Gamma'_0 \to B_0$ is label-sound, so $w \in \mathcal{L}(B_0)$. Thus we have indeed $\mathcal{L}(\Gamma_0) \subseteq \mathcal{L}(B_0)$, showing that any valid proof is sound. ◀

▶ Remark 10. This result is similar to the soundness in [13], but the shape of the sequents differs, and the transfinite case that follows uses an extension of our proof.

2.3 Cut-free regular completeness of the system S_ω

In order to prove the completeness of the system, we want to show that any sound sequent can be derived. Moreover, if we want this system to be interesting from a computational point of view, we need to obtain finitely representable proofs. The following lemma will help us do that by ensuring a finite number of different sequents in a proof. Then we will build a regular proof via a deterministic saturation process.

▶ **Lemma 11.** *In a preproof without cut, there can only be finitely many different sequents.*

Proof. Let us call $\mathrm{expr}(\Gamma)$ the expression formed by concatenating the expressions in Γ, and similarly $\mathrm{expr}(B)$ is the expression $\bigcup_{\Gamma \in B} \mathrm{expr}(\Gamma)$. We can verify that for every rule except cut, if $\Gamma_0 \to B_0$ is the conclusion sequent and if $\Gamma_1 \to B_1$ is a premise sequent, then $\mathrm{expr}(\Gamma_1)$ is in the (Fischer-Ladner) *closure* of $\mathrm{expr}(\Gamma_0)$. Roughly speaking, the *closure* of an expression e is the set of expressions that can be obtained from e by taking sub-expressions, and unfolding f^* or f^ω to the left, if this factor was at the beginning of the expression. See the long version for a precise definition of closure suited to our framework. Similarly, $\mathrm{expr}(B_1)$ is in the closure of $\mathrm{expr}(B_0)$. We can also note that given an expression, there are only finitely many ways to subdivide it into a list or into a set of lists, which gives us finitely many possible sequents. Notice that we rely here on the fact that we allow unfolding of $\cdot^+$ and $\cdot^\omega$ only at the beginning of a list, thereby preventing multiple consecutive unfoldings of the same expression. ◀

We will now take a sound sequent, and build a preproof using only invertible instances of our rules, i.e. rules that are locally sound in both directions: the premises are true if and only if the conclusion is true. We proceed by induction on the outermost operation of the first expression of a list.

Outermost operation	Left side	Right side
$+$	$\dfrac{e,\Gamma \to B \qquad f,\Gamma \to B}{e+f,\Gamma \to B}\ {+}\text{-l}$	$\dfrac{a,\Gamma \to \langle e,\Lambda\rangle;\langle f,\Lambda\rangle;B}{a,\Gamma \to \langle e+f,\Lambda\rangle;B}\ {+}\text{-r}$
$\cdot$	$\dfrac{e,f,\Gamma \to B}{e\cdot f,\Gamma \to B}\ {\cdot}\text{-l}$	$\dfrac{a,\Gamma \to \langle e,f,\Lambda\rangle;B}{a,\Gamma \to \langle e\cdot f,\Lambda\rangle;B}\ {\cdot}\text{-r}$
$\cdot^+$	$\dfrac{e,\Gamma \to B \qquad e,e^+,\Gamma \to B}{e^+,\Gamma \to B}\ *\text{-l}$	$\dfrac{a,\Gamma \to \langle e,\Lambda\rangle;\langle e,e^+,\Lambda\rangle;B}{a,\Gamma \to \langle e^+,\Lambda\rangle;B}\ *\text{-r}$
$\cdot^\omega$	$\dfrac{e,e^\omega \to B}{e^\omega \to B}\ \omega\text{-l}$	$\dfrac{a,\Gamma \to \langle f,f^\omega\rangle;B}{a,\Gamma \to \langle f^\omega\rangle;B}\ \omega\text{-r}$

Figure 3 Invertible rules of the system, without the match rule.

We first apply greedily the invertible rules from Figure 3. Notice that at each step, the first expression of the list becomes a (strict) subexpression of the previous one. Since the subexpression relation is well-founded, we must at some point obtain a finite tree with leaves of the form $a,\Gamma \to \langle a_1,\Gamma_1\rangle;\ldots;\langle a_n,\Gamma_n\rangle$ (with a and a_i letters). Moreover, each of those leaves are sound sequents, since all the rules we applied were invertible. For each leaf of this form, we can now remove each $\langle a_i,\Gamma_i\rangle$ with $a_i \neq a$ using the wkn rule, then match the remaining as follows.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \to \langle\Gamma_{i_1}\rangle;\ldots;\langle\Gamma_{i_k}\rangle}}{a,\Gamma \to \langle a,\Gamma_{i_1}\rangle;\ldots;\langle a,\Gamma_{i_k}\rangle}\ \text{match}}{a,\Gamma \to \langle a_1,\Gamma_1\rangle;\ldots;\langle a_n,\Gamma_n\rangle}\ \text{wkn}$$
$$\vdots$$

Since the bottom sequent is sound, we know that the top one is too (we only removed useless options). We can therefore repeat the process to get an infinite tree with only identity rules at the leaves. Any sequent in this tree is sound by a straightforward induction. We now need to check that this is a valid tree.

First note that, as this process will always reach a match rule in a finite number of steps, any infinite branch passes through infinitely many match rules, therefore processing an ω-word (every match rule corresponds to a new letter in the word). In other words, to each infinite branch β of the preproof, we can associate an infinite word $word(\beta)$ corresponding to the sequence of match rules performed along β.

▶ **Lemma 12.** *If β is an infinite branch starting in the root sequent $\Gamma_0 \to B_0$, then either β contains a $*$-l thread, or $word(\beta) \in \mathcal{L}(\Gamma_0)$.*

Proof. Let us assume β does not contain a $*$-l thread. Since the match rule strictly decreases the size of the list on the left, we know that β contains infinitely many $*$-l rules or ω-l rules, because these are the only rules that can increase the size of the list (if we call size the number of characters in the list).

The intuition is then that as soon as no $*$-l expression is unfolded infinitely many times, the unfoldings of both kinds of fixed points ($\cdot^+$ and $\cdot^\omega$) of Γ respect their semantics. It is then natural that the word obtained via such unfoldings, together with arbitrary choices for disjunctions, is in $\mathcal{L}(\Gamma)$. See the long version for a detailed proof. ◀

▶ **Theorem 13.** *The regular and cut-free fragment of S_ω is complete for ω-regular expressions.*

Proof. Given a sound sequent $\Gamma_0 \to B_0$, we consider the preproof defined above, and we prove its validity.

Let us consider an infinite branch β without $*$-l thread. Let $w = word(\beta)$, by Lemma 12 we have $w \in \mathcal{L}(\Gamma_0)$. Since $\Gamma_0 \to B_0$ is sound, we have $w \in \mathcal{L}(B_0)$. We will use this to build an ω-r thread validating the branch β. To do so, the intuition is that at each disjunctive choice in the right-hand side, we choose according to a parsing witnessing $w \in \mathcal{L}(B_0)$. However we cannot do that in a greedy manner, see the long version for an example showing why.

Let us describe how we build a validating thread for our branch. We start with a list from B_0 that contains our word w, and take the last expression of this list (the one containing $\cdot^\omega$) to begin our thread. We then build it going upwards and always staying on an expression containing $\cdot^\omega$.

The only choices we have to make when building this thread upwards are when we meet the rule $\dfrac{\Gamma \to \langle e, \Sigma\rangle; \langle e, e^+, \Sigma\rangle; B}{\Gamma \to \langle e^+, \Sigma\rangle; B}\ *\text{-r}$ or the rule $\dfrac{a, \Gamma \to \langle e, \Lambda\rangle; \langle f, \Lambda\rangle; B}{a, \Gamma \to \langle e + f, \Lambda\rangle; B}\ +\text{-r}$. In the first case ($*$-r rule), there is a smallest integer n such that e^+ can be replaced with e^n in the lower sequent while preserving the fact that the current remainder of w is in the language of the list. We will then continue while treating e^+ as e^n, and at every $*$-r rule on that thread we either go to e if $n = 1$ or e, e^{n-1} otherwise. This replacement is purely "virtual": we simply keep it in mind as a guide to pick a thread.

In the second case ($+$-r rule), there is at least one side containing our word (without the prefix we already read), so we simply choose it. Virtual replacements of some e^+ by e^n are still taken into consideration here, as can be seen in the example of the long version.

We will necessarily unfold infinitely many times the $\cdot^\omega$ expression chosen at the beginning, since we match all letters of w while keeping the invariant that it belongs to the chosen list.

In the end, we get a valid proof for any sound sequent, which proves the completeness of our system, using only regular proofs thanks to Lemma 11. Note that we could settle here for a weaker match rule, that would only match the first letter. ◀

2.4 Deciding the validity criterion

Given a preproof in our system, we want to decide whether it satisfies the validity criterion.

This section is dedicated to proving the following theorem:

▶ **Theorem 14.** *It is decidable in PSPACE whether a given cyclic preproof of S_ω is valid.*

The arguments are similar to those in e.g. [18, 13]. We summarise here the main ideas, we will build on them in the next section and for the transfinite case in Section 3.5.

We start by introducing an auxiliary notion:

▶ **Definition 15** (Sequent transition)**.** *Given two sequents S_1, S_2, a* transition *from S_1 to S_2 is a function $\varphi : pos(S_1) \times pos(S_2) \to \{\,\vdots, |, \Big|\}$. It encodes a way of linking S_1 to S_2 by threads: the value $\varphi(p_1, p_2)$ will be equal to*

- $\vdots$ *if there is no thread from p_1 to p_2,*
- $|$ *if there is a thread with no v-unfolding from p_1 to p_2,*
- $\Big|$ *if there is a thread with v-unfolding from p_1 to p_2.*

We only represent here non-trivial transitions, i.e. we consider only threads of length at least 1. Notice that here S_1 and S_2 are sequents in the finite representation of the proof tree, so they might represent an infinite set of sequents in the unfolded proof tree. Therefore, there might be several different ways of linking them by threads, yielding different transitions φ.

Composing transitions. We define an order on $\{\nmid, |, \blacklozenge\}$ by setting $\nmid < | < \blacklozenge$, and a product law $\cdot$ by setting $\nmid$ as absorbing and $|$ as neutral.

If we have transitions φ from S_1 to S_2, and φ' from S_2 to S_3, they can be composed to yield a transition $\varphi'' = \varphi \odot \varphi'$ from S_1 to S_3. This composed transition is defined by $\varphi''(p_1, p_3) = \max_{p_2 \in pos(S_2)} \varphi(p_1, p_2) \cdot \varphi(p_2, p_3)$. This gives to the set of transitions a structure of finite monoid.

Guessing a bad transition. A *self-transition* on a sequent S is a transition from S to S. A self-transition φ is called *idempotent* if $\varphi \odot \varphi = \varphi$. An idempotent transition on S is called *bad* if for all $p \in pos(S)$, we have $\varphi(p, p) \neq \blacklozenge$.

The validity algorithm is based on the following observation:

▶ **Lemma 16.** *A regular proof is invalid if and only if it contains a bad idempotent transition.*

Proof. This is a standard application of Ramsey's Theorem, see e.g. [13, Thm 4]. ◀

We can finally design a nondeterministic algorithm, which will guess such a bad idempotent transition. It amounts to guessing a branch and a segment along this branch witnessing the idempotent bad transition. The transition φ is computed on-the-fly on this segment. Since keeping a transition φ in memory only takes polynomial space, and NPSPACE =PSPACE, we end up with a PSPACE algorithm.

▶ Remark 17. If the size of sequents is logarithmic in the size of the proof, this algorithm is actually in LOGSPACE. This is put to use in the next section.

2.5 Pspace inclusion algorithm via proof search

We will now combine the above algorithm with our completeness result, in order to obtain a PSPACE algorithm for inclusion of ω-regular expression. This matches the known complexity of expression inclusion, which is PSPACE-complete even in the case of finite words.

We are now given only the sequent we aim to prove, and we will non-deterministically explore its proof as built in Section 2.3. Notice that this proof can be exponential in the size of the root sequent, but this is not a problem, since the algorithm only guesses a branch and follows it on-the-fly. We only have to ensure that each sequent, and therefore each transition φ, is polynomial in the size of the root sequent. This might however not be the case, because a list $\langle e^+, \Lambda\rangle$ can be unfolded into $\langle e, \Lambda\rangle; \langle e, e^+, \Lambda\rangle$, thereby duplicating an arbitrary sequent Λ. Iterating this could lead to sets of exponential size.

This is solved by adding some syntactic sugar in our system: the sequent $\langle e^+, \Lambda\rangle$ will be unfolded into $\langle e, e^+?, \Lambda\rangle$. More precisely, we perform the following rule replacement:

$$\frac{\Gamma \to \langle e, \Lambda\rangle; \langle e, e^+, \Lambda\rangle; B}{\Gamma \to \langle e^+, \Lambda\rangle; B}\ *\text{-r} \quad \rightsquigarrow \quad \frac{\Gamma \to \langle e, e^+?, \Lambda\rangle; B}{\Gamma \to \langle e^+, \Lambda\rangle; B}\ *\text{-r}$$

The notation $e?$ means that e optional. This is expressed by adding the following pseudo-rule:

$$\frac{\Gamma \to \langle \Lambda\rangle; \langle e, \Lambda\rangle; B}{\Gamma \to \langle e?, \Lambda\rangle; B}\ ?.$$

This does not change the behaviour of the system, but guarantees that all sequents stay polynomial in the size of the root sequent, see the long version. If the size of sequents is bounded by M, the size of any transition is in $O(M^2)$, so a bad idempotent transition, if

it exists, can be computed on-the-fly using polynomial space. Since the rules used in the preproof described in Section 2.3 follow deterministically from the root sequent, we can indeed use nondeterminism to guess a bad branch.

Thus we obtain a nondeterministic PSPACE algorithm for inclusion of ω-regular expressions, via proof search in the system S_ω.

3 The transfinite proof system

The goal is now to adapt the system in order to deal with transfinite expressions, recognising language of transfinite words.

3.1 Ordinals and transfinite words

Ordinals. Let us recall that ordinals are closed under taking successors and limits, and that besides the ordinal 0, every ordinal is either a *successor ordinal* (i.e. of the form $\alpha + 1$), or a *limit ordinal*, such as ω which is the smallest limit ordinal. If β is a limit ordinal and $(x_i)_{i<\beta}$ is a sequence of length β, a subsequence $(x_{i_j})_{j<\omega}$ of length ω is said *cofinal* if for all $i < \beta$, there exists $j \in \omega$ such that $i < i_j$.

Transfinite words. If α is an ordinal, a transfinite word of length α on alphabet Σ is a function $\alpha \to \Sigma$. See the long version for formal definitions and properties of transfinite words.

In this work, we will restrict the length α to be strictly smaller than ω^ω, i.e. α will be smaller than ω^k for some $k \in \mathbb{N}$. These ordinals describe the length of words obtained with expressions that are allowed to nest the ω-power finitely many times.

Transfinite expressions. They are similar to ω-regular expressions, except that the ω operators can now be used freely: they do not need to appear once at the end, but can appear anywhere and be nested. That is, transfinite expressions are generated by the grammar: $e, f := a \in \Sigma \mid e \cdot f \mid e + f \mid e^+ \mid e^\omega$ with no restriction.

The language $\mathcal{L}(e)$ of a transfinite expression e is defined as expected, the formal definitions at the beginning of section 2 for semantics of ω-regular expressions can be used in the transfinite case as well. For instance, the word $(a^\omega b)^\omega$ is in the language $\mathcal{L}((a^\omega + b^+)^\omega)$.

3.2 Adapting the proof system

The new proof system. To build a proof system dealing with transfinite expressions, we will basically keep the same rules as in S_ω, except that ω operators are not required to appear at the end of lists anymore. This gives rise to the following relaxed rules for ω:

$$\frac{e, e^\omega, \Gamma \to B}{e^\omega, \Gamma \to B}\ \omega\text{-l} \qquad \frac{\Gamma \to \langle f, f^\omega, \Lambda\rangle; B}{\Gamma \to \langle f^\omega, \Lambda\rangle; B}\ \omega\text{-r}$$

Another difference will be that a preproof will not be a tree anymore, but a *forest*, i.e. a set of trees with distinct roots. This will allow us to consider branches of ordinal length: after taking ω steps in a tree, a branch can "jump" to the root of another tree via a *limit condition*, analogous to the validity condition of the previous section.

Branches, threads and limit sequents. We define inductively these notions as follows. These definitions are mutually recursive, but well-founded: the notions are defined together for a fixed ordinal length, before going to the next one or the limit.

- A *transfinite (resp. limit) branch* is a transfinite sequence of sequent positions in the forest (resp. of limit length), starting at the main root sequent of the proof. The successor of a sequent must be just above it in the forest, and any non successor sequent must be the limit sequent of the limit branch before, as defined below.
- A *transfinite (resp. limit) thread* is a transfinite sequence of expression occurrences following a transfinite (resp. limit) branch, while respecting immediate ancestry for successor sequents, and going to the corresponding expression of the limit sequent when jumping to the limit sequent, as defined below.

  A limit thread with a cofinal sequence of expressions that are principal for a rule r is called a r thread.
- The *limit sequent* of a limit branch, when it exists, is a root sequent from some tree in the proof forest, possibly the tree containing this limit branch. We define it by considering the ω-l and ω-r limit threads following the branch cofinally. On the left side, there must be an ω-l thread, that is principal infinitely often on the same sequent of the form e^ω, Γ, such that no rule is applied on Γ after some point. The corresponding limit sequent will have Γ as left-hand side.

  We proceed similarly to get the lists on the right side of the limit sequent. Given an ω-r limit thread principal infinitely often on some list $\langle e^\omega, \Gamma\rangle$, with Γ untouched after some point, we will have a list $\langle\Gamma\rangle$ on the right-hand side of the limit sequent. Any list on the right that cannot meet these conditions is discarded in the limit sequent. In both cases (left and right of the sequent), we call e^ω the *frontier expression* of that list.

  The threads are prolonged to that limit sequent the natural way, by taking the limit of an inactive thread on the right of a frontier expression as the corresponding expression in the limit sequent.

These definitions are illustrated in the long version, with an example of a proof.

A visualisation of a limit branch of length ω^2 is given in Figure 4. Such a branch goes through ω trees, not necessarily distinct.

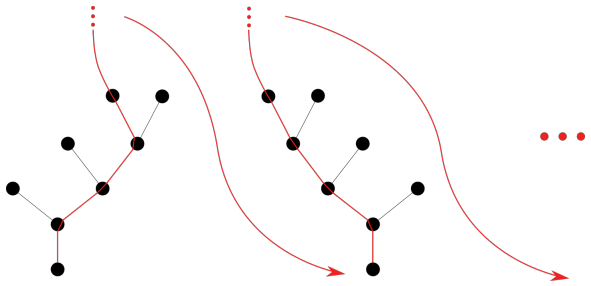


Figure 4 Example of a limit branch of length ω^2.

Validity condition. A proof forest is *valid* if any limit branch either contains a cofinal $*$-l thread, or has its limit sequent appearing as the root of a tree in the proof forest.

A proof forest is called *cyclic* (or *regular*) if it is the unfolding of a finite graph (not necessarily connected), or equivalently if it contains finitely many non-isomorphic subtrees.

Let us call $\mathcal{S}_t$ this proof system for inclusion of transfinite expressions.

3.3 Soundness

The soundness of $\mathcal{S}_t$ is shown in a similar way to the one of S_ω. We first note that as the rules are locally sound, Lemma 4 still holds for $\mathcal{S}_t$, meaning that any finite proof is sound.

Given a valid proof P in $\mathcal{S}_t$, with root sequent $\Gamma_0 \to B_0$, and a word $w \in \mathcal{L}(\Gamma_0)$, we can still build a labelled proof $P(w)$ to witness $w \in \mathcal{L}(B_0)$. This is done in the same way as before, but we also add the limit sequents for every new limit branch that appears, while preserving the labelling in the natural way.

Note that this process can lead to a bigger forest, since a single root sequent in the initial proof P can lead to several ones with different labellings in this proof. We can still build the forest using transfinite recursion, knowing that there are less than ω^ω trees.

We will reuse the concept of correct labelling and label-soundness of a sequent $\Gamma' \to B$, meaning that the word $\text{concat}(\Gamma')$ is in $\mathcal{L}(\Gamma')$ (and is correctly split if Γ' is a list) and $\mathcal{L}(B)$ respectively.

▶ **Lemma 18.** *In $P(w)$, any limit branch can be extended (i.e. has a limit sequent).*

Proof. This is simply a consequence of the fact that we build $P(w)$ according to a parsing of w in Γ_0. Therefore we cannot unfold infinitely many times a same expression e^+. Since the validity condition asks for either a $*$-l thread or a limit sequent, we must be in the second case on all limit branches of $P(w)$. ◀

▶ **Lemma 19.** *In $P(w)$, there is no transfinite branch that goes infinitely many times on the left at a cut rule.*

Proof. Suppose that there is a transfinite branch that does go infinitely many times on the left. Let us take the smallest prefix of that branch that still respects that condition (possible by well-foundedness). This is a limit branch (otherwise it can be made even smaller), which goes to the left premise of a cut rule cofinally, which cuts any ω-r thread. Since Lemma 18 ensures a limit sequent for that branch, there has to be an ω-r thread (the right side of a sequent is nonempty), hence the contradiction. ◀

As before, the maximal transfinite branches in $P(w)$ can be ordered from left-to-right, by comparing them at the first cut rule where they take a different direction (which exists by well-order property). We denote that order by $<$.

▶ **Lemma 20.** *The order $<$ over the maximal transfinite branches of $P(w)$ is well-founded.*

Proof. The proof is similar to the one of Lemma 7, the only notable change being that the word over $\{0, 1\}$ associated to a branch is now transfinite. ◀

▶ **Lemma 21.** *In $P(w)$, all lists on the left side are correctly labelled, and all sequents are label-sound.*

Proof. As in Lemma 8, we prove this by a transfinite induction on the left-to-right order $<$ on branches, which is well-founded by Lemma 20.

Let us call C the set of maximal transfinite branches from $P(w)$. Assume that, for some branch $\beta \in C$, every $\beta' < \beta$ verifies the property.

The first thing we want to prove is the correct labelling in β. This part is done as for S_ω, by induction on the branch. We need to add the limit case since the branch is transfinite. Since limit sequents are untouched in the limiting process, the limit case of the induction is straightforward i.e. limit sequents are correctly labelled.

We now want to prove the second part of the induction. Let us call v the vertex just above the last left cut of β. We want to prove the label-soundness of the proof rooted in v. We call β_v the part of β above v. We know that in β_v, a sequent is label-sound if its successor in the branch is. We want to prove by transfinite induction on the length of β_v that the sequent at v is label-sound if the last sequent of the branch is (recall that in $\mathcal{S}_t$, all branches have a last sequent).

What we need for that is to prove that if the limit sequent of a limit branch is label-sound, then so is (at least) one sequent in that branch.

Let us consider such a limit branch, with limit sequent $\Gamma' \to B$. The word $\mathrm{concat}(\Gamma')$ is in the language of some list $\Lambda \in B$. We can now use the exact same process as for S_ω (see long version) to prove that there is a label-sound sequent $\Pi', \Gamma' \to \langle f^\omega, \Delta\rangle; D$ in the branch before. The only difference is that transfinite branches can be hidden between two ω-r rules, but they are dealt with using the induction hypothesis for shorter branches.

This completes the inductive proof for the label-soundness of β_v. Using the global induction on the well-order $<$ on branches, we get the final result. ◀

▶ **Theorem 22.** *Any valid proof in $\mathcal{S}_t$ is sound.*

Proof. By Lemma 21, the root sequent of $P(w)$ is label-sound, and this is true for any $w \in \mathcal{L}(\Gamma_0)$. This means that for any $w \in \mathcal{L}(\Gamma_0)$, we have $w \in \mathcal{L}(B_0)$, thus any valid proof has a sound root sequent. ◀

3.4 Cut-free regular completeness of $\mathcal{S}_t$

We start with the following observation, a straightforward generalisation of Lemma 11:

▶ **Lemma 23.** *In a proof forest without cut and without useless trees (that can be removed while preserving the validity), there can only be finitely many different sequents.*

▶ **Theorem 24** (Completeness)**.** *Given two expressions e and f such that $\mathcal{L}(e) \subseteq \mathcal{L}(f)$, there exists a cut-free cyclic proof forest for $e \to \langle f\rangle$. Moreover, the construction is effective.*

Proof. Due to space constraints, we only sketch the proof here, details can be found in the long version. As before, we build the proof using a straightforward deterministic bottom-up process, which can be done algorithmically. This time however, in order to show that the obtained proof satisfies the validity condition, we use a model of transfinite automata that helps us to exhibit a validating thread or a limit sequent for each limit branch.

The idea of the proof is to follow the runs of automata $\mathcal{A}_e$ and $\mathcal{A}_f$ canonically associated to e and f, and to build a proof whose nodes are labelled by states of these automata. A state will be associated to each list of expressions, so for each sequent, we will have one state on the left side and possibly several on the right side. Notice that this intuitively corresponds to building a run in a product automaton $\mathcal{A}_e \times \mathcal{P}(\mathcal{A}_f)$, where $\mathcal{P}(\mathcal{A}_f)$ is a powerset automaton obtained from $\mathcal{A}_f$. Since the structure of automata closely follow the structure of expressions, we can always keep the wanted invariants. Limit nodes are built by looking at all the infinite threads in limit branches, and are labelled by the set of states seen cofinally in the corresponding runs. We thereby ensure that the resulting proof is valid. ◀

3.5 Decidability and Complexity

We generalise here the decidability and complexity results obtained in Sections 2.4 and 2.5:

▶ **Theorem 25.**

- *Given a cyclic preproof in $\mathcal{S}_t$, there is a* PSPACE *algorithm deciding whether it is valid.*
- *Given a sequent $\Gamma \to B$, there is a* PSPACE *algorithm deciding whether there is a valid proof of $\mathcal{S}_t$ with root $\Gamma \to B$.*

As before, the second item is deduced from the first together with Theorem 24.

Given a regular preproof that can be explored (or built) on-the-fly, we will again use the formalism of *transitions*: if S_1 and S_2 are sequents in the finite representation of a proof, we will use a function $\varphi : pos(S_1) \times pos(S_2) \to \{\,\vdots\,, |\,, \blacklozenge\}$ to sum up the information about threads from S_1 to S_2 in a particular path of the unfolded proof. We also mark the unfoldings of ω on the left, since we need those to compute limit sequents.

Limit processes

We have to account for the fact that a transition may now represent a path containing (nested) passages to the limit. We verify that such passages to the limit can be effectively computed, and incorporated in our saturation procedure. Remark that the information stored in a transition is enough to identify a frontier expression in an idempotent sequent. By another application of Ramsey's theorem, this will allow us to compute limit sequents, and build transitions corresponding to branches of any length (by keeping only threads to the right of frontier expressions). Now, according to the transfinite validity criterion, an idempotent transition is bad if it does not have a $*$-l thread or a limit sequent. As before, our goal is to guess a bad idempotent transition corresponding to a transfinite branch, if any exists. Notice that guessing such a transition involves guessing a starting point, and that starting points at different levels of ω nesting may differ. This means that our nondeterministic algorithm has to store a current prefix of guessed transition for each level, in order to build the final bad idempotent transition. An example of a run of this algorithm can be found in the long version.

Compact notation

When building the proof on-the-fly according to the construction of Section 3.4, we also need to ensure that transitions stay of polynomial size. To this end, as in Section 2.5, we will use the compact notation $e?$ to avoid an exponential blow-up of sequent size. Note that this simplified representation allows passage to the limit sequent, in the sense that the computation of the limit sequent of a branch using compact notation will yield a compact notation of the correct sequent. As before, this compact notation allows us to obtain a bound on the size of sequents which is polynomial with respect to the size of the root sequent, see the long version for details.

Thus, we obtain the following corollary, which is a new result to the best of our knowledge.

▶ **Corollary 26.** *Deciding the inclusion of transfinite expressions is in* PSPACE.

Conclusion

In our completeness proof, the sets of lists on the right sides of sequents perform some kind of powerset construction. Doing so, we avoid an intricate determinisation procedure such as the Safra construction [19]. We believe it can be considered that this complexity

of determinisation is "hidden" in the validity condition, following various infinite threads simultaneously. This has the advantage of modularity: we separate the pure powerset construction, located in the sequents of the proof, from the complexity of dealing with the acceptance condition, located in the validity condition of the proof. Whereas when determinising Büchi automata, these two causes for state-blowup are merged in the states of the resulting deterministic Rabin automaton. A more detailed investigation of this phenomenon and its advantages can be the subject of a future work.

Contrarily to what happens on ω words, the transfinite system $\mathcal{S}_t$ cannot be seen as an instance of a proof system for linear μ-calculus, as $\cdot^\omega$ is no longer a fixed point operator in the transfinite setting. This manifests concretely by the loss of symmetry between $\cdot^+$ and $\cdot^\omega$ in the validity condition when going from S_ω to $\mathcal{S}_t$.

References

1 Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukás Holík, Chih-Duo Hong, Richard Mayr, and Tomás Vojnar. Simulation subsumption in ramsey-based Büchi automata universality and inclusion testing. In *CAV*, volume 6174 of *LNCS*, pages 132–147. Springer Verlag, 2010. doi:10.1007/978-3-642-14295-6_14.

2 Bahareh Afshari and Graham E. Leigh. Cut-free completeness for modal mu-calculus. In *LICS*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005088.

3 Nicolas Bedon. Finite automata and ordinals. *Theor. Comput. Sci.*, 156(1&2):119–144, 1996. doi:10.1016/0304-3975(95)00006-2.

4 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013. doi:10.1145/2429069.2429124.

5 James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *TABLEAUX*, volume 3702 of *Lecture Notes in Artificial Intelligence*, pages 78–92. Springer Verlag, 2005. doi:10.1007/11554554_8.

6 James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 51–62, 2007. doi:10.1109/LICS.2007.16.

7 Samuel Buss. The undecidability of k-provability. *Annals of Pure and Applied Logic*, 53(1):75–102, 1991. doi:10.1016/0168-0072(91)90059-U.

8 Yaacov Choueka. Finite automata, definable sets, and regular expressions over ω^n-tapes. *J. Comput. Syst. Sci.*, 17(1):81–97, 1978. doi:10.1016/0022-0000(78)90036-3.

9 Liron Cohen and Reuben N. S. Rowe. Integrating induction and coinduction via closure operators and proof cycles. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 375–394, Cham, 2020. Springer International Publishing.

10 James Cranch, Michael R. Laurence, and Georg Struth. Completeness results for omega-regular algebras. *J. Log. Algebr. Meth. Program.*, 84(3):402–425, 2015. doi:10.1016/j.jlamp.2014.10.002.

11 Anupam Das and Damien Pous. A cut-free cyclic proof system for kleene algebra. In Renate A. Schmidt and Cláudia Nalon, editors, *Automated Reasoning with Analytic Tableaux and Related Methods – 26th International Conference, TABLEAUX 2017, Brasília, Brazil, September 25-28, 2017, Proceedings*, volume 10501 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2017.

12 Anupam Das and Damien Pous. Non-wellfounded proof theory for (Kleene+Action)(Algebras+Lattices). In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, pages 19:1–19:18, 2018.

13 Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, pages 273–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

14 Stephane Demri and Alexander Rabinovich. The complexity of linear-time temporal logic over the class of ordinals. *Logical Methods in Computer Science*, 6, September 2010. doi:10.2168/LMCS-6(4:9)2010.

15 Amina Doumane, David Baelde, Lucca Hirschi, and Alexis Saurin. Towards completeness via proof search in the linear time μ-calculus: The case of büchi inclusions. In *LICS*, pages 377–386. ACM, 2016. doi:10.1145/2933575.2933598.

16 Ioannis Kokkinis and Thomas Studer. Cyclic proofs for linear temporal logic. *Concepts of Proof in Mathematics, Philosophy, and Computer Science*, 6:171, 2016.

17 D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

18 Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, pages 81–92, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/360204.360210.

19 S. Safra. On the complexity of omega -automata. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 319–327, 1988. doi:10.1109/SFCS.1988.21948.

20 Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300. Springer Verlag, 2017. doi:10.1007/978-3-662-54458-7_17.

Decidability for Sturmian Words

Philipp Hieronymi
Department of Mathematics, University of Illinois at Urbana-Champaign, IL, USA
Mathematisches Institut, Universität Bonn, Germany

Dun Ma
Department of Mathematics, University of Illinois at Urbana-Champaign, IL, USA

Reed Oei
Department of Mathematics, University of Illinois at Urbana-Champaign, IL, USA

Luke Schaeffer
Institute for Quantum Computing, University of Waterloo, Canada

Christian Schulz
Department of Mathematics, University of Illinois at Urbana-Champaign, IL, USA

Jeffrey Shallit
School of Computer Science, University of Waterloo, Canada

Abstract

We show that the first-order theory of Sturmian words over Presburger arithmetic is decidable. Using a general adder recognizing addition in Ostrowski numeration systems by Baranwal, Schaeffer and Shallit, we prove that the first-order expansions of Presburger arithmetic by a single Sturmian word are uniformly ω-automatic, and then deduce the decidability of the theory of the class of such structures. Using an implementation of this decision algorithm called Pecan, we automatically reprove classical theorems about Sturmian words in seconds, and are able to obtain new results about antisquares and antipalindromes in characteristic Sturmian words.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Decidability, Sturmian words, Ostrowski numeration systems, Automated theorem proving

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.24

Related Version *Full Version*: `https://arxiv.org/abs/2102.08207`

Acknowledgements Part of this work was done in the research project "Building a theorem-prover" at the Illinois Geometry Lab in Spring 2020. P.H. and C.S. were partially supported by NSF grant DMS-1654725. We thank Mary Angelica Gramcko-Tursi for carefully reading a draft of this paper.

1 Introduction

It has been known for some time that, for certain infinite words $\mathbf{c} = c_0c_1c_2\cdots$ over a finite alphabet Σ, the first-order logical theory $\mathrm{FO}(\mathbb{N}, <, +, 0, 1, n \mapsto c_n)$ is decidable. In the case where $\mathbf{c}$ is a **k-automatic sequence** for $k \geq 2$, this is due to Büchi [5], although his original proof was flawed. The correct statement appears, for example, in Bruyère et al. [4]. Although the worst-case running time of the decision procedure is truly formidable (and non-elementary), it turns out that an implementation can, in many cases, decide the truth of interesting and nontrivial first-order statements about automatic sequences in a reasonable length of time. Thus, one can easily reprove known results, and obtain new ones, merely by translating the desired result into the appropriate first-order statement φ and running the decision procedure on φ. For an example of the kinds of things that can be proved, see, for example, Goč, Henshall, and Shallit [6].

More generally, the same ideas can be used for other kinds of sequences defined in terms of some numeration system for the natural numbers. Such a numeration system provides a unique (up to leading zeros) representation for n as a sum of terms of some other sequence $(s_n)_{n\geq 1}$. If the sequence $\mathbf{c} = c_0c_1c_2\cdots$ can be computed by a finite automaton taking the representation of n as input, and if further, the addition of represented integers is computable by another finite automaton, then once again the first-order theory $\mathrm{FO}(\mathbb{N}, <, +, 0, 1, n \mapsto c_n)$ is decidable. This is the case, for example, for the so-called Fibonacci-automatic sequences in Mousavi, Schaeffer, and Shallit [14] and the Pell-automatic sequences in Baranwal and Shallit [3].

More generally, the same kinds of ideas can handle Sturmian words. For quadratic numbers, this was first observed by Hieronymi and Terry [9]. In this paper we extend those results to all Sturmian characteristic words. Thus, the first-order theory of Sturmian characteristic words is decidable. As a result, many classical theorems about Sturmian words, which previously required intricate proofs, can be proved automatically by a theorem-prover in a few seconds. As examples, in Section 7 we reprove basic results such as the balanced property and the subword complexity of these words.

Let $\alpha, \rho \in \mathbb{R}$ be such that α is irrational. The **Sturmian word with slope $\boldsymbol{\alpha}$ and intercept $\boldsymbol{\rho}$** is the infinite $\{0,1\}$-word $\mathbf{c}_{\alpha,\rho} = c_{\alpha,\rho}(1)c_{\alpha,\rho}(2)\cdots$ such that for all $n \in \mathbb{N}$

$$c_{\alpha,\rho}(n) = \lfloor \alpha(n+1) + \rho \rfloor - \lfloor \alpha n + \rho \rfloor - \lfloor \alpha \rfloor.$$

When $\rho = 0$, we call $\mathbf{c}_{\alpha,\mathbf{0}}$ the **characteristic word** of slope α. Sturmian words and their combinatorical properties have been studied extensively. We refer the reader to the survey by Berstel and Séébold [12, Chapter 2]. Note that $\mathbf{c}_{\alpha,\rho}$ can be understood as a function from $\mathbb{N}$ to $\{0,1\}$. Let $\mathcal{L}$ be the signature[1] of the first-order logical theory $\mathrm{FO}(\mathbb{N}, <, +, 0, 1)$ and let $\mathcal{L}_c$ denote the signature obtained by adding a single unary function symbol c to $\mathcal{L}$. Now let $\mathcal{N}_{\alpha,\rho}$ be the $\mathcal{L}_c$-structure $(\mathbb{N}, <, +, 0, 1, n \mapsto c_{\alpha,\rho}(n))$, where we expand Presburger arithmetic by a Sturmian word interpreted as a unary function. The main result of this paper is the decidability of the theory of the collection of such expansions. Set $\mathbf{Irr} := (0,1) \setminus \mathbb{Q}$. Let $\mathcal{K}_{\mathrm{sturmian}} := \{\mathcal{N}_{\alpha,\rho} \ : \ \alpha \in \mathbf{Irr}, \rho \in \mathbb{R}\}$, and let $\mathcal{K}_{\mathrm{char}} := \{\mathcal{N}_{\alpha,0} \ : \ \alpha \in \mathbf{Irr}\}$.

▶ **Theorem A.** *The first-order logical theories*[2] $\mathrm{FO}(\mathcal{K}_{\mathrm{sturmian}})$ *and* $\mathrm{FO}(\mathcal{K}_{\mathrm{char}})$ *are decidable.*

So far, decidability was only known for individual $\mathrm{FO}(\mathcal{N}_{\alpha,\rho})$, and only for very particular α. By [9] the logical theory $\mathrm{FO}(\mathcal{N}_{\alpha,0})$ is decidable when α is a quadratic irrational[3]. Moreover, if the continued fraction of α is not computable, it can be seen rather easily that $\mathrm{FO}(\mathcal{N}_{\alpha,0})$ is undecidable.

Theorem A is rather powerful, as it allows to automatically decide combinatorial statements about all Sturmian words. Consider the $\mathcal{L}_c$-sentence φ

$$\forall p \ (p > 0) \to \Big(\forall i \ \exists j \ j > i \wedge c(j) \neq c(j+p)\Big).$$

We observe that $\mathcal{N}_{\alpha,\rho} \models \varphi$ if and only if $\mathbf{c}_{\alpha,\rho}$ is not eventually periodic. Thus the decision procedure from Theorem A allows us to check that no Sturmian word is eventually periodic. Of course, it is well-known that no Sturmian word is eventually periodic, but this example indicates potential applications of Theorem A. We outline some of these in Section 7.

---

[1] In model theory this is usually called (or identified with) the language of the theory. However, here this conflicts with the convention of calling an arbitrary set of words a language.

[2] Given a signature $\mathcal{L}_0$ and a class $\mathcal{K}$ of $\mathcal{L}_0$-structures, the first-order logical theory of $\mathcal{K}$ is defined as the set of all $\mathcal{L}_0$-sentences that are true in all structures in $\mathcal{K}$. This theory is denoted by $\mathrm{FO}(\mathcal{K})$.

[3] A real number is **quadratic** if it is the root of a quadratic equation with integer coefficients.

We not only prove Theorem A, but instead establish a vastly more general theorem of which Theorem A is an immediate corollary. To state this general result, let $\mathcal{L}_m$ be the signature of $\mathrm{FO}(\mathbb{R}, <, +, \mathbb{Z})$, and let $\mathcal{L}_{m,a}$ be the extension of $\mathcal{L}_m$ by a unary predicate. For $\alpha \in \mathbb{R}_{>0}$, we let $\mathcal{R}_\alpha$ denote $\mathcal{L}_{m,a}$-structure $(\mathbb{R}, <, +, \mathbb{Z}, \alpha\mathbb{Z})$. When $\alpha \in \mathbb{Q}$, it has long been known that $\mathrm{FO}(\mathcal{R}_\alpha)$ is decidable (arguably due to Skolem [19]). Recently this result was extended to quadratic numbers.

▶ **Fact 1** (Hieronymi [7, Theorem A]). *Let α be a quadratic irrational. Then $\mathrm{FO}(\mathcal{R}_\alpha)$ is decidable.*

See also Hieronymi, Nguyen and Pak [8] for a computational complexity analysis of this decision procedure. The proof of Fact 1 establishes that if α is quadratic, then $\mathcal{R}_\alpha$ is an ω-automatic structure; that is it can be represented by Büchi automata. Since every ω-automatic structure has a decidable first-order theory, so does $\mathcal{R}_\alpha$. See Khoussainov and Minnes [10] for a survey on ω-automatic structures. The key insight needed to prove ω-automaticity of $\mathcal{R}_\alpha$ is that addition in the Ostrowski-numeration system based on α is recognizable by a Büchi automaton when α is quadratic. See Section 2 for a definition of Ostrowski numeration systems.

As observed in [7], there are examples of non-quadratic irrationals α such that $\mathcal{R}_\alpha$ has an undecidable theory and hence is not ω-automatic. However, in this paper we show that the common theory of the $\mathcal{R}_\alpha$ is decidable. Let $\mathcal{K}$ denote the class of $\mathcal{L}_{m,a}$-structures $\{\mathcal{R}_\alpha \ : \ \alpha \in \mathbf{Irr}\}$.

▶ **Theorem B.** *The theory $\mathrm{FO}(\mathcal{K})$ is decidable.*

Indeed, we will even prove a substantial generalization of Theorem B. For each $\mathcal{L}_{m,a}$-sentence φ, we set $M_\varphi := \{\alpha \in \mathbf{Irr} \ : \ \mathcal{R}_\alpha \models \varphi\}$. Let $\mathbf{Irr}_{\mathrm{quad}}$ be the set of all quadratic irrational real numbers in $\mathbf{Irr}$. Define $\mathcal{M} = (\mathbf{Irr}, <, (\mathrm{M}_\varphi)_\varphi, (\mathrm{q})_{\mathrm{q}\in\mathbf{Irr}_{\mathrm{quad}}})$ to be the expansion of the dense linear order $(\mathbf{Irr}, <)$ by predicates for M_φ for each $\mathcal{L}_{m,a}$-sentence φ, and constant symbols for each quadratic irrational real number in $\mathbf{Irr}$.

▶ **Theorem C.** *The theory $\mathrm{FO}(\mathcal{M})$ is decidable.*

Observe that Fact 1 and Theorem B follow immediately from Theorem C. We outline how Theorem B implies Theorem A. Note that for every irrational α, the structure $\mathcal{R}_\alpha$ defines the usual floor function $\lfloor\cdot\rfloor : \mathbb{R} \to \mathbb{Z}$, the singleton $\{\alpha\}$ and the successor function on $\alpha\mathbb{Z}$. Hence $\mathcal{R}_\alpha$ also defines the set $\{(\rho, \alpha n, c_{\alpha,\rho}(n)) \ : \ \rho \in \mathbb{R}, n \in \mathbb{N}\}$. From the definability of $\{\alpha\}$, we have that the function from $\alpha\mathbb{N}$ to $\{0, \alpha\}$ given by $\alpha n \mapsto \alpha c_{\alpha,\rho}(n)$ is definable in $\mathcal{R}_\alpha$. Thus the $\mathcal{L}_c$-structure $(\alpha\mathbb{N}, <, +, 0, \alpha, \alpha n \mapsto \alpha c_{\alpha,\rho}(n))$ can be defined in $\mathcal{R}_\alpha$, and this definition is uniform in α. Since the former structure is $\mathcal{L}_c$-isomorphic to $\mathcal{N}_{\alpha,\rho}$, we have that for every $\mathcal{L}_c$-sentence φ there is an $\mathcal{L}_{m,a}$-formula $\psi(x)$ such that

- $\varphi \in \mathrm{FO}(\mathcal{K}_{\mathrm{sturmian}})$ if and only if $\forall \mathrm{x}\ \psi(\mathrm{x}) \in \mathrm{FO}(\mathcal{K})$ and
- $\varphi \in \mathrm{FO}(\mathcal{K}_{\mathrm{char}})$ if and only if $\psi(0) \in \mathrm{FO}(\mathcal{K})$.

Even Theorem C is not the most general result we prove. Its statement is more technical and we postpone it until Section 6. However, we want to point out that we can add predicates for interesting subsets of $\mathbf{Irr}$ to $\mathcal{M}$ without changing the decidability of the theory. Examples of such subsets are the set of all $\alpha \in \mathbf{Irr}$ such that the terms in the continued fraction expansion of α are powers of 2, or the set of all $\alpha \in \mathbf{Irr}$ such that the terms in the continued fraction expansion of α are not in some fixed finite set. This means we can not only automatically prove theorems about all characteristic Sturmian words, but also prove theorems about all characteristic Sturmian words whose slope is one of these sets. There is a limit to this

technique. If we add a predicate for the set of all $\alpha \in \mathbf{Irr}$ such that the terms of continued fraction expansion of α are bounded, or add a predicate for the set of elements in **Irr** whose continued fractions have strictly increasing terms, then our method is unable to conclude whether the resulting structure has a decidable theory. See Section 6 for a more precise statement about what kind of predicates can be added.

The proof of Theorem C follows closely the proof from [7] of the ω-automaticity of $\mathcal{R}_\alpha$ for fixed quadratic α. Here we show that the construction of the Büchi automata needed to represent $\mathcal{R}_\alpha$ is actually uniform in α. See Abu Zaid, Grädel, and Reinhardt [20] for a systematic study of uniformly automatic classes of structures. Deduction of Theorem C from this result is then rather straightforward. The key ingredient to establish the ω-automaticity of $\mathcal{R}_\alpha$ is an automaton that can perform addition in Ostrowski-numeration systems. By [9] there is an automaton that recognizes the addition relation for α-Ostrowski numeration systems for fixed quadratic α. So for a fixed quadratic number, there exists a 3-input automaton that accepts the α-Ostrowski representations of all triples of natural numbers x, y, z with $x + y = z$. In order to prove Theorem C, we need a uniform version of such an adder. This general adder is described in Baranwal, Schaeffer, and Shallit [2]. There a 4-input automaton is constructed that accepts 4-tuples consisting of an encoding of a real number α and three α-Ostrowski representations of natural numbers x, y, z with $x + y = z$. See Section 4 for details.

As mentioned above, an implementation of the decision algorithm provided by Theorem A can be used to study Sturmian words. We created a software program called Pecan [15] that includes such an implementation. Pecan is inspired by Walnut [13] by Mousavi, an automated theorem-prover for deciding properties of automatic words. The main difference is that Walnut is based on finite automata, while Pecan uses Büchi automata. In our setting it is more convenient to work with Büchi automata instead of finite automata, since the infinite families of words we want to consider – like Sturmian words – are indexed by real numbers. Section 7 provides more information about Pecan and contains further examples how Pecan is used prove statements about Sturmian words. Pecan's implementation is discussed in more detail in [16].

2 Preliminaries

Throughout, i, j, k, ℓ, m, n are used for natural numbers. Let X, Y be two sets and $Z \subseteq X \times Y$. For $x \in X$, we let Z_x denote the set $\{y \in Y \ : \ (x, y) \in Z\}$. Similarly, given a function $f : X \times Y \to W$ and $x \in X$, we write f_x for the function $f_x : Y \to W$ that maps $y \in Y$ to $f(x, y)$.

Given a (possibly infinite word) w over an alphabet Σ, we write w_i for the i-th letter of w, and $w|_n$ for $w_1 \cdots w_n$. We write $|w|$ for the length of w. We denote the set of infinite words over Σ by Σ^ω. If Σ is totally ordered by an $\prec$, we let $\prec_{\text{lex}}$ denote the corresponding lexicographic order on Σ^ω. Letting $u, v \in \Sigma^\omega$, we also write $u \prec_{\text{colex}} v$ if there is a maximal i such that $u_i \neq v_i$, and $u_i < v_i$ for this i. Note that while $\prec_{\text{lex}}$ is a total order on Σ^ω, the order $\prec_{\text{colex}}$ is only a partial order. However, for a given $\sigma \in \Sigma$, the order $\prec_{\text{colex}}$ is a total order on the set of all words $v \in \Sigma^\omega$ such that v_j is eventually equal to σ.

A **Büchi automaton (over an alphabet Σ)** is a quintuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accept states.

Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be a Büchi automaton. Let $\sigma \in \Sigma^\omega$. A **run of σ from p** is an infinite sequence s of states in Q such that $s_0 = p$, $(s_n, \sigma_n, s_{n+1}) \in \Delta$ for all $n < |\sigma|$. If $p \in I$, we say s is a **run of σ**. Then σ is **accepted by $\mathcal{A}$** if there is a run $s_0 s_1 \cdots$ of σ such that

$\{n : s_n \in F\}$ is infinite. We call this run an accepting run. We let $L(\mathcal{A})$ be the set of words accepted by $\mathcal{A}$. There are other types of ω-automata with different acceptance conditions, but in this paper we only consider Büchi automata.

Let Σ be a finite alphabet. We say a subset $X \subseteq \Sigma^\omega$ is **ω-regular** if it is recognized by some Büchi automaton. Let $u_1, \dots, u_n \in \Sigma^\omega$. We define the **convolution** $c(u_1, \dots, u_n)$ of $u_1, \dots, u_n$ as the element of $(\Sigma^n)^\omega$ whose value at position i is the n-tuple consisting of the values of $u_1, \dots, u_n$ at position i. We say that $X \subseteq (\Sigma^\omega)^n$ is **ω-regular** if $c(X)$ is ω-regular.

▶ **Fact 2.** *The collection of ω-regular sets is closed under union, intersection, complementation and projection.*

Closure under complementation is due to Büchi [5]. We refer the reader to Khoussainov and Nerode [11] for more information and a proof of Fact 2. As consequence of Fact 2, we have that for every ω-regular subset $W \subseteq (\Sigma^\omega)^{m+n}$ the set $\{s \in (\Sigma^\omega)^m \ : \ \forall t \in (\Sigma^\omega)^n \ (s,t) \in W\}$ is also ω-regular.

2.1 ω-regular structures

Let $\mathcal{U} = (U; R_1, \dots, R_m)$ be a structure, where U is a non-empty set and $R_1, \dots, R_m$ are relations on U. We say $\mathcal{U}$ is **ω-regular** if its domain and its relations are ω-regular.

Büchi's theorem [5] on the decidability of monadic second-order theory of one successor immediately gives the following well-known fact.

▶ **Fact 3.** *Let $\mathcal{U}$ be an ω-regular structure. Then the theory $\mathrm{FO}(\mathcal{U})$ is decidable.*

In this paper, we will consider families of ω-regular structures that are uniform in the following sense. Fix $m \in \mathbb{N}$ and a map $\mathrm{ar} : \{1, \dots, m\} \to \mathbb{N}$. Let Z be a set and for $z \in Z$ let $\mathcal{U}_z$ be a structure $(U_z; R_{1,z}, \dots, R_{m,z})$ such that $R_{i,z} \subseteq U_z^{\mathrm{ar}(i)}$. We say that $(\mathcal{U}_z)_{z \in Z}$ is a **uniform family of ω-regular structures** if

- $\{(z, y) \ : \ y \in U_z\}$ is ω-regular,
- $\{(z, y_1, \dots, y_{\mathrm{ar}(i)}) \ : \ (y_1, \dots, y_{\mathrm{ar}(i)}) \in R_{i,z}\}$ is ω-regular for each $i \in \{1, \dots, m\}$.

From Büchi's theorem, we immediately obtain the following.

▶ **Fact 4.** *Let $(\mathcal{U}_z)_{z \in Z}$ be a uniform family of ω-regular structures, and let φ be a formula in the signature of these structures. Then the set $\{(z, u) \ : \ z \in Z, u \in U_z, \ \mathcal{U}_z \models \varphi(u)\}$ is ω-regular, and the theory $\mathrm{FO}(\{\mathcal{U}_z \ : \ z \in Z\})$ is decidable.*

Proof. When φ is an atomic formula, the statement follows immediately from the definition of a uniform family of ω-regular structures and the ω-regularity of equality. By Fact 2, the statement holds for all formulas. ◀

2.2 Binary representations

For $k \in \mathbb{N}_{>1}$ and $b = b_0 b_1 b_2 \cdots b_n \in \{0,1\}^*$, we define $[b]_k := \sum_{i=0}^n b_i k^i$. For $N \in \mathbb{N}$ we say $b \in \{0,1\}^*$ is a **binary representation** of N if $[b]_2 = N$.

Throughout this paper, we will often consider infinite words over the (infinite) alphabet $\{0,1\}^*$. Let $[\cdot]_2 : (\{0,1\}^*)^\omega \to \mathbb{N}^\omega$ be the function that maps $u = u_1 u_2 \cdots \in (\{0,1\}^*)^\omega$ to $[u_1]_2 [u_2]_2 [u_3]_2 \cdots$ We will consider the following different relations on $(\{0,1\}^*)^\omega$.
Let $u, v \in (\{0,1\}^*)^\omega$. We write $u <_{\mathrm{lex},2} v$ if $[u]_2$ is lexicographically smaller than $[v]_2$. We write $u <_{\mathrm{colex},2} v$ if there is a maximal i such that $[u_i]_2 \neq [v_i]_2$, and $[u_i]_2 < [v_i]_2$. Note that while $<_{\mathrm{lex},2}$ is a total order on $(\{0,1\}^*)^\omega$, the order $<_{\mathrm{colex},2}$ is only a partial order. However,

$<_{\mathrm{colex},2}$ is a total order on the set of all words $v \in (\{0,1\}^*)^\omega$ such that $[v]_j$ is eventually 0. Let $u = u_1u_2\cdots, v = v_1v_2\cdots \in (\{0,1\}^*)^\omega$. Let k be minimal such that $[u_k]_2 \neq [v_k]_2$. We write $u <_{\mathrm{alex},2} v$ if either k is even and $[u_k]_2 < [v_k]_2$, or k is odd and $[u_k]_2 > [v_k]_2$.

2.3 Ostrowski representations

We now introduce Ostrowski representations based on the continued fraction expansions of real numbers. We refer the reader to Allouche and Shallit [1] and Rockett and Szüsz [18] for more details. A **finite continued fraction expansion** $[a_0; a_1, \ldots, a_k]$ is an expression of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_k}}}}$$

For a real number α, we say $[a_0; a_1, \ldots, a_k, \ldots]$ is **the continued fraction expansion of α** if $\alpha = \lim_{k\to\infty}[a_0; a_1, \ldots, a_k]$ and $a_0 \in \mathbb{Z}$, $a_i \in \mathbb{N}_{>0}$ for $i > 0$. In this situation, we write $\alpha = [a_0; a_1, \ldots]$. Every irrational number has precisely one continued fraction expansion. We recall the following well-known fact about continued fractions.

▶ **Fact 5.** *Let $\alpha = [a_0; a_1, \ldots], \alpha' = [a'_0; a'_1, \ldots] \in \mathbb{R}$ be irrational. Let $k \in \mathbb{N}$ be minimal such that $a_k \neq a'_k$. Then $\alpha < \alpha'$ if and only if*

- *k is even and $a_k < a'_k$, or*
- *k is odd and $a_k > a'_k$.*

For the rest of this subsection, fix a positive irrational real number $\alpha \in (0,1)$ and let $[a_0; a_1, a_2, \ldots]$ be the continued fraction expansion of α. Let $k \geq 1$. A quotient $p_k/q_k \in \mathbb{Q}$ is the ***k*-th convergent of α** if $p_k \in \mathbb{N}$, $q_k \in \mathbb{Z}$, $\gcd(p_k, q_k) = 1$ and $\frac{p_k}{q_k} = [a_0; a_1, \ldots, a_k]$. Set $p_{-1} := 1, q_{-1} := 0$ and $p_0 := a_0, q_0 := 1$. The convergents satisfy the following equations for $n \geq 1$:

$$p_n = a_n p_{n-1} + p_{n-2}, \quad q_n = a_n q_{n-1} + q_{n-2}.$$

The ***k*-th difference β_k of α** is defined as $\beta_k := q_k\alpha - p_k$. We use the following facts about k-th differences: for all $n \in \mathbb{N}$

1. $\beta_n > 0$ if and only if n is even,
2. $\beta_0 > -\beta_1 > \beta_2 > -\beta_3 > \beta_4 > \ldots$, and
3. $-\beta_n = a_{n+2}\beta_{n+1} + a_{n+4}\beta_{n+3} + a_{n+6}\beta_{n+5} + \ldots$.

We now recall a numeration system due to Ostrowski [17].

▶ **Fact 6** ([18, Ch. II-§4])**.** *Let $X \in \mathbb{N}$. Then X can be written uniquely as*

$$X = \sum_{n=0}^{N} b_{n+1}q_n \tag{1}$$

where $0 \leq b_1 < a_1$, $0 \leq b_{n+1} \leq a_{n+1}$ and $b_n = 0$ whenever $b_{n+1} = a_{n+1}$.

For $X \in \mathbb{N}$ satisfying (1) we write $X = [b_1b_2\cdots b_nb_{n+1}]_\alpha$ and call the word $b_1b_2\cdots b_{n+1}$ an α-Ostrowski representation of X. This representation is unique up to trailing zeros. Let $X, Y \in \mathbb{N}$ and let $b_1b_2\cdots b_{n+1}$ and $c_1c_2\cdots c_{n+1}$ be α-Ostrowski representations of X and

Y respectively. Since Ostrowski representations are obtained by a greedy algorithm, one can see easily that $X < Y$ if and only if $b_1 b_2 \cdots b_{n+1}$ is co-lexicographically smaller than $c_1 c_2 \cdots c_{n+1}$.

We now introduce a similar way to represent real numbers, also due to Ostrowski [17]. Let I_α be the interval $\big[\lfloor\alpha\rfloor - \alpha, 1 + \lfloor\alpha\rfloor - \alpha\big)$.

▶ **Fact 7** (cp. [18, Ch. II.6 Theorem 1]). *Let $x \in I_\alpha$. Then x can be written uniquely as*

$$\sum_{k=0}^{\infty} b_{k+1}\beta_k, \tag{2}$$

where $b_k \in \mathbb{Z}$ with $0 \leq b_k \leq a_k$, and $b_{k-1} = 0$ whenever $b_k = a_k$,(in particular, $b_1 \neq a_1$), and $b_k \neq a_k$ for infinitely many odd k.

For $x \in I_\alpha$ satisfying (2) we write $x = [b_1 b_2 \cdots]_\alpha$ and call the infinite word $b_1 b_2 \cdots$ the α-Ostrowski representation of x. This is closely connected to the integer Ostrowski representation. Note that for every real number there a unique element of I_α such that that their difference is an integer. We define $f_\alpha : \mathbb{R} \to I_\alpha$ to be the function that maps x to $x - u$, where u is the unique integer such that $x - u \in I_\alpha$.

▶ **Fact 8** ([7, Lemma 3.4]). *Let $X \in \mathbb{N}$ be such that $\sum_{k=0}^{N} b_{k+1}q_k$ is the α-Ostrowski representation of X. Then $f_\alpha(\alpha X) = \sum_{k=0}^{\infty} b_{k+1}\beta_k$ is the α-Ostrowski representation of $f_\alpha(\alpha X)$, where $b_{k+1} = 0$ for $k > N$.*

Since $\beta_k > 0$ if and only if k is even, the order of two elements in I_α can be determined by the Ostrowski representation as follows.

▶ **Fact 9** ([7, Fact 2.13]). *Let $x, y \in I_\alpha$ with $x \neq y$ and let $[b_1 b_2 \cdots]_\alpha$ and $[c_1 c_2 \cdots]_\alpha$ be the α-Ostrowski representations of x and y. Let $k \in \mathbb{N}$ be minimal such that $b_k \neq c_k$. Then $x < y$ if and only if*

- **(i)** *$b_{k+1} < c_{k+1}$ if k is even;*
- **(ii)** *$b_{k+1} > c_{k+1}$ if k is odd.*

3 #-binary coding

In this section, we introduce #-binary coding. Fix the alphabet $\Sigma_\# := \{0, 1, \#\}$. Let H_∞ denote the set of all infinite $\Sigma_\#$-words in which # appears infinitely many times. Clearly H_∞ is ω-regular.

Let $C_\# : (\{0,1\}^*)^\omega \to H_\infty$ map an infinite word $b = b_1 b_2 b_3 \cdots$ over $\{0,1\}^*$ to the infinite $\Sigma_\#$-word $\#b_1\#b_2\#b_3\# \cdots$ We note that the map $C_\#$ is a bijection. Let $u = u_1 u_2 u_3 \cdots, v = v_1 v_2 v_3 \cdots \in \Sigma_\#^\omega$. We say u and v are **aligned** if for all $i \in \mathbb{N}$ $u_i = \#$ if and only if $v_i = \#$. This defines an ω-regular equivalence relation on $\Sigma_\#^\omega$. We denote this equivalence relation by $\sim_\#$. The following fact follows easily.

▶ **Fact 10.** *The following sets are ω-regular:*

- $\{(u, v) \in H_\infty^2 \ : \ u \sim_\# v \text{ and } C_\#^{-1}(u) <_{\text{lex},2} C_\#^{-1}(v)\}$,
- $\{(u, v) \in H_\infty^2 \ : \ u \sim_\# v \text{ and } C_\#^{-1}(u) <_{\text{colex},2} C_\#^{-1}(v)\}$,
- $\{(u, v) \in H_\infty^2 \ : \ u \sim_\# v \text{ and } C_\#^{-1}(u) <_{\text{alex},2} C_\#^{-1}(v)\}$.

3.1 #-binary coding of continued fractions

We now code the continued fraction expansions of real numbers as infinite $\Sigma_{\#}$-words.

▶ **Definition 11.** *Let $\alpha \in (0,1)$ be irrational such that $[0; a_1, a_2, \dots]$ is the continued fraction expansion of α. Let $u = u_1 u_2 \cdots \in (\{0,1\}^*)^\omega$ such that $u_i \in \{0,1\}^*$ is a binary representation of a_i for each $i \in \mathbb{Z}_{\geq 0}$. We say that $C_{\#}(u)$ is a **#-binary coding of the continued fraction of α**.*

Let R be the set of elements of $\Sigma_{\#}^\omega$ of the form $(\#(0|1)^*1(0|1)^*)^\omega$. Obviously, R is ω-regular.

▶ **Lemma 12.** *Let $w \in R$. Then there is a unique irrational number $\alpha \in [0,1]$ such that w is a #-binary coding of the continued fraction of α.*

The proof of Lemma 12 is in Appendix A. For $w \in R$, let $\alpha(w)$ be the real number given by Lemma 12. When $v = (v_1, \dots, v_n) \in R^n$, we write $\alpha(v)$ for $(\alpha(v_1), \dots, \alpha(v_n))$.

Even though continued fractions are unique, their #-binary codings are not, because binary representations can have trailing zeroes. This ambiguity is required in order to properly recognize relationships between multiple numbers, as one of the numbers involved may require more bits in a coefficient than the other(s). Occasionally we need to ensure that all possible representations of a given tuple of numbers are contained in a set. For this reason, we introduce the zero-closure of subsets of R^n.

▶ **Definition 13.** *Let $X \subseteq R^n$. The **zero-closure** of X is $\{u \in R^n \ : \ \exists v \in X \ \alpha(u) = \alpha(v)\}$.*

▶ **Lemma 14.** *Let $X \subseteq R^n$ be ω-regular. Then the zero-closure of X is also ω-regular.*

The essence of the proof is that trailing zeroes are the only way that #-binary codings of the same number can differ. The details of proof are technical and can be found in Appendix A.

▶ **Lemma 15.** *The set $\{(w_1, w_2) \in R^2 \ : \ w_1 \sim_{\#} w_2$ and $\alpha(w_1) < \alpha(w_2)\}$ is ω-regular.*

Proof. Let $w_1, w_2 \in R$ be such that $w_1 \sim_{\#} w_2$. By Fact 5 we have that $\alpha(w_1) < \alpha(w_2)$ if only $C_{\#}^{-1}(w_1) <_{\text{alex},2} C_{\#}^{-1}(w_2)$. Thus ω-regularity follows from Fact 10. ◀

▶ **Lemma 16.** *Let $a \in [0,1)$ be a quadratic irrational. Then $\{w \in R \ : \ \alpha(w) = a\}$ is ω-regular.*

Proof. The continued fraction expansion of a is eventually periodic. Thus there is an eventually periodic $u \in (\{0,1\}^*)^\omega$ such that $C_{\#}(u)$ is a #-binary coding of the continued fraction of a. The singleton set containing an eventually periodic string is ω-regular. It remains to expand this set to contain all representations via Lemma 14. ◀

▶ **Lemma 17.** *The set $\{w \in R \ : \ \alpha(w) < \frac{1}{2}\}$ is ω-regular.*

Proof. Let $\alpha(w) = [0; a_1, a_2, \dots]$. It is easy to see that $\alpha(w) < \frac{1}{2}$ if and only if $a_1 > 1$. Thus we need only check that $a_1 \neq 1$. The set of $w \in R$ for which this true is just $R \setminus Y$, where $Y \subseteq \Sigma_{\#}^\omega$ is given by the regular expression $\#10^*(\#(0 \cup 1)^*)^\omega$. ◀

3.2 #-Ostrowski-representations

We now extend the #-binary coding to Ostrowski representations.

▶ **Definition 18.** *Let $v, w \in (\Sigma_\#)^\omega$, let $x = x_1x_2x_3\cdots \in \mathbb{N}^\omega$ and let $b = b_1b_2b_3\cdots \in (\{0,1\}^*)^\omega$ be such that $w = C_\#(b)$ and $[b_i]_2 = x_i$ for each i.*

- *For $N \in \mathbb{N}$, we say that w is a **#-v-Ostrowski representation of X** if v and w are aligned and x is an $\alpha(v)$-Ostrowski representation of N.*
- *For $c \in I_{\alpha(v)}$, we say that w is a **#-v-Ostrowski representation of c** if v and w are aligned and x is an $\alpha(v)$-Ostrowski representation of c.*

We let A_v denote the set of all words $w \in \Sigma_\#^\omega$ such that w is a #-v-Ostrowski representation of some $c \in I_{\alpha(v)}$, and similarly, by A_v^{fin} the set of all words $w \in \Sigma_\#^\omega$ such that w is a #-v-Ostrowski representation of some $N \in \mathbb{N}$.

▶ **Lemma 19.** *The sets*

$$A^{\text{fin}} := \{(v,w) \ : \ v \in R, w \in A_v^{\text{fin}}\}, \text{ and } A := \{(v,w) \ : \ v \in R, w \in A_v\}.$$

are ω-regular. Moreover, $A^{\text{fin}} \subseteq A$.

The straightforward proof is in Appendix A.

▶ **Definition 20.** *Let $v \in R$. We define $Z_v : A_v^{\text{fin}} \to \mathbb{N}$ to be the function that maps w to the natural number whose #-v-Ostrowski representation is w.*
Similarly, we define $O_v : A_v \to I_{\alpha(v)}$ to be the function that maps w to the real number whose #-v-Ostrowski representation is w.

▶ **Lemma 21.** *Let $v \in R$. Then $Z_v : A_v^{\text{fin}} \to \mathbb{N}$ and $O_v : A_v \to I_{\alpha(v)}$ are bijective.*

The proof is in Appendix A.

▶ **Definition 22.** *Let $v \in R$. We write $\mathbf{0}_v$ for $Z_v^{-1}(0)$, and $\mathbf{1}_v$ for $Z_v^{-1}(1)$.*

▶ **Lemma 23.** *The relations $\mathbf{0}_* = \{(v, \mathbf{0}_v) \ : \ v \in R\}$ and $\mathbf{1}_* = \{(v, \mathbf{1}_v) \ : \ v \in R\}$ are ω-regular.*

▶ **Lemma 24.** *Let $s \in A_v^{\text{fin}}$. Then $\alpha(v)Z_v(s) - O_v(s) \in \mathbb{Z}$ and*

$$O_v(\mathbf{1}_v) = \begin{cases} \alpha(v) & \text{if } \alpha(v) < \frac{1}{2}; \\ \alpha(v) - 1 & \text{otherwise.} \end{cases}$$

Again, proofs of these lemmas are in Appendix A.

▶ **Lemma 25.** *The sets*

$$\prec^{\text{fin}} := \{(v,s,t) \in \Sigma_\#^3 \ : \ s,t \in A_v^{\text{fin}} \wedge Z_v(s) < Z_v(t)\},$$
$$\prec := \{(v,s,t) \in \Sigma_\#^3 \ : \ s,t \in A_v \wedge O_v(s) < O_v(t)\}$$

are ω-regular.

Proof of Lemma 25. For $\prec^{\text{fin}}$, first recall that for $X, Y \in \mathbb{N}$ and α irrational, we have $X < Y$ if and only if the α-Ostrowski representation of X is co-lexicographically smaller than the α-Ostrowski representation of Y. Therefore, we need only recognize co-lexicographic ordering on the list of coefficients, with each coefficient ordered according to binary. This follows immediately from Fact 10.

For $\prec$, note that by Fact 9 the usual order on real numbers corresponds to the alternating lexicographic ordering on real Ostrowski representations. Therefore, we need only recognize the alternating lexicographic ordering on the list of coefficients, with each coefficient ordered according to binary. This follows immediately from Fact 10. ◀

We consider $\mathbb{R}^n$ as a topological space using the usual order topology. For $X \subseteq \mathbb{R}^n$, we denote its topological closure by $\overline{X}$.

▶ **Corollary 26.** *Let $W \subseteq (\Sigma_\#^{n+1})^*$ ω-regular be such that $W \subseteq \{(v, s_1, \ldots, s_n) \in (\Sigma_\#^{n+1})^* : s_1, \ldots, s_n \in A_v\}$. Then the following set is also ω-regular:*

$$\overline{W} := \{(v, s_1, \ldots, s_n) \in (\Sigma_\#^{n+1})^* : s_1, \ldots, s_n \in A_v \wedge (O_v(s_1), \ldots, O_v(s_n)) \in \overline{O(W_v)}\}.$$

Proof. Let $(v, s_1, \ldots, s_n) \in (\Sigma_\#^{n+1})^*$ be such that $s_1, \ldots, s_n \in A_v$. Let $X_i = O_v(s_i)$. By the definition of the topological closure, we have that $(X_1, \ldots, X_n) \in \overline{O(W_v)}$ if and only if for all $Y_1, \ldots Y_n, Z_1, \ldots, Z_n \in \mathbb{R}$ with $Y_i < X_i < Z_i$ for $i = 1, \ldots, n$ there are $X' = (X'_1, \ldots, X'_n) \in O(W_v)$ such that $Y_i < X'_i < Z_i$ for $i = 1, \ldots, n$. Thus by Lemma 25, $(v, s_1, \ldots, s_n) \in \overline{W}$ if and only if for all $t_1, \ldots t_n, u_1, \ldots, u_n \in A_v$ with $t_i \prec s_i \prec u_i$, there are $s' = (s'_1, \ldots, s'_n) \in W_v$ such that $t_i \prec s'_i \prec Z_i$ for $i = 1, \ldots, n$. The latter condition is ω-regular by Fact 2. ◀

4 Recognizing addition in Ostrowski numeration systems

The key to the rest of this paper is a general automaton for recognizing addition of Ostrowski representations uniformly. We will prove the following.

▶ **Theorem 27.** *The set $\oplus^{\text{fin}} := \{(v, s_1, s_2, s_3) : s_1, s_2, s_3 \in A_v^{\text{fin}} \wedge Z_v(s_1) + Z_v(s_2) = Z_v(s_3)\}$ is ω-regular.*

Proof. In [2, Section 3] the authors generate an automaton $\mathcal{A}$ over the alphabet $\mathbb{N}^4$ such that a finite word $(d_1, x_1, y_1, z_1)(d_2, x_2, y_2, z_2) \cdots (d_m, x_m, y_m, z_m) \in (\mathbb{N}^4)^*$ is accepted by $\mathcal{A}$ if and only if there are $d_{m+1}, \ldots \in \mathbb{N}$ and $x, y, z \in \mathbb{N}$ such that for $\alpha = [0; d_1, d_2, \ldots]$ we have $[x_1 x_2 \ldots x_m]_\alpha = x$, $[y_1 y_2 \ldots y_m]_\alpha = y$, $[z_1 z_2 \ldots z_m]_\alpha = z$, and $x + y = z$.

We now describe how to adjust the the automaton $\mathcal{A}$ for our purposes. The input alphabet will be $\Sigma_\#^4$ instead of $\mathbb{N}^4$. The new automaton will take four inputs w, s_1, s_2, s_3, where $s_1, s_2, s_3 \in A_w^{\text{fin}}$. We can construct this automaton as follows:

1. Begin with the automaton $\mathcal{A}$ from [2].
2. Add an initial state that transitions to the original start state on $(\#, \#, \#, \#)$. This will ensure that the first character in each string is $\#$.
3. Replace each transition in the automaton with a sub-automaton that recognizes the corresponding relationship between w, s_1, s_2, s_3 in binary. As an example, one of the transitions in Figure 3 of [2] is given as "$-d_i + 1$," meaning that it represents all cases where, letting $w_i, s_{1i}, s_{2i}, s_{i3}$ be the ith letter of w, s_1, s_2, s_3 respectively, we have $s_{3i} - s_{1i} - s_{2i} = -w_i + 1$. This is an affine and hence an automatic relation. Thus it can be recognized by a sub-automaton.
4. The accept states in the resulting automaton are precisely the accept states from the original automaton.

The resulting automaton recognizes $\oplus^{\text{fin}}$. ◀

The automaton constructed above has 82 states. Using our software Pecan, we can formally check that this automaton recognizes the set in Theorem 27. Following a strategy already used in Mousavi, Schaeffer, and Shallit [14, Remark 2.1] we check that our adder satisfies the standard recursive definition of addition on the natural numbers; that is for all $x, y \in \mathbb{N}$

$$\begin{aligned} 0 + y &= y \\ s(x) + y &= s(x + y) \end{aligned}$$

where $x, y \in \mathbb{N}$ and $s(x)$ denotes the successor of x in $\mathbb{N}$. The successor function on $\mathbb{N}$ can be defined using only $<$ as follows:

$s(x) = y$ if and only if $(x < y) \wedge (\forall z\ (z \leq x) \vee (z \geq y))$.

Thus in Pecan we define `bco_succ(a,x,y)` as

```
bco_succ(a,x,y) := bco_valid(a,x) ∧ bco_valid(a,y) ∧ bco_leq(x,y)
∧ ¬bco_eq(x,y) ∧ ∀z. bco_valid(a,z) => (bco_leq(z,x) ∨ bco_leq(y,z))
```

where `bco_eq` recognizes $\{(x, y) : x = y\}$, `bco_leq` recognizes $\{(x, y) : x \leq_{\text{colex}} y\}$, and `bco_valid` recognizes A_{fin}. We now confirm that our adder satisfies the above equations using the following Pecan code:

```
Let x,y,z be ostrowski(a).
Theorem ("Addition base case (0 + y = y).", {
    ∀a. ∀x,y,z. bco_zero(x) => (bco_adder(a,x,y,z) ⇔ bco_eq(y,z))
}).
Theorem ("Addition inductive case (s(x) + y = s(x + y)).", {
    ∀a. ∀x,y,z,u,v. (bco_succ(a,u,x) ∧ bco_succ(a,v,z))
                    => (bco_adder(a,x,y,z) ⇔ bco_adder(a,u,y,v))
}).
```

In the above code `bco_adder` recognizes $\oplus^{\text{fin}}$, `bco_zero` recognizes $\mathbf{0}_*$, and `bco_succ` recognizes $\{(v, x, y) : x, y \in A_v^{\text{fin}}, Z_v(x) + 1 = Z_v(y)\}$. Pecan confirms both statements are true. This proves Theorem 27 modulo correctness of Pecan and the correctness of the implementations of the automata for `bco_eq`, `bco_leq`, `bco_valid` and `bco_zero`. For more details about Pecan, see Section 7.

Using Corollary 26 we can extend the automaton in Theorem 27 to an automaton for addition modulo 1 on I_α. The details are in Appendix B.

▶ **Lemma 28.** *The set* $\oplus := \{(v, s_1, s_2, s_3) : s_1, s_2, s_3 \in A_v \wedge O_v(s_1) + O_v(s_2) \equiv O_v(s_3) \pmod 1\}$ *is ω-regular. Moreover,* $\oplus^{\text{fin}} \subseteq \oplus$.

5 The uniform ω-regularity of $\mathcal{R}_\alpha$

In this section, we turn to the question of the decidability of the logical first-order theory of $\mathcal{R}_\alpha$. Recall that $\mathcal{R}_\alpha := (\mathbb{R}, <, +, \mathbb{Z}, \alpha\mathbb{Z})$ for $\alpha \in \mathbb{R}$. The main result of this section is the following:

▶ **Theorem 29.** *There is a uniform family of ω-regular structures $(\mathcal{D}_v)_{v \in R}$ such that $\mathcal{D}_v \simeq \mathcal{R}_{\alpha(v)}$ for each $v \in R$.*

Theorem 29 then hinges on the following lemma.

▶ **Lemma 30.** *There is a uniform family of ω-regular structures $(\mathcal{C}_a)_{a \in R}$ such that $\mathcal{C}_a \simeq ([-\alpha(a), \infty), <, +, \mathbb{N}, \alpha(a)\mathbb{N})$ for each $a \in R$.*

The proof of Lemma 30 is a uniform version of the argument given in [7] that also fixes some minor errors of the original proof. By Lemma 10 and Theorem 27, we already know that $Z_v : (A_v^{\text{fin}}, \prec_v^{\text{fin}}, \oplus_v^{\text{fin}}) \to (\mathbb{N}, <, +)$ is an isomorphism for every $v \in R$. As our eventual goal also requires us to define the set $\alpha\mathbb{N}$, it turns out to be much more natural to instead use the isomorphism $\alpha(v)Z_v : (A_v^{\text{fin}}, \prec_v^{\text{fin}}, \oplus_v^{\text{fin}}) \to (\alpha(v)\mathbb{N}, <, +)$ and recover $\mathbb{Z}$. We do so by following the argument in [7]. The full proof is availabe in Appendix C.

Proof of Theorem 29. We just observe that $([-\alpha, \infty), <, +, \mathbb{N}, \alpha\mathbb{N})$ defines (in a matter uniform in α) an isomorphic copy of $\mathcal{R}_\alpha$. Now apply Lemma 30. ◀

6 Decidability results

We are now ready to prove the results listed in the introduction. We first recall some notation. Let $\mathcal{L}_m$ be the signature of the first-order structure $(\mathbb{R}, <, +, \mathbb{Z})$, and let $\mathcal{L}_{m,a}$ be the extension of $\mathcal{L}_m$ by a unary predicate. For $\alpha \in \mathbb{R}_{>0}$, let $\mathcal{R}_\alpha$ denote the $\mathcal{L}_{m,a}$-structure $(\mathbb{R}, <, +, \mathbb{Z}, \alpha\mathbb{Z})$. For each $\mathcal{L}_{m,a}$-sentence φ, we set $R_\varphi := \{v \in R \ : \ \mathcal{R}_{\alpha(v)} \models \varphi\}$.

▶ **Theorem 31.** *Let φ be an $\mathcal{L}_{m,a}$-sentence. Then R_φ is ω-regular.*

Proof. By Theorem 29 there is a uniform family of ω-regular structures $(\mathcal{D}_v)_{v\in R}$ such that such that $\mathcal{D}_v \simeq \mathcal{R}_{\alpha(v)}$ for each $v \in R$. Then $R_\varphi = \{v \in R \ : \ \mathcal{D}_v \models \varphi\}$. This set is ω-regular by Fact 4. ◀

Let $\mathcal{N} = (R; (R_\varphi)_\varphi, (X)_{X\subseteq R^n \ \omega\text{-regular}})$ be the relational structure on R with the relations R_φ for every $\mathcal{L}$-sentences φ and $X \subseteq R^n$ ω-regular. Because $\mathcal{N}$ is an ω-regular structure, the theory of $\mathcal{N}$ is decidable.

We now proceed towards the proof of Theorem C. Recall that $\mathbf{Irr} := (0,1) \setminus \mathbb{Q}$.

▶ **Definition 32.** *Let $X \subseteq \boldsymbol{Irr}^{\mathrm{n}}$. Let X_R be defined by*

$$X_R := \{(v_1, \dots, v_n) \in R^n \ : \ v_1 \sim_\# v_2 \sim_\# \cdots \sim_\# v_n \wedge (\alpha(v_1), \dots, \alpha(v_n)) \in X\}$$

*We say X is **recognizable modulo $\sim_\#$** if X_R is ω-regular.*

▶ **Lemma 33.** *The collection of sets recognizable modulo $\sim_\#$ is closed under Boolean operations and coordinate projections.*

Proof. Let $X, Y \subseteq \mathbf{Irr}$ be recognizable modulo $\sim_\#$. It is clear that $(X \cap Y)_R = X_R \cap Y_R$. Thus $X \cap Y$ is recognizable modulo $\sim_\#$. Let X^c be $\mathbf{Irr}^{\mathrm{n}} \setminus \mathrm{X}$, the complement of X. For ease of notation, set $E := \{(v_1, \dots, v_n) \in R^n \ : \ v_1 \sim_\# v_2 \sim_\# \cdots \sim_\# v_n\}$. Then

$$\begin{aligned}
(X^c)_R &= \{(v_1, \dots, v_n) \in R^n \ : \ v_1 \sim_\# v_2 \sim_\# \cdots \sim_\# v_n \wedge (\alpha(v_1), \dots, \alpha(v_n)) \notin X\} \\
&= E \cap \{(v_1, \dots, v_n) \in R^n \ : (\alpha(v_1), \dots, \alpha(v_n)) \notin X\} \\
&= E \cap \{(v_1, \dots, v_n) \in R^n \ : (\alpha(v_1), \dots, \alpha(v_n)) \notin X \vee \neg(v_1 \sim_\# v_2 \sim_\# \cdots \sim_\# v_n)\} \\
&= E \cap (R^n \setminus X_R).
\end{aligned}$$

This set is ω-regular, and hence X^c is recognizable modulo $\sim_\#$.

For coordinate projections, it is enough to consider projections onto the first $n-1$ coordinates. Let $n > 0$ and let π be the coordinate projection onto first $n-1$ coordinates. Observe that $\pi(X) = \{(\alpha_1, \dots, \alpha_{n-1}) \in \mathbb{R}^{n-1} \ : \ \exists \alpha_n \in \mathbb{R} \ (\alpha_1, \dots, \alpha_{n-1}, \alpha_n) \in X\}$. Thus $\pi(X)_R$ is equal to $\{(v_1, \dots, v_{n-1}) \in R^{n-1} : v_1 \sim_\# \cdots \sim_\# v_{n-1} \wedge \exists \alpha_n : (\alpha(v_1), \dots, \alpha(v_{n-1}), \alpha_n) \in X\}$. Note that $v \mapsto \alpha(v)$ is a surjection $R \twoheadrightarrow (0,1) \setminus \mathbb{Q}$. Thus $\pi(X)_R$ is also equal to:

$$\{(v_1, \dots, v_{n-1}) \in R^{n-1} : v_1 \sim_\# \cdots \sim_\# v_{n-1} \wedge \exists v_n : (\alpha(v_1), \dots, \alpha(v_n)) \in X\}.$$

Unfortunately, this set is not necessarily equal to $\pi(X_R)$. There might be tuples $(v_1, \dots, v_{n-1})$ such that no v_n can be found, because it would require more bits in one of its coefficients than $v_1, \dots, v_{n-1}$ have for that coefficient. But $\pi(X_R)$ always contains *some* representation of $\alpha(v_1), \dots, \alpha(v_{n-1})$ with the appropriate number of digits. We need only ensure that removal of trailing zeroes does not affect membership in the language. Thus $\pi(X)_R$ is just the zero-closure of $\pi(X_R)$. Thus $\pi(X)_R$ is ω-regular by Lemma 14. ◀

▶ **Theorem 34.** *Let $X_1, \ldots, X_n$ be recognizable modulo $\sim_\#$ by Büchi automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$, and let $\mathcal{Q}$ be the structure $(\boldsymbol{Irr}, \mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n})$. Then the theory of $\mathcal{Q}$ is decidable.*

Proof. By Lemma 33 every set definable in $\mathcal{Q}$ is recognizable modulo $\sim_\#$. Moreover, for each definable set Y the automaton that recognizes Y modulo $\sim_\#$, can be computed from the automata $\mathcal{A}_\infty, \ldots, \mathcal{A}_\backslash$. Let ψ be a sentence in the signature of $\mathcal{Q}$. Without loss of generality, we can assume that ψ is of the form $\exists x\ \chi(x)$. Set $Z := \{a \in \mathbf{Irr}^\mathrm{n} \ : \ \mathcal{Q} \models \chi(\mathrm{a})\}$. Observe that $\mathcal{Q} \models \psi$ if and only if Z is non-empty. Note for every $a \in \mathbf{Irr}^\mathrm{n}$ there are $v_1, \ldots, v_n \in R$ such that $v_1 \sim_\# v_2 \sim_\# \cdots \sim_\# v_n$ and $(\alpha(v_1), \ldots, \alpha(v_n)) = a$. Thus Z is non-empty if and only if $\{(v_1, \ldots, v_n) \in R^n \ : \ v_1 \sim_\# v_2 \sim_\# \cdots \sim_\# v_n \wedge (\alpha(v_1), \ldots, \alpha(v_n)) \in Z\}$ is non-empty. Thus to decide whether $\mathcal{Q} \models \psi$, we first compute the automaton $\mathcal{B}$ that recognizes Z modulo $\sim_\#$, and then check whether the automaton accepts any word. ◀

We are now ready to prove Theorem C; that is decidability of the theory of the structure $\mathcal{M} = (\mathbf{Irr}, <, (\mathrm{M}_\varphi)_\varphi, (\mathrm{q})_{\mathrm{q} \in \mathbf{Irr}_{\mathrm{quad}}})$, where M_φ is defined for each $\mathcal{L}_{m,a}$-formula as $M_\varphi := \{\alpha \in \mathbf{Irr} \ : \ \mathcal{R}_\alpha \models \varphi\}$.

Proof of Theorem C. We just need to check that the relations we are adding are all recognizable modulo $\sim_\#$. By Lemma 15 the ordering $<$ is recognizable modulo $\sim_\#$. By Lemma 16, the singleton $\{q\}$ is is recognizable modulo $\sim_\#$ for every $q \in \mathbf{Irr}_{\mathrm{quad}}$. Since $M_\varphi = \alpha(R_\varphi)$, recognizability of M_φ modulo $\sim_\#$ follows from Theorem 31. ◀

We can add to $\mathcal{M}$ a predicate for every subset of $\mathbf{Irr}^\mathrm{n}$ that is recognizable modulo $\sim_\#$, and preserve the decidability of the theory. The reader can check that examples of subsets of $\mathbf{Irr}$ recognizable modulo $\sim_\#$ are the set of all $\alpha \in \mathbf{Irr}$ such that the terms in the continued fraction expansion of α are powers of 2, the set of all $\alpha \in \mathbf{Irr}$ such that the terms in the continued fraction expansion of α are in (or are not in) some fixed finite set, and the set of all $\alpha \in \mathbf{Irr}$ such that all even (or odd) terms in their continued fraction expansion are 1.

7 Automatically Proving Theorems about Sturmian Words

We have created an automatic theorem-prover based on the ideas and the decision algorithms outlined above, called Pecan [15]. We use Pecan to provide proofs of known and unknown results about characteristic Sturmian words. We begin by giving automated proofs for several classical result result about Sturmian words. We refer the reader to [12] for more information and traditional proofs of these results.

In the following, we assume that $a \in \mathbb{R}$ and i, j, k, n, m, p, s are a-Ostrowski representations. This can be expressed in Pecan as

```
Let a ∈ bco_standard.
Let i,j,k,n,m,p,s ∈ ostrowski(a).
```

We write $c_{a,0}(i)$ as `$C[i]` in Pecan.

▶ **Theorem 35.** *Characteristic Sturmian words are balanced and aperiodic.*

Proof of Theorem 35. To show that a characteristic Sturmian word $c_{\alpha,0}$ is balanced, note that it is sufficient to show that there is no palindrome w in $c_{\alpha,0}$ such that $0w0$ and $1w1$ are in $c_{\alpha,0}$ (see [12, Proposition 2.1.3]). We encode this in Pecan as follows. The predicate `palindrome(a,i,n)` is true when $c_{a,0}[i..i+n] = c_{a,0}[i..i+n]^R$. The predicate `factor_len(a,i,n,j)` is true when $c_{a,0}[i..i+n] = c_{a,0}[j..j+n]$.

```
Theorem ("Balanced", {
∀a. ¬(∃i,n. palindrome(a,i,n) ∧
            (∃j. factor_len(a,i,n,j) ∧ $C[j-1] = 0 ∧ $C[j+n] = 0) ∧
            (∃k. factor_len(a,i,n,k) ∧ $C[k-1] = 1 ∧ $C[k+n] = 1))
}).
```

Pecan takes `321.73` seconds to prove the theorem.

Encoding the property that a word is eventually periodic is straightforward:

```
eventually_periodic(a, p) :=
    p > 0 ∧ ∃n. ∀i. if i > n then $C[i] = $C[i+p]
```

The resulting automaton has 4941 states and 35776 edges, and takes `117.78` seconds to build. We then state the theorem in Pecan, which confirms the theorem is true.

```
Theorem ("Aperiodic", {
    ∀a. ∀p. if p > 0 then ¬eventually_periodic(a, p)
}) .
```

◀

Let $w \in \{0,1\}^*$. We let $\overline{w}$ denote the $\{0,1\}$-word obtained by replacing each 1 in w by 0 and each 0 in w by 1. A word $w \in \{0,1\}^*$ is an **antisquare** if $w = v\overline{v}$ for some $v \in \{0,1\}^*$. We define $A_O : (0,1) \setminus \mathbb{Q} \to \mathbb{N} \cup \{\infty\}$ to map an irrational α to the maximum order of any antisquare in $c_{\alpha,0}$ if such a maximum exists, and to ∞ otherwise. We let $A_L : (0,1) \setminus \mathbb{Q} \to \mathbb{N} \cup \{\infty\}$ map α to the maximum length of any antisquare in $c_{\alpha,0}$ if such a maximum exists and ∞ otherwise. Note that $A_L(\alpha) = 2A_O(\alpha)$.

We let w^R denote the reversal of a word w. We say a word w is a **palindrome** if $w = w^R$. A word $w \in \{0,1\}^*$ is an **antipalindrome** if $w = \overline{w^R}$. We set $A_P : (0,1) \setminus \mathbb{Q} \to \mathbb{N} \cup \{\infty\}$ to be the map that takes an irrational α to the maximum length of any antipalindrome in $c_{\alpha,0}$ if such a maximum, and to ∞ otherwise. We will use Pecan to prove that $A_O(\alpha), A_L(\alpha)$ and $A_P(\alpha)$ are finite for every α. While the quantities $A_O(\alpha)$, $A_P(\alpha)$ and $A_L(\alpha)$ can be arbitrarily large, we prove the new results that the length of the Ostrowski representations of these quantities is bounded, independent of α.

Let $\alpha \in (0,1)$ be irrational and $N \in \mathbb{N}$. Let $|N|_\alpha$ denote the length of the α-Ostrowski representation of N, that is the index of the last nonzero digit of α-Ostrowski representation of N, or 0 otherwise.

▶ **Theorem 36.** *For every irrational $\alpha \in (0,1)$,*

$$|A_O(\alpha)|_\alpha \le 4,\ |A_P(\alpha)|_\alpha \le 4,\ |A_L(\alpha)|_\alpha \le 6, A_O(\alpha) \le A_P(\alpha) \le A_L(\alpha) = 2A_O(\alpha).$$

There are irrational numbers $\alpha, \beta \in (0,1)$ such that $A_O(\alpha) = A_P(\alpha)$ and $A_P(\beta) = A_L(\beta)$.

Proof. Using Pecan, we create automata which compute A_O, A_P, and A_L:

$$A_O(\alpha,n) := \texttt{has\_antisquare}(\alpha,n) \wedge \forall m.\texttt{has\_antisquare}(\alpha,m) \implies m \le n$$
$$A_P(\alpha,n) := \texttt{has\_antipalindrome}(\alpha,n) \wedge \forall m.\texttt{has\_antipalindrome}(\alpha,m) \implies m \le n$$
$$A_L(\alpha,n) := \texttt{has\_antisquare\_len}(\alpha,n) \wedge \forall m.\texttt{has\_antisquare\_len}(\alpha,m) \implies m \le n$$

We build automata recognizing α-Ostrowski representations of at most 4 and 6 nonzero digits, called `has_4_digits`(n) and `has_6_digits`(n). Then we use Pecan to prove all the parts of the theorem by checking the following statement.

```
Theorem ("(i), (ii), (iii), and (iv)", {
∀a. has_4_digits(max_antisquare(a)) ∧
    has_4_digits(max_antipalindrome(a)) ∧
    has_6_digits(max_antisquare_len(a)) ∧
    max_antisquare(a) <= max_antipalindrome(a) ∧
    max_antipalindrome(a) <= max_antisquare_len(a)
}).
```

We also use Pecan to find examples of the equality: when $\alpha = [0; 3, 3, \overline{1}]$, we have $A_O(\alpha) = A_P(\alpha) = 2$, and when $\alpha = [0; 4, 2, \overline{1}]$, we have $A_P(\alpha) = A_L(\alpha) = 2$. ◀

▶ **Theorem 37.** *For every irrational $\alpha \in (0, 1)$, all antisquares and antipalindromes in $c_{\alpha,0}$ are either of the form $(01)^*$ or of the form $(10)^*$.*

Proof. We begin by creating a predicate called `is_all_01` stating that a subword $c_{\alpha,0}[i..i+n]$ is of the form $(01)^*$ or $(10)^*$. We do this simply stating that $c_{\alpha,0}[k] \neq c_{\alpha,0}[k+1]$ for all k with $i \leq k < i + n - 1$.

```
is_all_01(a,i,n) :=
    ∀k. if i <= k ∧ k < i+n-1 then $C[k] ≠ $C[k+1]
```

We can now directly state both parts of the theorem; Pecan proves both in `76.1` seconds.

```
Theorem ("All antisquares are of the form (01)^* or (10)^*", {
∀a. ∀i,n. if antisquare(a,i,n) then is_all_01(a,i,n)
}).

Theorem ("All antipalindromes are of the form (01)^* or (10)^*", {
∀a. ∀i,n. if antipalindrome(a,i,n) then is_all_01(a,i,n)
}).
```

◀

References

1. Jean-Paul Allouche and Jeffrey Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, 2003. doi:10.1017/CBO9780511546563.
2. Aseem Baranwal, Luke Schaeffer, and Jeffrey Shallit. Ostrowski-automatic sequences: Theory and applications. *Theor. Comput. Sci.*, 858:122–142, 2021.
3. Aseem R. Baranwal and Jeffrey Shallit. Critical exponent of infinite balanced words via the Pell number system. In *Combinatorics on words*, volume 11682 of *Lecture Notes in Comput. Sci.*, pages 80–92. Springer, Cham, 2019. doi:10.1007/978-3-030-28796-2.
4. Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc. Simon Stevin*, 1(2):191–238, 1994. Journées Montoises (Mons, 1992). URL: http://projecteuclid.org.proxy2.library.illinois.edu/euclid.bbms/1103408547.
5. J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr.)*, pages 1–11. Stanford Univ. Press, Stanford, Calif., 1962.
6. Daniel Goč, Dane Henshall, and Jeffrey Shallit. Automatic theorem-proving in combinatorics on words. *Internat. J. Found. Comput. Sci.*, 24(6):781–798, 2013. doi:10.1142/S0129054113400182.
7. Philipp Hieronymi. Expansions of the ordered additive group of real numbers by two discrete subgroups. *J. Symb. Log.*, 81(3):1007–1027, 2016. doi:10.1017/jsl.2015.34.

8 Philipp Hieronymi, Danny Nguyen, and Igor Pak. Presburger arithmetic with algebraic scalar multiplications. *arXiv:1805.03624*, 2018. arXiv:1805.03624.

9 Philipp Hieronymi and Alonza Terry Jr. Ostrowski numeration systems, addition, and finite automata. *Notre Dame J. Form. Log.*, 59:215–232, July 2014. doi:10.1215/00294527-2017-0027.

10 Bakhadyr Khoussainov and Mia Minnes. Three lectures on automatic structures. In *Logic Colloquium 2007*, volume 35 of *Lect. Notes Log.*, pages 132–176. Assoc. Symbol. Logic, La Jolla, CA, 2010.

11 Bakhadyr Khoussainov and Anil Nerode. *Automata Theory and Its Applications*. Birkhauser Boston, Inc., Secaucus, NJ, USA, 2001.

12 M. Lothaire. *Algebraic combinatorics on words.*, volume 90. Cambridge: Cambridge University Press, 2002.

13 Hamoon Mousavi. Automatic Theorem Proving in Walnut. *CoRR*, abs/1603.06017, 2016. arXiv:1603.06017.

14 Hamoon Mousavi, Luke Schaeffer, and Jeffrey Shallit. Decision algorithms for Fibonacci-automatic words, I: Basic results. *RAIRO Theor. Inform. Appl.*, 50(1):39–66, 2016. doi:10.1051/ita/2016010.

15 Reed Oei, Eric Ma, Christian Schulz, and Philipp Hieronymi. Pecan. available at https://github.com/ReedOei/Pecan, 2020.

16 Reed Oei, Eric Ma, Christian Schulz, and Philipp Hieronymi. Pecan: An Automated Theorem Prover for Automatic Sequences using Büchi automata. *arXiv*, 2021. arXiv:2102.01727.

17 Alexander Ostrowski. Bemerkungen zur Theorie der Diophantischen Approximationen. *Abh. Math. Semin. Univ. Hambg.*, 1(1):77–98, 250–251, Reprinted in *Collected Mathematical Papers*, Vol. 3, pp. 57–80., December 1922. doi:10.1007/BF02940581.

18 Andrew M. Rockett and Peter Szüsz. *Continued fractions*. World Scientific Publishing Co., Inc., River Edge, NJ, 1992. doi:10.1142/1725.

19 Thoralf Skolem. Über einige Satzfunktionen in der Arithmetik. *Skr. Norske Vidensk. Akad., Oslo, Math.-naturwiss. Kl.*, 7:1–28, 1931.

20 Faried Abu Zaid, Erich Grädel, and Frederic Reinhardt. Advice Automatic Structures and Uniformly Automatic Classes. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:20, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2017.35.

A Proofs from Section 2

Proof of Lemma 12. By the definition of R, there is $w_1w_2\cdots \in ((0|1)^*1(0|1)^*)^\omega$ such that $w = \#w_1\#w_2\#\cdots$. Since $w_i \in (0|1)^*1(0|1)^*$, we have that w_i is a $\{0,1\}$-word containing at least one 1. Let a_i be the natural number that $a_i = [w_i]_2$. Because w_i contains a 1, we must have $a_i \neq 0$. Thus w is a #-binary coding of the infinite continued fraction of the irrational $\alpha = [0; a_1, a_2, \dots]$. Uniqueness follows directly from the fact that both binary expansions and continued fraction expansions only represent one number. ◀

Proof of Lemma 14. Let $\mathcal{A}$ be a Büchi automaton recognizing X. We use Q to denote the set of states of $\mathcal{A}$. We create a new automaton $\mathcal{A}'$ that recognizes the zero-closure of X, as follows:

(Step 1) Start with the automata $\mathcal{A}$.

(Step 2) For each transition on the n-tuple $(\#, \dots, \#)$ from a state p to a state q, we add a new state $\mu(p,q)$ that loops to itself on the n-tuple $(0, \dots, 0)$ and transitions to state q on $(\#, \dots, \#)$. We add a transition from p to $\mu(p,q)$ on $(0, \dots, 0)$.

(Step 3) For every pair p, q of states of $\mathcal{A}$ for which p has a run to q on a word of the form $(0,\dots,0)^m(\#,\dots,\#)$ for some m, we add a transition from state p to a new state $\nu(p,q)$ on $(\#,\dots,\#)$, and for every transition out of state q, we create a copy of the transition that starts at state $\nu(p,q)$ instead. If any original run from state p to state q passes through a final state, we make $\nu(p,q)$ a final state.

(Step 4) Denote the resulting automaton by $\mathcal{A}'$ and its set of states by Q'.

We now show that $L(\mathcal{A}')$ is the zero-closure of X. We first show that the zero-closure is contained in $L(\mathcal{A}')$. Let $v \in X$ and $w \in R$ be such that $\alpha(v) = \alpha(w)$. Let $b = b_1 b_2 \cdots, c = c_1 c_2 \in (\{0,1\}^*)^\omega$ such that $C_\#(b) = v$ and $C_\#(c) = w$. Since $\alpha(v) = \alpha(w)$, we have that $[b_i]_2 = [c_i]_2$ for $i \in \mathbb{N}$. Therefore, for each $i \in \mathbb{N}$, the words b_i and c_i only differ by trailing zeroes. Let $s = s_1 s_2 \cdots \in Q^\omega$ be an accepting run of v on $\mathcal{A}$. We now transfer this run into an accepting run $s' = s'_1 s'_2 \cdots$ of w on $\mathcal{A}'$. For $i \in \mathbb{N}$, let $y(i)$ be the position of the i-th $(\#,\dots,\#)$ in v and let $z(i)$ be the position of the i-th $(\#,\dots,\#)$ in w. For each $i \in \mathbb{N}$, we define a sequence $s'_{z(i)+1} \cdots s'_{z(i+1)}$ of states of $\mathcal{A}'$ as follows:

1. If $|c_i| = |b_i|$, then $c_i = b_i$. We set

$$s'_{z(i)+1} \cdots s'_{z(i+1)} := s_{y(i)+1} \cdots s_{y(i+1)}.$$

2. If $|c_i| > |b_i|$, then $c_i = b_i(0,\dots,0)^{|c_i|-|b_i|}$. We set

$$s'_{z(i)+1} \cdots s'_{z(i+1)} := s_{y(i)+1} \cdots s_{y(i+1)-1} \underbrace{\mu(s_{y(i+1)-1}, s_{y(i+1)}) \cdots \mu(s_{y(i+1)-1}, s_{y(i+1)})}_{(|c_i|-|b_i|)\text{-times}} s_{y(i+1)}$$

   Thus the new run follows the old run up to $s_{y(i+1)-1}$ and then transitions to one of the newly added states in the Step 2. It loops on $(0,\dots,0)$ for $|c_i| - |b_i| - 1$-times before moving to $s_{y(i+1)}$.

3. If $|c_i| < |b_i|$, then $b_i = c_i(0,\dots,0)^{|b_i|-|c_i|}$. We set

$$s'_{z(i)+1} \cdots s'_{z(i+1)} := s_{y(i)+1} \cdots s_{y(i)+|c_i|} \nu(s_{y(i)+|c_i|}, s_{y(i+1)})$$

   The new run utilizes one of the newly added $(\#,\dots,\#)$ transitions and corresponding states added in Step 3.

The reader can now easily check that s' is an accepting run of w on $\mathcal{A}'$.

We now show that $L(\mathcal{A}')$ is contained in the zero-closure of X. We prove that the only accepting runs on $\mathcal{A}'$ are based on accepting runs on $\mathcal{A}$ with trailing zeroes either added or removed. Let $w = w_1 w_2 \cdots \in L(\mathcal{A}')$ and let $c = c_1 c_2 \cdots \in (\{0,1\}^*)^\omega$ be such that $C_\#(c) = w$. Let $s' = s'_1 s'_2 \cdots \in Q'^\omega$ be an accepting run of w on $\mathcal{A}'$. We construct $v \in X$ and a run $s = s_1 s_2 \cdots \in Q^\omega$ of w_2 on $\mathcal{A}$ such that $\alpha(v) = \alpha(w)$ and s is an accepting run of v. We start by setting $v := w_1 w_2 \cdots$ and $s := s'_1 s'_2 \cdots$. For each $i \in \mathbb{N}$, we replace w_i in v and s'_i in s as follows:

1. If $s'_i \in Q$, then we make no changes to s'_i and w_i.
2. If $s'_i = \mu(p,q)$ for some $p, q \in Q$, we delete the s'_i in s and delete w_i in v.
3. If $s_i = \nu(p,q)$ for some $p, q \in Q$, then we replace
   (a) s'_i by a run $t = t_1 \cdots t_{n+1}$ of $(0,\dots,0)^n(\#,...,\#)$ from p to q, and
   (b) w_i by $(0,\dots,0)^n(\#,...,\#)$.
   If $\nu(p,q)$ is a final state of $\mathcal{A}'$, we choose t such that it passed through a final state of $\mathcal{A}$.

It is clear that the resulting s is in Q^ω. The reader can check s is an accepting run of v on $\mathcal{A}$ and that $\alpha(v) = \alpha(w)$. Thus w is in the zero-closure of X. ◀

Proof of Lemma 19. The statement that $A^{\text{fin}} \subseteq A$, follows immediately from the definitions of A^{fin} and A and Fact 8. It is left to establish the ω-regularity of the two sets.

For (1): Let $B \supseteq A^{\text{fin}}$ be the set of all pairs (v, w) such that $v \in R$ and $v \sim_{\#} w$. Note that B is ω-regular. Let $(v, w) \in B$. Since v and w have infinitely many # characters and are aligned, there are unique $a = a_1 a_2 \cdots, b = b_1 b_2 \cdots \in (\{0,1\}^*)^\omega$ such that $C_{\#}(a) = v$, $C_{\#}(b) = w$ and $|a_i| = |b_i|$ for each $i \in \mathbb{N}$. Then by Fact 6, $(v, w) \in A^{\text{fin}}$ if and only if

(a) b has finitely many 1 characters;

(b) $b_1 <_{\text{colex}} a_1$;

(c) $b_i \leq_{\text{colex}} a_i$ for all $i > 1$;

(d) if $b_i = a_i$, then $b_{i-1} = 0$.

It is easy to check that all four conditions are ω-regular.

For (2): As above, let $(v, w) \in B$. Since v and w have infinitely many # characters and are aligned, there are unique $a = a_1 a_2 \cdots, b = b_1 b_2 \cdots \in (\{0,1\}^*)^\omega$ such that $C_{\#}(a) = v$, $C_{\#}(b) = w$ and $|a_i| = |b_i|$ for each $i \in \mathbb{N}$. Then by Fact 7, $(v, w) \in A$ if and only if

(e) $b_1 <_{\text{colex}} a_1$;

(f) $b_i \leq_{\text{colex}} a_i$ for all $i > 1$;

(g) if $b_i = a_i$, then $b_{i-1} = 0$;

(h) $b_i \neq a_i$ for infinitely many odd i.

Again, it is easy to see that all for conditions are ω-regular. ◀

Proof of Lemma 21. We first consider injectivity. By Fact 6 and Fact 7 a number in $\mathbb{N}$ or in $I_{\alpha(v)}$ only has one $\alpha(v)$-Ostrowski representation. So we need only explain why such a representation will only have one encoding in A_v^{fin} (respectively A_v). This follows from the uniqueness of binary representations up to the length of the representation, and from the fact that the requirement of having the # characters aligned with v determines the length of each binary-encoded coefficient.

For surjectivity we need only explain why an $\alpha(v)$-Ostrowski representation can always be encoded into a string in A_v^{fin} (respectively A_v). It suffices to show that the requirement of having the # characters aligned with v will never result in needing to fit the binary encoding of a number into too few characters, i.e. that it will never result in having to encode a natural number n in binary in fewer than $1 + \lfloor \log_2 n \rfloor$ characters. Since the function $1 + \lfloor \log_2 n \rfloor$ is monotone increasing, we can encode any natural number below n in k characters if we can encode n in binary in k characters. However, by Fact 6 and Fact 7, the coefficients in an $\alpha(v)$-Ostrowski representation never exceed the corresponding coefficients in the continued fraction for $\alpha(v)$, i.e. $b_n \leq a_n$. ◀

Proof of Lemma 23. Recognizing $\mathbf{0}_*$ is trivial, as the Ostrowski representations of 0 are of the form $0 \cdots 0$ for all irrational α. Thus $\mathbf{0}_*$ is just the relation

$$\{(v, w) \ : v \in R, w \text{ is } v \text{ with all 1 bits replaced by 0 bits}\}.$$

This is clearly ω-regular.

We now consider $\mathbf{1}_*$. Let $\alpha = [0; a_1, a_2, \dots]$ be an irrational number. If $a_1 > 1$, the α-Ostrowski representations of 1 are of the form $10 \cdots 0$. If $a_1 = 1$, the α-Ostrowski representations of 1 are of the form $010 \cdots 0$. Thus, in order to recognize $\mathbf{1}_*$, we only need to be able to recognize if a number in binary representation is 0, 1, or greater than 1. Of course, this is easily done on a Büchi automaton. ◀

Proof of Lemma 24. By Fact 8, $O_v(s) = f_\alpha(\alpha(v)Z_v(s))$. Thus

$$\alpha(v)Z_v(s) - O_v(s) = \alpha(v)Z_v(s) - f_\alpha(\alpha(v)Z_v(s)),$$

which is an integer by the definition of f. By the definition of $\mathbf{1}_v$ and by Fact 8, we know $O_v(\mathbf{1}_v) = f_\alpha(\alpha)$ is the unique element of $I_{\alpha(v)}$ that differs from $\alpha(v)$ by an integer. If $0 < \alpha(v) < \frac{1}{2}$, then $-\alpha(v) < \alpha(v) < 1 - \alpha(v)$. Thus in this case, $\alpha(v) \in I_{\alpha(v)}$ and $O_v(\mathbf{1}_v) = \alpha(v)$. When $\frac{1}{2} < \alpha(v) < 1$, then $-\alpha < \alpha - 1 < 1 - \alpha$. Therefore $\alpha(v) - 1 \in I_{\alpha(v)}$ and $O_v(\mathbf{1}_v) = \alpha(v) - 1$. ◀

B Proofs from Section 3

Proof of Lemma 28. First, let v, s_1, s_2, s_3 be such that $s_1, s_2, s_3 \in A_v^{\text{fin}}$. We claim that on this domain, $(s_1, s_2, s_3) \in \oplus_v$ if and only if $(s_1, s_2, s_3) \in \oplus_v^{\text{fin}}$. By Fact 8 we know that for all $s \in A_v^{\text{fin}}$

$$\alpha(v)Z_v(s) - O_v(s) \equiv 0 \pmod 1. \tag{3}$$

Let $(s_1, s_2, s_3) \in \oplus_v^{\text{fin}}$. Then by (3)

$$\begin{aligned} O_v(s_3) &\equiv \alpha(v)Z_v(s_3) \pmod 1 \\ &= \alpha(v)Z_v(s_1) + \alpha(v)Z_v(s_2) \\ &\equiv O_v(s_1) + O_v(s_2) \pmod 1. \end{aligned}$$

Thus $(s_1, s_2, s_3) \in \oplus_v$.

Now suppose that $(s_1, s_2, s_3) \in \oplus_v$. Then by (3) and the definition of $\oplus$, we obtain that $\alpha(v)Z(s_1) + \alpha(v)Z(s_2) \equiv \alpha(v)Z(s_3) \pmod 1$. However, then $\alpha(v)(Z(s_1) + Z(s_2) - Z(s_3)) \equiv 0 \pmod 1$. Since α is irrational, we obtain $Z(s_1) + Z(s_2) - Z(s_3) = 0$. Thus $(s_1, s_2, s_3) \in \oplus_v^{\text{fin}}$.

Thus for each $v \in R$, we have $\oplus_v \cap (A_v^{\text{fin}})^3 = \oplus_v^{\text{fin}}$. Let $v \in R$. We observe that the set $O_v(A_v^{\text{fin}})$ is dense in $O_v(A_v)$. Since addition is continuous, it follows that $O_v(\oplus_v^{\text{fin}})$ is dense in $O_v(\oplus_v)$. Since the graph of a continuous function is closed, the topological closure of $O_v(\oplus_v^{\text{fin}})$ is $O_v(\oplus_v)$. Thus $\oplus$ is ω-regular by Corollary 26. ◀

C Proofs from Section 4

In this section we present the proof of Lemma 30. We first state and prove three lemmas used in the proof.

▶ **Lemma 38.** *Let $v \in R$, and let $t_1, t_2, t_3 \in A_v$ be such that $t_1 \oplus_v t_2 = t_3$. Then*

$$O_v(t_1) + O_v(t_2) = \begin{cases} O_v(t_3) + 1 & \text{if } \mathbf{0}_v \prec_v t_1 \text{ and } t_3 \prec_v t_2; \\ O_v(t_3) - 1 & \text{if } t_1 \prec_v \mathbf{0}_v \text{ and } t_2 \prec_v t_3; \\ O_v(t_3) & \text{otherwise.} \end{cases}$$

Proof. For ease of notation, let $\alpha = \alpha(v)$, and set $x_i = O_v(t_i)$ for $i = 1, 2, 3$. By definition of $\oplus_v$, we have that $x_1, x_2, x_3 \in I_{\alpha(v)}$ with $x_1 + x_2 \equiv x_3 \pmod 1$. Note that $t_i \prec_v t_j$ if and only if $x_i < x_j$.

We first consider the case that $0 < x_1$ and $x_3 < x_2$. Thus $x_1 + x_2 > 1 - \alpha$. Note that

$$-\alpha = 1 - \alpha - 1 < x_1 + x_2 - 1 < (1 - \alpha) + (1 - \alpha) - 1 = 1 - 2\alpha < 1 - \alpha.$$

Thus $x_1 + x_2 - 1 \in I_\alpha$ and $x_3 = x_1 + x_2 - 1$.

Now assume that $x_1 < 0$ and $x_2 < x_3$. Then $x_1 + x_2 < -\alpha$, and therefore

$$1 - \alpha > x_1 + x_2 + 1 \geq (-\alpha) + (-\alpha) + 1 = (1 - \alpha) - \alpha > -\alpha.$$

Thus $x_1 + x_2 + 1 \in I_\alpha$ and hence $x_3 = x_1 + x_2 + 1$.

Finally consider that $0, x_1$ are ordered the same way as x_2, x_3. Since $x_1 + x_2 \equiv x_3 \pmod 1$, we know that $|x_1 - 0|$ and $|x_3 - x_2|$ differ by an integer k. If $k > 0$, would imply that one of these differences is at least 1, which is impossible within the interval I_α. Therefore $x_1 - 0 = x_3 - x_2$ and hence $x_3 = x_1 + x_2$. ◀

For $i \in \mathbb{N}$, set $\mathbf{i}_v := \underbrace{\mathbf{1}_v \oplus \cdots \oplus \mathbf{1}_v}_{i \text{ times}}$.

▶ **Lemma 39.** *The set $F := \{(v, s) \in A^{\text{fin}} \ : \ Z_v(s)\alpha(v) < 1\}$ is ω-regular, and for each $(v, s) \in F$*

$$O_v(s) = \begin{cases} \alpha(v)Z_v(s) & \text{if } (\alpha(v) + 1)Z_v(s) < 1; \\ \alpha(v)Z_v(s) - 1 & \text{otherwise.} \end{cases}$$

Proof. By Lemma 17, we can first consider the case that $\alpha(v) > \frac{1}{2}$. In this situation, F_v is just the set $\{\mathbf{0}_v, \mathbf{1}_v\}$, and hence obviously ω-regular.

Now assume that $\alpha(v) < \frac{1}{2}$. Let w be the $\prec_v^{\text{fin}}$-minimal element of A_v^{fin} with $w \prec_v \mathbf{0}_v$. We will show that

$$F_v = \{s \in A_v^{\text{fin}} \ : \ s \preceq_v^{\text{fin}} w\}.$$

Then ω-regularity of F follows then immediately.

Let $n \in \mathbb{N}$ be maximal such that $n\alpha(v) < 1$. It is enough to show that $Z_v(w) = n$. By Lemma 24, $O_v(\mathbf{1}_v) = \alpha(v)$. Hence $1\alpha(v), 2\alpha(v), \ldots, (n-1)\alpha(v) \in I_{\alpha(v)}$, but $n\alpha(v) > 1 - \alpha(v)$. Then for $i = 1, \ldots, n-1$

$$O_v(\mathbf{i}_v) = i\alpha(v), \ O_v(\mathbf{n}_v) = n\alpha(v) - 1 < 0.$$

So $\mathbf{i}_v \succeq \mathbf{0}_v$ for $i = 1, \ldots, n$, but $\mathbf{n}_v \prec \mathbf{0}_v$. Thus $\mathbf{n}_v = w$ and $Z_v(w) = n$. ◀

▶ **Lemma 40.** *Let $v \in R$ and $t \in A_v^{\text{fin}}$. Then there is an $s \in F_v$ and $t' \in A_v^{\text{fin}}$ such that $t' \preceq_v 0$ and $t = t' \oplus_v s$. In particular, $A_v^{\text{fin}} = \{t \in A_v^{\text{fin}} \ : \ t \preceq_v \mathbf{0}_v\} \oplus_v F_v$.*

Proof of Lemma 40. Let $n \in \mathbb{N}$ be maximal such that $n\alpha < 1$. Let $t \in A_v^{\text{fin}}$. We need to find $s \in A_v^{\text{fin}}$ and $u \in F_v$ such that $t = s \oplus_v^{\text{fin}} v$. We can easily reduce to the case that $t \succ \mathbf{0}_v$ and $Z_v(t) > n$.

Let $i \in \{0, \ldots, n\}$ be such that $0 \geq O_v(t) - i\alpha > -\alpha$. Then let $s \in A_v^{\text{fin}}$ be such that $Z_v(s) = Z_v(t) - i$. Note $t = s \oplus_v^{\text{fin}} \mathbf{i}_v$. Thus we only need to show that $s \preceq \mathbf{0}_v$.

To see this, observe that by Lemma 39

$$O_v(s) + \alpha i \equiv O_v(s) + O_v(\mathbf{i}_v) \equiv O_v(t) \pmod 1.$$

Since $O_v(t) - i\alpha(v) \in I_{\alpha(v)}$, we know that $O_v(s) = O_v(t) - i\alpha(v) \leq 0$. Therefore $O_v(s) \preceq \mathbf{0}_v$. ◀

Proof of Lemma 30. Define $B \subseteq A^{\text{fin}}$ to be $\{(v, s) \in A^{\text{fin}} \ : \ s \preceq_v \mathbf{0}_v\}$. Clearly, B is ω-regular. We now define $\prec^B$ and $\oplus^B$ such that for each $v \in R$, the structure $(B_v, \prec_v^B, \oplus_v^B)$ is isomorphic to $(\mathbb{N}, <, +)$ under the map g_v defined as $g_v(s) = \alpha(v)Z_v(s) - O_v(s)$.

We define $\prec^B$ to be the restriction of $\prec^{\text{fin}}$ to B. That is, for $(v, s_1), (v, s_2) \in B$ we have $(v, s_1) \prec^B (v, s_2)$ if and only if $(v, s_1) \prec^{\text{fin}} (v, s_2)$.

It is immediate that $\prec^B$ is ω-regular, since both B and $\prec^{\text{fin}}$ are ω-regular.

We define $\oplus^B$ as follows:

$$(v, s_1) \oplus^B (v, s_2) = \begin{cases} (v, s_1 \oplus_v s_2) & \text{if } s_1 \oplus_v^{\text{fin}} s_2 \preceq_v \mathbf{0}_v; \\ (v, s_1 \oplus_v s_2 \oplus_v \mathbf{1}_v) & \text{otherwise.} \end{cases}$$

We now show that $g_v(s_1 \oplus_v^B s_2) = g_v(s_1) + g_v(s_2)$ for every $s_1, s_2 \in B_v$.

Let $(v, s_1), (v, s_2) \in B$. We first consider the case that $s_1 \oplus_v s_2 \preceq_v \mathbf{0}_v$. By Lemma 38, $O_v(s_1 \oplus_v s_2) = O_v(s_1) + O_v(s_2)$. Thus

$$\begin{aligned} g_v(s_1 \oplus_v^B s_2) &= g_v(s_1 \oplus_v s_2) \\ &= \alpha(v) Z_v(s_1 \oplus_v s_2) - O_v(s_1 \oplus_v s_2) \\ &= \alpha Z_v(s_1) + \alpha Z_v(s_2) - O_v(s_1) - O_v(s_2) \\ &= g_v(s_1) + g_v(s_2). \end{aligned}$$

Now suppose that $s_1 \oplus_v s_2 \succ_v \mathbf{0}_v$. Since $-\alpha(v) \leq O_v(s_1), O_v(s_2) \leq 0$, we get that

$$1 - \alpha(v) > O_v(s_1) + O_v(s_2) + \alpha(v) \geq -\alpha(v).$$

Thus by Lemma 24,

$$O_v(s_1 \oplus_v s_2 \oplus_v \mathbf{1}_v) = O_v(s_1) + O_v(s_2) + \alpha(v).$$

We obtain

$$\begin{aligned} g_v(s_1 \oplus_v^B s_2) &= g_v(s_1 \oplus_v s_2 \oplus_v \mathbf{1}_v) \\ &= \alpha Z_v(s_1 \oplus_v s_2 \oplus_v \mathbf{1}_v) - O_v(s_1 \oplus_v s_2 \oplus_v \mathbf{1}_v) \\ &= \alpha(v)\big(Z_v(s_1) + Z_v(s_2)\big) + \alpha(v) - O_v(s_1) - O_v(s_2) - \alpha(v) \\ &= g_v(s_1) + g_v(s_2). \end{aligned}$$

Since $s_1 \prec_v s_2$ if and only if $Z_v(s_1) < Z_v(s_2)$, we get that g_v is an isomorphism between $(B_v, \prec_v^B, \oplus_v^B)$ and $(\mathbb{N}, <, +)$.

Let C be defined by

$$\{(v, s, t) \in (\Sigma_{\#}^{\omega})^3 \ : \ (v, s) \in B \wedge (v, t) \in A\}.$$

Clearly C is ω-regular. Let $T_v : C_v \to [-\alpha(v), \infty) \subseteq \mathbb{R}$ map $(s, t) \mapsto g_v(s) + O_v(t)$.

Note that T_v is bijective for each $v \in R$, since every real number decomposes uniquely into a sum $n + y$, where $n \in \mathbb{Z}$ and $y \in I_v$.

We define an ordering $\prec_v^C$ on C_v lexicographically: $(s_1, t_1) \prec_v^C (s_2, t_2)$ if either

Table 1 Definitions of sets used in the proof.

Name	Definition
A	$\{(v, w) \ : \ v \in R, \ w \text{ is a } \#\text{-}v\text{-Ostrowski representation}\}$
A^{fin}	$\{(v, w) \ : \ v \in R, \ w \text{ is a } \#\text{-}v\text{-Ostrowski representation and eventually } 0\}$
B	$\{(v, s) \in A^{\text{fin}} \ : \ s \preceq_v \mathbf{0}_v\}$
C	$\{(v, s, t) \ : \ (v, s) \in B \wedge (v, t) \in A\}$

- $s_1 \prec_v^B s_2$, or
- $s_1 = s_2$ and $t_1 \prec_v t_2$.

The set

$$\{(v, s_1, t_1, s_2, t_2) \ : \ (s_1, t_1), (s_2, t_2) \in C_v \wedge (s_1, t_1) \prec_v^C (s_2, t_2)\}$$

is ω-regular. We can easily check that $(s_1, t_1) \prec_v^C (s_2, t_2)$ if and only if $T_v(s_1, t_1) < T_v(s_2, t_2)$.

Let $\mathbf{0}^B$ be $g_v^{-1}(0)$ and $\mathbf{1}^B$ be $g_v^{-1}(1)$. Let $\ominus^B$ be the (partial) inverse of $\oplus^B$. We define $\oplus^C$ for $(s_1, t_1), (s_2, t_2) \in C$ as follows:

$$(s_1, t_1) \oplus_v^C (s_2, t_2) = \begin{cases} (s_1 \oplus_v^B s_2 \ominus^B \mathbf{1}^B, t_1 \oplus_v t_2) & \text{if } t_1 \prec \mathbf{0}_v \wedge t_2 \prec_v t_1 \oplus_v t_2; \\ (s_1 \oplus_v^B s_2 \oplus_v^B \mathbf{1}^B, t_1 \oplus_v t_2) & \text{if } \mathbf{0}_v \prec t_1 \wedge t_1 \oplus_v t_2 \prec_v t_2; \\ (s_1 \oplus_v^B s_2, t_1 \oplus_v t_2) & \text{otherwise.} \end{cases}$$

(Note that $\oplus^C$ is only a partial function, as the case where $s_1 = s_2 = \mathbf{0}^B$ and $t_1 \prec \mathbf{0}_v \wedge t_2 \prec_v t_1 \oplus_v t_2$ is outside of the domain of $\ominus^B$.) It is easy to check that $\oplus^C$ is ω-regular. It follows directly from Lemma 38 that

$$T_v((s_1, t_1) \oplus_v^C (s_2, t_2)) = T_v((s_1, t_1)) + T_v((s_2, t_2)).$$

Thus for each $v \in R$, the function T_v is an isomorphism between $(C_v, \prec_v^C, \oplus_v^C)$ and $([-\alpha(v), \infty), <, +)$. To finish the proof, it is left to establish the ω-regularity of the following two sets:

1. $\{(v, s, t) \in C \ : \ T_v(s, t) \in \mathbb{N}\}$,
2. $\{(v, s, t) \in C \ : \ T_v(s, t) \in \alpha(u)\mathbb{N}\}$.

For (1), observe that the set $T_v^{-1}(\mathbb{N})$ is just the set $\{(s, t) \in C_v : t = \mathbf{0}_v\}$.

For (2), consider the following two sets:

- $U_1 = \{(v, s, t) \in C \ : \ s = t\}$,
- $U_2 = \{(v, \mathbf{0}_v, t) \in C \ : \ t \in F_v\}$.

Let $\mathbf{1}_v^C$ be $T_v^{-1}(1)$. Set

$$\begin{aligned} U := \{(v, (s_1, t_1) \oplus_v^c (\mathbf{0}_v, t_2)) \ : \ (v, s_1, t_1) \in U_1, (v, \mathbf{0}_v, t_2) \in U_2, t_2 \succeq 0\} \\ \cup \{(v, (s_1, t_1) \oplus_v^c (\mathbf{0}_v, t_2) \oplus \mathbf{1}_v^C) \ : \ (v, s_1, t_1) \in U_1, (v, \mathbf{0}_v, t_2) \in U_2, t_2 \prec 0\} \end{aligned}$$

The set U is clearly ω-regular, since both U_1 and U_2 are ω-regular. We now show that $T_v(U) = \alpha(v)\mathbb{N}$.

Let $(v, s, s) \in U_1$ and $(v, \mathbf{0}_v, t) \in U_2$. If $t \succeq \mathbf{0}_v$, then by Lemma 39

$$\begin{aligned} T_v((s, s) \oplus_C (\mathbf{0}_v, t)) &= T_v(s, s) + T_v(\mathbf{0}_v, t) \\ &= \alpha(v)Z_v(s) - O_v(s) + O_v(s) + O_v(t) \\ &= \alpha(v)Z_v(s) + \alpha(v)Z_v(t) = \alpha(v)Z_v(s \oplus_v t). \end{aligned}$$

Table 2 A list of the maps and their domains and codomains.

Map	Domain	Codomain
α	R	**Irr**
O_v	A_v	$I_{\alpha(v)}$
Z_v	A_v^{fin}	$\mathbb{N}$
$g_v := \alpha(v)Z_v - O_v$	B_v	$\mathbb{N}$
$T_v := g_v + O_v$	C_v	$[-\alpha(v), \infty) \subseteq \mathbb{R}$

If $t \prec \mathbf{0}_v$, then by Lemma 39

$$\begin{aligned} T_v((s,s) \oplus_v^C (\mathbf{0}_v, t) \oplus_v^C \mathbf{1}_v^C) &= T_v(s,s) + T_v(\mathbf{0}_v, t) + 1 \\ &= \alpha(v)Z_v(s) - O_v(s) + O_v(s) + O_v(t) + 1 \\ &= \alpha(v)Z_v(s) + \alpha(v)Z_v(t) = \alpha(v)Z_v(s \oplus_v t). \end{aligned}$$

Thus $T_v(U) \subseteq \alpha(v)\mathbb{N}$. By Lemma 40, $T_v(U) = \alpha(v)\mathbb{N}$. ◀

Games, Mobile Processes, and Functions

Guilhem Jaber
Université de Nantes, France

Davide Sangiorgi
Università di Bologna, Italy
Inria Sophia Antipolis, France

Abstract

We establish a tight connection between two models of the λ-calculus, namely Milner's encoding into the π-calculus (precisely, the Internal π-calculus), and operational game semantics (OGS). We first investigate the operational correspondence between the behaviours of the encoding provided by π and OGS.

We do so for various LTSs: the standard LTS for π and a new "concurrent" LTS for OGS; an "output-prioritised" LTS for π and the standard alternating LTS for OGS. We then show that the equivalences induced on λ-terms by all these LTSs (for π and OGS) coincide.

These connections allow us to transfer results and techniques between π and OGS. In particular we import up-to techniques from π onto OGS and we derive congruence and compositionality results for OGS from those of π. The study is illustrated for call-by-value; similar results hold for call-by-name.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases λ-calculus, π-calculus, game semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.25

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-03407123` [21]

Funding *Davide Sangiorgi*: supported by the MIUR-PRIN project "Analysis of Program Analyses" (ASPRA, ID 201784YSZ5_004).

1 Introduction

The topic of the paper is the comparison between *Operational Game semantics* (OGS) and the *π-calculus*, as generic models or frameworks for the semantics of higher-order languages.

Game semantics [4, 20] provides intentional models of higher-order languages, where the denotation of a program brings up its possible interactions with the surrounding context. Distinct points of game semantics are the rich categorical structure and the emphasis on compositionality. Game semantics provides a modular characterization of higher-order languages with computational effects like control operators [25], mutable store [3, 5] or concurrency [15, 27]. This gives rise to the "Semantic Cube" [2], a characterization of the absence of such computational effects in terms of appropriate restrictions on the interactions, with conditions like *alternation, well-bracketing, visibility* or *innocence*. For instance, well-bracketing corresponds to the absence of control operators like call/cc.

Game semantics has spurred Operational Game Semantics (OGS) [16, 23, 24, 28, 30], as a way to describe the interactions of a program with its environment by embedding programs into appropriate configurations and then defining rules that turn such configurations into an LTS. Besides minor differences on the representation of causality between actions, the main distinction with "standard" game semantics is in the way in which the denotation of programs is obtained: via an LTS, rather than, compositionally, by induction on the structure of the programs (or their types). It is nonetheless possible to establish a formal correspondence between these two representations [30].

OGS is particularly effective on higher-order programs. To avoid being too intensional, functional values exchanged between the program and its environment are represented as atoms, seen as free variables. Therefore OGS configurations include open terms. The basic actions in the LTS produced by OGS represent the calls and returns of functions between a program and its environment. The OGS semantics has been shown fully-abstract, that is, to characterize observational equivalence, for a wide class of programming languages, including effectful subsets of ML [22, 28], fragments of Java [24], aspect-oriented programs [23]. The conditions in the above-mentioned Semantic Cube (alternation, well-bracketing, etc.) equally apply to OGS.

In this paper, we consider forms of OGS for the pure untyped call-by-value λ-calculus, which enforce some of such conditions. Specifically we consider: an *Alternating* OGS, where only one term can be run at a time, and the control on the interactions alternates between the term and the environment; and a *Concurrent* OGS, where multiple terms can be run in parallel.

The π-calculus is the paradigmatical name-passing calculus, that is, a calculus where names (a synonym for "channels") may be passed around. In the literature about the π-calculus, and more generally in Programming Language theory, Milner's work on functions as processes [33], which shows how the evaluation strategies of *call-by-name λ-calculus* and *call-by-value λ-calculus* [1, 35] can be faithfully mimicked, is generally considered a landmark. The work promotes the π-calculus to be a model for higher-order programs, and provides the means to study λ-terms in contexts other than the purely sequential ones and with the instruments available to reason about processes. In the paper, π-calculus is actually meant to be the *Internal π-calculus* (Iπ), a subset of the original π-calculus in which only fresh names may be exchanged among processes [41]. The use of Iπ avoids a few shortcomings of Milner's encodings, notably for call-by-value; e.g., the failure of the β_v rule (i.e., the encodings of $(\lambda x.\, M)V$ and $M\{^V/_x\}$ may be behaviourally distinguishable in π).

Further investigations into Milner's encodings [11,42] have revealed what is the equivalence induced on λ-terms by the encodings, whereby two λ-terms are equal if their encodings are behaviourally equivalent (i.e., bisimilar) Iπ terms. In call-by-value, this equivalence is *eager normal-form bisimilarity* [29], a tree structure proposed by Lassen (and indeed sometimes referred to as "Lassen's trees") as the call-by-value counterpart of Böhm Trees (or Lévy-Longo Trees).

In a nutshell, when used to give semantics to a language, major strengths of the π-calculus are its algebraic structure and the related algebraic properties and proof techniques; major strengths of OGS are its proximity to the source language – the configurations of OGS are built directly from the terms of the source language, as opposed to an encoding as in the π-calculus – and its flexibility – the semantics can be tuned to account for specific features of the source language like control operators or references.

The general goal of this paper is to show that there is a tight and precise correspondence between OGS and π-calculus as models of programming languages and that such a correspondence may be profitably used to take advantage of the strengths of the two models. We carry out the above program in the specific case of (untyped) call-by-value λ-calculus, Λ_{V}, which is richer and (as partly suggested above) with some more subtle aspects than call-by-name. However similar results also hold for call-by-name; see [21] for the technical details for more comments on it. Analogies and similarities between game semantics and π-calculus have been pointed out in various papers in the literature (e.g., [6,19]; see Section 9), and used to, e.g., explain game semantics using π-like processes, and enhance type systems for π-terms. In this paper, in contrast, we carry out a direct comparison between the two models, on their interpretation of functions.

We take the (arguably) canonical representations of Λ_V into Iπ and OGS. The latter representation is Milner's encoding, rewritten in Iπ. We consider two variant behaviours for the Iπ terms, respectively produced by the ordinary LTS of Iπ, and by an "output-prioritised" LTS, opLTS, in which input actions may be observed only in the absence of outputs and internal actions. Intuitively, the opLTS is intended to respect sequentiality constraints in the Iπ terms: an output action stands for an ongoing computation (for instance, returning the result of a previous request) whereas an input action starts a new computation (for instance, a request of a certain service); therefore, in a sequential system, an output action should have priority over input actions. For OGS, the Λ_V representation is the straightforward adaptation of the OGS representations of typed λ-calculi in the literature, e.g., [28].

We then develop a thorough comparison between the behaviours of the OGS and Iπ representations. For this we define a mapping from OGS configurations to Iπ processes. We also exploit the fact that, syntactically, the actions in the OGS and Iπ LTSs are the same. We derive a tight correspondence between the two models, which allows us to transfer techniques and to switch freely between the two models in the analysis of the OGS and Iπ representations of Λ_V, so to establish new results or obtain new proofs. On these aspects, our main results are the following:

1. We show that the representation of Λ_V in the Alternating OGS is behaviourally the same as the representation in Iπ assuming the opLTS. Thus the semantics on λ-terms induced by the OGS and Iπ representations coincide. The same results are obtained between the Concurrent OGS and Iπ under its ordinary LTS.
2. We transfer "bisimulation up-to techniques" for Iπ, notably a form of "up-to context", onto (Concurrent) OGS. The result is a powerful technique, called "up-to composition" that allows us to split an OGS configuration into more elementary configurations during the bisimulation game.
3. We show that the semantics induced on Λ_V by the Alternating and by the Concurrent OGS are the same, both when the equality in OGS is based on traces and when it is based on bisimulation. In other words, all the OGS views of Λ_V (Alternating or Concurrent, traced-based or bisimulation-based) coincide. Moreover, we show that such induced semantics is the equality of Lassen's trees. We derive the result in two ways: one in which we directly import it from Iπ; the other in which we lift eager normal-form bisimulations into OGS bisimulations via the up-to-composition technique.
4. We derive congruence and compositionality properties for the OGS semantics, as well as a notion of tensor product over configurations that computes interleavings of traces.

The results about OGS in (2-4) are obtained exploitingthe mapping into Iπ and its algebraic properties and proof techniques, as well as the up-to-composition technique for OGS imported from Iπ.

Structure of the paper. Sections 2 to 5 contain background material: general notations, Iπ, Λ_V, the representations of Λ_V in the Alternating OGS (A-OGS) and in Iπ. The following sections contain the new material. In Section 6 we study the relationship between the two Λ_V representations, in Iπ using the output-prioritised LTS. In Section 7 we establish a similar relationship between a new Concurrent OGS (C-OGS) and Iπ using its ordinary LTS. We also transport up-to techniques onto OGS, and prove that all the semantics of Λ_V examined (OGS, Iπ, traces, bisimulations) coincide. We import compositionality results for OGS from Iπ in Section 8.

2 Notations

In the paper, we use various LTSs and behavioural relations for them, both for OGS and for the π-calculus. In this section, we introduce or summarise common notations.

We use a tilde, like in $\widetilde{a}$, for (possibly empty) tuples of objects (usually names). Let $K \xrightarrow{\mu}_{\mathtt{g}} K'$ be a generic LTS (for OGS or Iπ; the grammar for actions in the LTSs for OGS and Iπ will be the same). Actions, ranged over by μ, can be of the form $a(\widetilde{b})$, $\overline{a}(\widetilde{b})$, τ, and $(\widetilde{a})$, where τ, called *silent* or (*invisible*) action, represents an internal step in K, that is, an action that does not require interaction with the outside, and $(\widetilde{a})$ is a special action performed by abstractions in Iπ and initial configurations in OGS. If $\mu \neq \tau$ then μ is a *visible* action; we use ℓ to range over them. We sometimes abbreviate $\xrightarrow{\tau}_{\mathtt{g}}$ as $\longrightarrow_{\mathtt{g}}$. We write $\Longrightarrow_{\mathtt{g}}$ for the reflexive and transitive closure of $\xrightarrow{\tau}_{\mathtt{g}}$. We also write $K \overset{\mu}{\Longrightarrow}_{\mathtt{g}} K'$ if $K \Longrightarrow_{\mathtt{g}} \xrightarrow{\mu}_{\mathtt{g}} \Longrightarrow_{\mathtt{g}} K'$ (the composition of the three relations. Then $\overset{\widehat{\mu}}{\Longrightarrow}_{\mathtt{g}}$ is $\overset{\mu}{\Longrightarrow}_{\mathtt{g}}$ if $\mu \neq \tau$, and $\Longrightarrow_{\mathtt{g}}$ if $\mu = \tau$.

Traces, ranged over by s, are finite (and possibly empty) sequences of visible actions. If $s = \ell_1, \ldots, \ell_n$ $(n \geq 0)$, then $K \overset{s}{\Longrightarrow}_{\mathtt{g}} K'$ holds if there are $K_0, \ldots, K_n$ with $K_0 = K$, $K_n = K'$, and $K_i \overset{\ell_{i+1}}{\Longrightarrow}_{\mathtt{g}} K_{i+1}$ for $0 \leq i < n$; and $K \overset{s}{\Longrightarrow}_{\mathtt{g}}$ if there is K' with $K \overset{s}{\Longrightarrow}_{\mathtt{g}} K'$.

Two states K_1, K_2 of the LTS are *trace equivalent*, written $K_1 \eqsim_{\mathtt{g}} K_2$, if ($K_1 \overset{s}{\Longrightarrow}_{\mathtt{g}}$ iff $K_2 \overset{s}{\Longrightarrow}_{\mathtt{g}}$), for all s.

Similarly, *bisimilarity*, written $\approx_{\mathtt{g}}$, is the largest symmetric relation on the state of the LTS such that whenever $K_1 \approx_{\mathtt{g}} K_2$ then $K_1 \overset{\mu}{\Longrightarrow}_{\mathtt{g}} K_1'$ implies there is K_2' with $K_2 \overset{\widehat{\mu}}{\Longrightarrow}_{\mathtt{g}} K_2'$ and $K_1' \approx_{\mathtt{g}} K_2'$. For instance, in the I$\pi$ LTS $\xrightarrow{\mu}_{\pi}$ of Section 3.1, $P \eqsim_{\pi} Q$ means that the Iπ processes P and Q are trace equivalent, and $P \approx_{\pi} Q$ means that they are bisimilar.

▶ Remark 1 (bound names). In an action $a(\widetilde{b})$ or $\overline{a}(\widetilde{b})$ or $(\widetilde{b})$, name a is *free* whereas $\widetilde{b}$ are *bound*; the free and bound names of a trace are defined accordingly. Throughout the paper, in any statement (concerning OGS or Iπ), the bound names of an action or of a trace that appears in the statement are supposed to be all *fresh*; i.e., all distinct from each other and from the free names of the objects in the statement.

3 Background

3.1 The Internal π-calculus

The Internal π-calculus, Iπ, is, intuitively, a subset of the π-calculus in which all outputs are bound. This is syntactically enforced by having outputs written as $\overline{a}(\widetilde{b})$ (which in the π-calculus would be an abbreviation for $\boldsymbol{\nu}\widetilde{b}\,\overline{a}\langle\widetilde{b}\rangle$). All tuples of names in Iπ are made of pairwise distinct components. *Abstractions* are used to write name-parametrised processes, for instance, when writing recursive process definitions. The instantiation of the parameters of an abstraction B is done via the *application* construct $B\langle\widetilde{a}\rangle$. Processes and abstractions form the set of *agents*, ranged over by T. Lowercase letters $a, b, \ldots, x, y, \ldots$ range over the infinite set of names. The grammar of Iπ is thus:

$$\begin{array}{lcll} P & \triangleq & 0 \mid a(\widetilde{b}).\,P \mid \overline{a}(\widetilde{b}).\,P \mid \boldsymbol{\nu} a\,P \mid P_1 \mid P_2 \mid\, !a(\widetilde{b}).\,P \mid B\langle\widetilde{a}\rangle & \text{(processes)} \\ B & \triangleq & (\widetilde{a})\,P \mid \mathtt{K} & \text{(abstractions)} \end{array}$$

The operators have the usual meaning; we omit the standard definition of free names, bound names, and names of an agent, respectively indicated with $\mathtt{fn}(-)$, $\mathtt{bn}(-)$, and $\mathtt{n}(-)$.

In the grammar, $\mathtt{K}$ is a constant, used to write recursive definitions. Each constant $\mathtt{K}$ has a defining equation of the form $\mathtt{K} \triangleq (\tilde{x})\ P$, where $(\tilde{x})\ P$ is name-closed (that is, without free names); $\tilde{x}$ are the formal parameters of the constant. Replication could be avoided in the syntax since it can be encoded with recursion. However, its semantics is simple, and it is useful in encodings.

An application redex $((\widetilde{x})P)\langle\widetilde{a}\rangle$ can be normalised as $P\{\widetilde{a}/\widetilde{x}\}$. An agent is *normalised* if all such application redexes have been contracted. In the remainder of the paper *we identify an agent with its normalised expression.*

Since the calculus is polyadic, we assume a *sorting system* [32] to avoid disagreements in the arities of the tuples of names Being not essential, it will not be presented here.

Operational semantics and behavioural relations

In the LTS for Iπ, recalled in [21] transitions are of the form $T \xrightarrow{\mu}_{\pi} T'$, where the bound names of μ are fresh, i.e., they do not appear free in T.

Trace equivalence ($\eqsim_\pi$) and bisimilarity ($\approx_\pi$) have been defined in Section 2. We refer to [21] for the standard definition of *expansion*, written $\lesssim_\pi$. (The expansion relation $\lesssim_\pi$ is an asymmetric variant of $\approx_\pi$ in which, intuitively, $P \lesssim_\pi Q$ holds if $P \approx_\pi Q$ but also Q has at least as many τ-moves as P.) All behavioural relations are extended to abstractions by requiring ground instantiation of the parameters; this is expressed by means of a transition; e.g., the action $(\tilde{x})\ P \xrightarrow{(\tilde{x})}_{\pi} P$.

The "up-to" techniques

The "up-to" techniques allow us to reduce the size of a relation $\mathcal{R}$ to exhibit for proving bisimilarities. Our main up-to technique will be *up-to context and expansion* [40], which admits the use of contexts and of behavioural equivalences such as expansion to achieve the closure of a relation in the bisimulation game. So the bisimulation clause becomes:

- if $P\ \mathcal{R}\ Q$ and $P \xrightarrow{\mu} P''$ then there are a static context $C_{\mathtt{ctx}}$ and processes P' and Q' s.t. $P'' \ {}_{\pi}\gtrsim C_{\mathtt{ctx}}[P']$, $Q \stackrel{\widehat{\mu}}{\Longrightarrow} {}_{\pi}\gtrsim C_{\mathtt{ctx}}[Q']$ and $P'\ \mathcal{R}\ Q'$ $\quad (*)$

where a *static context* is a context of the form $\boldsymbol{\nu}\tilde{c}\,(R \mid [\cdot])$.

We will also employ: *bisimulation up-to* $\approx_\pi$ [31], whereby bisimilarity itself is employed to achieve the closure of the candidate relation during the bisimulation game; a variant of bisimulation up-to context and expansion, called *bisimulation up-to context and up-to* $({}_{\pi}\gtrsim, \approx_\pi)$, in which, in $(*)$, when μ is a visible action, expansion is replaced by the coarser bisimilarity, at the price of imposing that the static context $C_{\mathtt{ctx}}$ cannot interact with the processes P or Q. (This technique, as far as we know, does not appear in the literature.) Details on these techniques may be found in [21].

3.2 The Call-By-Value λ-calculus

The grammar of the untyped call-by-value λ-calculus, Λ_{V}, has values V, terms M, evaluation contexts E, and general contexts C:

$$\begin{array}{llcl@{\qquad}llcl} \text{Vals} & V & \triangleq & x \mid \lambda x.\, M & \text{ECtxs} & E & \triangleq & [\cdot] \mid VE \mid EM \\ \text{Terms} & M, N & \triangleq & V \mid MN & \text{Ctxs} & C & \triangleq & [\cdot] \mid \lambda x.\, C \mid MC \mid CM \end{array}$$

where $[\cdot]$ stands for the hole of a context. The call-by-value reduction $\to_{\mathtt{v}}$ has two rules:

$$\frac{}{(\lambda x.\, M)V \to_{\mathtt{v}} M\{V/x\}} \qquad \frac{M \to_{\mathtt{v}} N}{E[M] \to_{\mathtt{v}} E[N]}$$

In the following, we write $M \Downarrow M'$ to indicate that $M \Rightarrow_{\mathtt{v}} M'$ with M' an eager normal form, that is, either a value or a stuck call $E[xV]$.

4 Operational Game Semantics

We introduce the representation of Λ_{V} in OGS. The LTS produced by the embedding of a term intends to capture the possible interactions between this term and its environment. Values exchanged between the term and the environment are represented by names, akin to free variables, called *variable names* and ranged over by x, y, z. Continuations (i.e., evaluation contexts) are also represented by names, called *continuation names* and ranged over by p, q, r.

Actions μ have been introduced in Section 2. In OGS, we have five kinds of (visible) actions:

- *Player Answers* (PA), $\bar{p}(x)$, and *Opponent Answers* (OA), $p(x)$, that exchange a variable x through a continuation name p;
- *Player Questions* (PQ), $\bar{x}(y, p)$, and *Opponent Questions* (OQ), $x(y, p)$, that exchange a variable y and a continuation name p through a variable x;
- *Initial Opponent Questions* (IOQ), (p), that introduce the initial continuation name p.

▶ Remark 2. The denotation of terms is usually represented in game semantics using the notion of *pointer structure* rather than traces. A pointer structure is defined as a sequence of *moves*, together with a pointer from each move (but the initial one) to a previous move that "justifies" it. Taking a trace s, one can reconstruct this pointer structure in the following way: an action μ is *justified* by an action μ' if the free name of μ is bound by μ' in s (here we are taking advantage of the "freshness" convention on the bound names of traces, Remark 1).

Environments, ranged over by γ, maintain the association from names to values and evaluations contexts, and are partial maps. A single mapping is either of the form $[x \mapsto V]$ (the variable x is mapped onto the value V), or $[p \mapsto (E, q)]$ (the continuation name p is mapped onto the pair of the evaluation context E and the continuation q).

There are two main kinds of configurations F: *active configurations* $\langle M, p, \gamma, \phi\rangle$ and *passive configurations* $\langle \gamma, \phi\rangle$, where M is a term, p a continuation name, γ an environment and ϕ a set of names called its *name-support*. Names in $\mathrm{dom}(\gamma)$ are called *P-names*, and those in $\phi \backslash \mathrm{dom}(\gamma)$ are called *O-names*. So we obtain a *polarity function* pol_F associated to F, defined as the partial maps from ϕ to $\{O, P\}$ mapping names to their polarity. In the following, we only consider *valid configurations*, for which:

- $\mathrm{dom}(\gamma) \subseteq \phi$
- $\mathtt{fv}(M), p$ are O-names;
- for all $a \in \mathrm{dom}(\gamma)$, the names appearing in $\gamma(a)$ are O-names.

The LTS is introduced in Figure 1. It is called *Alternating*, since, forgetting the Pτ transition, it is bipartite between active configurations that perform Player actions and passive configurations that perform Opponent actions. Accordingly, we call Alternating the resulting OGS, abbreviated A-OGS. In the OA rule, E is "garbage-collected" from γ, a behavior corresponding to *linear continuations*.

The term of an active configuration determines the next transition performed. First, the term needs to be reduced, using the rule (Pτ). When the term is a value V, a Player Answer (PA) is performed, providing a fresh variable x to Opponent, while V is stored in γ at position x. Freshness is enforced using the disjoint union $\uplus$. When the term is a callback $E[xV]$, with p the current continuation name, a Player Question (PQ) at x is performed, providing two fresh names y, q to Opponent, while storing V at y and (E, p) at q in γ.

$$
\begin{array}{l|lll}
(P\tau) & \langle M,p,\gamma,\phi\rangle & \xrightarrow{\tau}_{\mathtt{a}} \langle N,p,\gamma,\phi\rangle & \text{when } M \to_{\mathtt{v}} N \\
(PA) & \langle V,p,\gamma,\phi\rangle & \xrightarrow{\bar{p}(x)}_{\mathtt{a}} \langle \gamma\cdot[x\mapsto V],\phi\uplus\{x\}\rangle & \\
(PQ) & \langle E[xV],p,\gamma,\phi\rangle & \xrightarrow{\bar{x}(y,q)}_{\mathtt{a}} \langle \gamma\cdot[y\mapsto V]\cdot[q\mapsto(E,p)],\phi\uplus\{y,q\}\rangle & \\
(OA) & \langle \gamma\cdot[q\mapsto(E,p)],\phi\rangle & \xrightarrow{q(x)}_{\mathtt{a}} \langle E[x],p,\gamma,\phi\uplus\{x\}\rangle & \\
(OQ) & \langle \gamma,\phi\rangle & \xrightarrow{x(y,p)}_{\mathtt{a}} \langle Vy,p,\gamma,\phi\uplus\{y,p\}\rangle & \text{when } \gamma(x)=V \\
(IOQ) & \langle [?\mapsto M],\phi\rangle & \xrightarrow{(p)}_{\mathtt{a}} \langle M,p,\varepsilon,\phi\uplus\{p\}\rangle &
\end{array}
$$

Figure 1 The LTS for the Alternating OGS (A-OGS).

$$
\mathcal{V}[\![V]\!] \triangleq (p)\ \overline{p}(y).\,\mathcal{V}^*[\![V]\!]\langle y\rangle \qquad \mathcal{V}^*[\![\lambda x.\,M]\!] \triangleq (y)\ !y(x,q).\,\mathcal{V}[\![M]\!]\langle q\rangle \qquad \mathcal{V}^*[\![x]\!] \triangleq (y)\ y\triangleright x
$$
$$
\mathcal{V}[\![MN]\!] \triangleq (p)\ \boldsymbol{\nu} q\ \big(\mathcal{V}[\![M]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} r\ \big(\mathcal{V}[\![N]\!]\langle r\rangle \mid r(w).\,\overline{y}(w',p').\,(w'\triangleright w \mid p'\triangleright p)\big)\big)
$$

Figure 2 The encoding of call-by-value λ-calculus into Iπ.

On passive configurations, Opponent has the choice to perform different actions. It can perform an Opponent Answer (OA) by interrogating an evaluation context E stored in γ. For this, Opponent provides a fresh variable x that is plugged into the hole of E, while the continuation name q associated to E in γ is restored. Opponent may also perform an Opponent Question (OQ), by interrogating a value V stored in γ. For this, Opponent provides a fresh variable y as an argument to V.

To build the denotation of a term M, we introduce an *initial configuration* associated with it, written $\langle [?\mapsto M],\phi\rangle$, with ϕ the set of free variables we start with. When this set is taken to be the free variables of M, we simply write it as $\langle M\rangle$. In the initial configuration, the choice of the continuation name p is made by performing an Initial Opponent question (IOQ). (Formally, initial configurations should be considered as passive configurations.)

5 The encoding of call-by-value λ-calculus into the π-calculus

We recall here Milner's encoding of call-by-value λ-calculus, transplanted into Iπ. The core of any encoding of the λ-calculus into a process calculus is the translation of function application. This becomes a particular form of parallel combination of two processes, the function, and its argument; β-reduction is then modelled as a process interaction.

As in OGS, so in Iπ the encoding uses *continuation names* $p,q,r,\ldots$, and *variable names* $x,y,v,w\ldots$. Figure 2 presents the encoding. Process $a\triangleright b$ represents a *link* (sometimes called forwarder; for readability we have adopted the infix notation $a\triangleright b$ for the constant $\triangleright$). It transforms all outputs at a into outputs at b thus the body of $a\triangleright b$ is replicated, unless a and b are continuation names:

$$
\triangleright \ \triangleq\ \begin{cases} (p,q).\,p(x).\,\overline{q}(y).\,y\triangleright x & \text{if } p,q \text{ are continuation names} \\ (x,y).\,!x(z,p).\,\overline{y}(w,q).\,(q\triangleright p \mid w\triangleright z) & \text{if } x,y \text{ are variable names} \end{cases}
$$

The equivalence induced on call-by-value λ-terms by their encoding into Iπ coincides with Lassen's *eager normal-form (enf) bisimilarity* [29]. That is, $\mathcal{V}[\![M]\!] \approx_\pi \mathcal{V}[\![N]\!]$ iff M and N are enf-bisimilar [11]. In proofs about the behaviour of the Iπ representation of λ-terms we sometimes follow [11] and use an optimisation of Milner's encoding, reported in [21].

Encoding of environments:

$$\begin{array}{rcl} \mathcal{V}[\![[y \mapsto V] \cdot \gamma']\!] & \triangleq & \mathcal{V}^*[\![V]\!]\langle y\rangle \mid \mathcal{V}[\![\gamma']\!] \\ \mathcal{V}[\![[q \mapsto (E,p)] \cdot \gamma']\!] & \triangleq & q(x).\, \mathcal{V}[\![E[x]]\!]\langle p\rangle \mid \mathcal{V}[\![\gamma']\!] \\ \mathcal{V}[\![\varepsilon]\!] & \triangleq & 0 \end{array}$$

Encoding of configurations:

$$\begin{array}{rcl} \mathcal{V}[\![\langle M, p, \gamma, \phi\rangle]\!] & \triangleq & \mathcal{V}[\![M]\!]\langle p\rangle \mid \mathcal{V}[\![\gamma]\!] \\ \mathcal{V}[\![\langle \gamma, \phi\rangle]\!] & \triangleq & \mathcal{V}[\![\gamma]\!] \\ \mathcal{V}[\![\langle [? \mapsto M], \phi\rangle]\!] & \triangleq & \mathcal{V}[\![M]\!] \end{array}$$

Figure 3 From OGS environments and configurations to Iπ.

6 Relationship between Iπ and A-OGS

To compare the A-OGS and Iπ representations of the (call-by-value) λ-calculus, we set a mapping from A-OGS configurations and environments to Iπ processes. The mapping is reported in Figure 3. It is an extension of Milner's encoding of the λ-calculus and is therefore indicated with the same symbol $\mathcal{V}$. The mapping uses a representation of environments γ as associative lists.

▶ Remark 3. The encoding of a configuration F with name-support ϕ does not depend on ϕ. This name-support ϕ is used in OGS both to enforce freshness of names, and to deduce the polarity of names, as represented by the function pol. And indeed, the process $\mathcal{V}[\![F]\!]$ has its set of free names included in ϕ, and uses P-names in outputs and O-names in inputs. The polarity property could be stated in π-calculus using i/o-sorting [34]. Indeed, a correspondence between arenas of game semantics (used to enforce polarities of moves) and sorting has been explored [18, 19].

6.1 Operational correspondence

The following theorems establish the operational correspondence between the A-OGS and Iπ representations. In Theorem 4, as well as in following theorems such as Theorems 5, 7, and 13, the appearance of the expansion relation ${}_\pi\gtrsim$ (in place of the coarser $\approx_\pi$), in the statement about silent actions, is essential, both to derive the statement in the theorems about visible actions and to use the theorems in up-to techniques for Iπ (more generally, in applications of the theorems in which one reasons about the number of steps performed).

▶ **Theorem 4.**

1. *If* $F \Longrightarrow_{\mathsf{a}} F'$, *then* $\mathcal{V}[\![F]\!] \Longrightarrow_\pi {}_\pi\gtrsim \mathcal{V}[\![F']\!]$;
2. *If* $F \overset{\ell}{\Longrightarrow}_{\mathsf{a}} F'$, *then* $\mathcal{V}[\![F]\!] \overset{\ell}{\Longrightarrow}_\pi \approx_\pi \mathcal{V}[\![F']\!]$.

▶ **Theorem 5.**

1. *If* $\mathcal{V}[\![F]\!] \Longrightarrow_\pi P$ *then there is* F' *such that* $F \Longrightarrow_{\mathsf{a}} F'$ *and* $P \,{}_\pi\gtrsim \mathcal{V}[\![F']\!]$;
2. *If* $\mathcal{V}[\![F]\!] \overset{\ell}{\Longrightarrow}_\pi P$ *and* ℓ *is an output, then there is* F' *such that* $F \overset{\ell}{\Longrightarrow}_{\mathsf{a}} F'$ *and* $P \,{}_\pi\gtrsim \mathcal{V}[\![F']\!]$;
3. *If* F *is passive and* $\mathcal{V}[\![F]\!] \overset{\ell}{\Longrightarrow}_\pi P$, *then there is* F' *such that* $F \overset{\ell}{\Longrightarrow}_{\mathsf{a}} F'$ *and* $P \approx_\pi \mathcal{V}[\![F']\!]$.

In Theorem 5, a clause is missing for input actions from $\mathcal{V}[\![F]\!]$ when F active. Indeed such actions are possible in Iπ, stemming from the (encoding of the) environment of F, whereas they are not possible in A-OGS. This is rectified in Section 6.2, introducing a constrained LTS for Iπ, and in Section 7, considering a concurrent OGS.

▶ **Corollary 6.** *If* $F \overset{s}{\Longrightarrow}_{\mathsf{a}}$ *then also* $\mathcal{V}[\![F]\!] \overset{s}{\Longrightarrow}_\pi$.

6.2 An output-prioritised Transition System

We define an LTS for Iπ in which input actions are visible only if no output can be consumed, either as a visible action or through an internal action (i.e., syntactically the process has no unguarded output). The new LTS, called *output-prioritised* and indicated as opLTS, is defined on the top of the ordinary one by means of the two rules below. A process P is *input reactive* if whenever $P \xrightarrow{\mu}_{\pi} P'$, for some μ, P' then μ is an input action.

$$\frac{P \xrightarrow{\mu}_{\pi} P' \quad P \text{ input reactive}}{P \xrightarrow{\mu}_{\mathsf{o}\pi} P'} \qquad \frac{P \xrightarrow{\mu}_{\pi} P'}{P \xrightarrow{\mu}_{\mathsf{o}\pi} P'} \mu \text{ is an output or } \tau \text{ action}$$

The opLTS captures an aspect of *sequentiality* in π-calculi: a free input prefix is to be thought of as a service offered to the external environment; in a sequential system such a service is available only if there is no ongoing computations due to previous interrogations of the server. An ongoing computation is represented by a τ-action, indicating a step of computation internal to the server, or an output, indicating either an answer to a client or a request to an external server. The constraint imposed by the new LTS could also be formalised compositionally, see [21].

Under the opLTS, the analogous of Theorem 4 continue to hold: in A-OGS configurations, input transitions only occur in passive configurations, and the encodings of passive configurations are input-reactive processes. However, now we have the full converse of Theorem 5 and, as a consequence, we can also establish the converse direction of Corollary 6.

▶ **Theorem 7.**

1. *If* $\mathcal{V}[\![F]\!] \stackrel{\tau}{\Longrightarrow}_{\mathsf{o}\pi} P$ *then there is* F' *such that* $F \Longrightarrow_{\mathsf{a}} F'$ *and* $P \; {}_{\pi}\!\gtrsim \mathcal{V}[\![F']\!]$ *;*
2. *If* $\mathcal{V}[\![F]\!] \stackrel{\ell}{\Longrightarrow}_{\mathsf{o}\pi} P$ *then there is* F' *such that* $F \stackrel{\ell}{\Longrightarrow}_{\mathsf{a}} F'$ *and* $P \approx_{\pi} \mathcal{V}[\![F']\!]$.

▶ **Corollary 8.** *For any configuration* F *and trace* s*, we have* $F \stackrel{s}{\Longrightarrow}_{\mathsf{a}}$ *iff* $\mathcal{V}[\![F]\!] \stackrel{s}{\Longrightarrow}_{\mathsf{o}\pi}$.

▶ Remark 9. We recall that, following Remark 1 on the usage of bound names, in Corollary 8 the bound names in s are fresh; thus they do not appear in F. (Similarly, in Theorems 5 and 7 for the bound names in ℓ).

For both results we first establish a correspondence result on strong transitions. See [21] for details.

▶ Remark 10. Corollary 8 relies on Theorems 4 and 7. The corollary talks about the opLTS of Iπ; however the theorems make use of the ordinary expansion relation $\lesssim_{\pi}$, that is defined on the ordinary LTS. Such uses of expansion can be replaced by expansion on the opLTS (defined as ordinary expansion but on the opLTS). For more details on this, see [21].

As a consequence of Corollary 8, trace equivalence is the same, on A-OGS configurations and on the encoding Iπ terms. Moreover, from Theorem 7 the same result holds under a bisimulation semantics. Further, since the LTS produced by A-OGS is deterministic, its trace semantics coincides with its bisimulation semantics. We can thus conclude as in Corollary 11. We recall that $\eqsim_{\mathsf{o}\pi}$ and $\approx_{\mathsf{o}\pi}$ are, respectively, trace equivalence and bisimilarity between Iπ processes in the opLTS; similarly for $\eqsim_{\mathsf{a}}$ and $\approx_{\mathsf{a}}$ between A-OGS terms.

▶ **Corollary 11.** *For any* F, F' *we have:* $F \eqsim_{\mathsf{a}} F'$ *iff* $\mathcal{V}[\![F]\!] \eqsim_{\mathsf{o}\pi} \mathcal{V}[\![F']\!]$ *iff* $F \approx_{\mathsf{a}} F'$ *iff* $\mathcal{V}[\![F]\!] \approx_{\mathsf{o}\pi} \mathcal{V}[\![F']\!]$.

Corollary 11 holds in particular when F is the initial configuration for a λ-term. That is, the equality induced on call-by-value λ-terms by their representation in A-OGS and in Iπ (under the opLTS) is the same, both employing traces and employing bisimulation to handle the observables for the two models.

$(P\tau)$	$\langle A \cdot [p \mapsto M], \gamma, \phi\rangle$	$\xrightarrow{\tau}_{\mathsf{c}}$	$\langle A \cdot [p \mapsto N], \gamma, \phi\rangle$	when $M \to_{\mathsf{v}} N$
(PA)	$\langle A \cdot [p \mapsto V], \gamma, \phi\rangle$	$\xrightarrow{\bar{p}(x)}_{\mathsf{c}}$	$\langle A, \gamma \cdot [x \mapsto V], \phi \uplus \{x\}\rangle$	
(PQ)	$\langle A \cdot [p \mapsto E[xV]], \gamma, \phi\rangle$	$\xrightarrow{\bar{x}(y,q)}_{\mathsf{c}}$	$\langle A, \gamma \cdot [y \mapsto V] \cdot [q \mapsto (E,p)], \phi \uplus \{y,q\}\rangle$	
(OA)	$\langle A, \gamma \cdot [p \mapsto (E,q)], \phi\rangle$	$\xrightarrow{p(x)}_{\mathsf{c}}$	$\langle A \cdot [q \mapsto E[x]], \gamma, \phi \uplus \{x\}\rangle$	
(OQ)	$\langle A, \gamma, \phi\rangle$	$\xrightarrow{x(y,p)}_{\mathsf{c}}$	$\langle A \cdot [p \mapsto Vy], \gamma, \phi \uplus \{y,p\}\rangle$	when $\gamma(x) = V$
(IOQ)	$\langle [? \mapsto M], \phi\rangle$	$\xrightarrow{(p)}_{\mathsf{c}}$	$\langle [p \mapsto M], \varepsilon, \phi \uplus \{p\}\rangle$	

Figure 4 The LTS for the Concurrent OGS.

▶ **Corollary 12.** *For any λ-terms M, N, we have:*
$\langle M\rangle \mathrel{\hat{=}}_{\mathsf{a}} \langle N\rangle$ *iff* $\langle M\rangle \approx_{\mathsf{a}} \langle N\rangle$ *iff* $\mathcal{V}[\![M]\!] \mathrel{\hat{=}}_{\mathsf{o}\pi} \mathcal{V}[\![N]\!]$ *iff* $\mathcal{V}[\![M]\!] \approx_{\mathsf{o}\pi} \mathcal{V}[\![N]\!]$.

From Theorem 5 and Corollary 8, it also follows that F and $\mathcal{V}[\![F]\!]$ are weakly bisimilar, on the union of the respective LTSs.

7 Concurrent Operational Game Semantics

In this section, we explore another way to derive an exact correspondence between OGS and Iπ, by relaxing the Alternating LTS for OGS so to allow multiple terms in configurations to run concurrently. We refer to the resulting OGS as the *Concurrent OGS*, briefly C-OGS (we recall that A-OGS refers to the Alternating OGS of Section 4).

We introduce *running terms*, ranged over by A, B, as finite mappings from continuation names to λ-terms. A *concurrent configuration* is a triple $\langle A, \gamma, \phi\rangle$ of a running term A, an environment γ, and a set of names ϕ. Moreover, the domains of A and γ must be disjoint. We extend the definition of the polarity function, considering names in the domain of both A and γ as Player names.

Passive and active configurations can be seen as special case of C-OGS configurations with zero and one running term, respectively. For this reason we still use F, G to range over C-OGS configurations. Moreover we freely take A-OGS configurations to be C-OGS configurations, and conversely for C-OGS configurations with zero and one running term, omitting the obvious syntactic coercions. Both the running term and the environment may be empty.

We present the rules of C-OGS in Figure 4. Since there is no more distinction between passive and active configurations, a given configuration can perform both Player and Opponent actions. Notice that only Opponent can add a new term to the running term A. A *singleton* is a configuration F whose P-support has only one element (that is, in C-OGS, F is either of the form $\langle [p \mapsto M], \varepsilon, \phi\rangle$, or $\langle \varepsilon, [x \mapsto V], \phi\rangle$, or $\langle \varepsilon, [p \mapsto (E,q)], \phi\rangle$).

In this and in the following section F, G ranges over C-OGS configurations, as reminded by the index "c" in the symbols for LTS and behavioural equivalence with which F, G appear (e.g., $\mathrel{\hat{=}}_{\mathsf{c}}$).

7.1 Comparison between C-OGS and Iπ

The encoding of C-OGS into Iπ is a simple adaptation of that for A-OGS. We only have to consider the new or modified syntactic elements of C-OGS, namely running terms and configurations; the encoding remains otherwise the same. The encoding of running term is:

$$\mathcal{V}[\![[p \mapsto M] \cdot A]\!] \stackrel{\text{def}}{=} \mathcal{V}[\![M]\!]\langle p\rangle \mid \mathcal{V}[\![A]\!] \qquad\qquad \mathcal{V}[\![\varepsilon]\!] \stackrel{\text{def}}{=} 0$$

The encoding of configurations is then defined as: $\mathcal{V}[\![\langle A, \gamma, \phi\rangle]\!] \stackrel{\text{def}}{=} \mathcal{V}[\![A]\!] \mid \mathcal{V}[\![\gamma]\!]$.

The results about operational correspondence between C-OGS and Iπ are as those between A-OGS and Iπ under the opLTS.

▶ **Theorem 13.**
1. *If* $F \Longrightarrow_{\mathsf{c}} F'$ *then* $\mathcal{V}[\![F]\!] \Longrightarrow_{\pi} {}_{\pi}\gtrsim \mathcal{V}[\![F']\!]$;
2. *if* $F \stackrel{\ell}{\Longrightarrow}_{\mathsf{c}} F'$ *then* $\mathcal{V}[\![F]\!] \stackrel{\ell}{\Longrightarrow}_{\pi} \approx_{\pi} \mathcal{V}[\![F']\!]$;
3. *the converse of (1), i.e. if* $\mathcal{V}[\![F]\!] \Longrightarrow_{\pi} P$ *then there is* F' *such that* $F \Longrightarrow_{\mathsf{c}} F'$ *and* $P \; {}_{\pi}\gtrsim \mathcal{V}[\![F']\!]$.
4. *the converse of (2), i.e. if* $\mathcal{V}[\![F]\!] \stackrel{\ell}{\Longrightarrow}_{\pi} P$ *then there is* F' *such that* $F \stackrel{\ell}{\Longrightarrow}_{\mathsf{c}} F'$ *and* $P \approx_{\pi} \mathcal{V}[\![F']\!]$.

▶ **Corollary 14.** *For any* C-OGS *configuration* F *and trace* s*, we have* $F \stackrel{s}{\Longrightarrow}_{\mathsf{c}}$ *iff* $\mathcal{V}[\![F]\!] \stackrel{s}{\Longrightarrow}_{\pi}$.

From Corollary 14 and Theorem 13, we derive:

▶ **Lemma 15.** *For any* F, F' *we have:*
1. $F_1 \eqsim_{\mathsf{c}} F_2$ *iff* $\mathcal{V}[\![F_1]\!] \eqsim_{\pi} \mathcal{V}[\![F_2]\!]$;
2. $F_1 \approx_{\mathsf{c}} F_2$ *iff* $\mathcal{V}[\![F_1]\!] \approx_{\pi} \mathcal{V}[\![F_2]\!]$.

To derive the full analogous of Corollary 12, we now show that, on the Iπ representation of λ-terms, trace equivalence is the same as bisimilarity. This result needs a little care: it is known that on deterministic LTSs bisimilarity coincides with trace equivalence. However, the behaviour of the Iπ representation of a C-OGS configuration need not be deterministic, because there could be multiple silent transitions as well as multiple output transitions (for instance, in C-OGS rule OQ may be applicable to different terms).

▶ **Lemma 16.** *For any* M, N *we have:* $\mathcal{V}[\![M]\!] \eqsim_{\pi} \mathcal{V}[\![N]\!]$ *iff* $\mathcal{V}[\![M]\!] \approx_{\pi} \mathcal{V}[\![N]\!]$.

The proof uses the "bisimulation up-to context and up-to $({}_{\pi}\gtrsim, \approx_{\pi})$" technique. We can finally combine Lemmas 16 and 15 to derive that the C-OGS and Iπ semantics of λ-calculus coincide, both for traces and for bisimilarity.

▶ **Corollary 17.** *For all* M, N *we have:* $\langle M\rangle \eqsim_{\mathsf{c}} \langle N\rangle$ *iff* $\langle M\rangle \approx_{\mathsf{c}} \langle N\rangle$ *iff* $\mathcal{V}[\![M]\!] \eqsim_{\pi} \mathcal{V}[\![N]\!]$ *iff* $\mathcal{V}[\![M]\!] \approx_{\pi} \mathcal{V}[\![N]\!]$.

More details on proofs may be found in [21].

7.2 Tensor Product

We now introduce a way of combining configurations, which corresponds to the notion of tensor product of arenas and strategies in (denotational) game semantics.

▶ **Definition 18.** *Two concurrent configurations* F, G *are said to be* compatible *if their polarity functions* $\mathrm{pol}_F, \mathrm{pol}_G$ *are compatible – that is, for all* $a \in \mathrm{dom}(\mathrm{pol}_F) \cap \mathrm{dom}(\mathrm{pol}_G)$*, we have* $\mathrm{pol}_F(a) = \mathrm{pol}_G(a)$.

▶ **Definition 19.** *For compatible configurations* $F = \langle A, \gamma, \phi\rangle$ *and* $G = \langle B, \delta, \phi'\rangle$*, the* tensor product $F \otimes G$ *is defined as* $F \otimes G \triangleq \langle A \cdot B, \gamma \cdot \delta, \phi \cup \phi'\rangle$

The polarity function of $F \otimes G$ is then equal to $\mathrm{pol}_F \cup \mathrm{pol}_G$, and $\mathcal{V}[\![F_1 \otimes F_2]\!] \equiv \mathcal{V}[\![F_1]\!] \mid \mathcal{V}[\![F_2]\!]$, where $\equiv$ is the standard structural congruence of π-calculi. In the following, we write $\mathtt{inter}(s_1, s_2)$ for the set of traces obtained from an interleaving of the elements in the sequences s_1 and s_2.

▶ **Lemma 20.** *Suppose F_1, F_2 are compatible concurrent configurations. The set of traces generated by $F_1 \otimes F_2$ is the union of the sets of interleaving $\mathtt{inter}(s_1, s_2)$, for $F_1 \stackrel{s_1}{\Longrightarrow}_{\mathsf{c}}$ and $F_2 \stackrel{s_2}{\Longrightarrow}_{\mathsf{c}}$.*

The tensor product of A-OGS configurations is defined similarly, with the additional hypothesis that at most one of the two configurations is active, in order for their tensor product to be a valid A-OGS configuration. The details can be found in [21].

7.3 Up-to techniques for games

We introduce up-to techniques for C-OGS, which allow, in bisimulation proofs, to split two C-OGS configurations into separate components and then to reason separately on these. These up-to techniques are directly imported from Iπ. Abstract settings for up-to techniques have been developed, see [36, 37]; but we cannot derive the OGS techniques from them because these settings are specific to first-order LTS (i.e., CCS-like, without binders within actions).

The new techniques are then used to prove that C-OGS and A-OGS yield the same semantics on λ-terms; a further application is in Section 7.5, discussing eager normal-form bisimilarity.

A relation $\mathcal{R}$ on configuration is *well-formed* if it relates configurations with the same polarity function. Below, all relations on configurations are meant to be well formed. Given a well-formed relation $\mathcal{R}$ we write:

- $\mathcal{R}^{|}$ for the relation $\{(F_1, F_2) \ : \ \exists\ G \text{ s.t. } F_i = F_i' \otimes G \ (i = 1, 2) \text{ and } F_1' \ \mathcal{R}\ F_2'\}$.
- $\mathcal{R}^{|\star}$ for the reflexive and transitive closure of $\mathcal{R}^{|}$. Thus from $F_1\ \mathcal{R}\ G_1$ and $F_2\ \mathcal{R}\ G_2$ we obtain $(F_1 \otimes F_2)\ \mathcal{R}^{|\star}\ (G_1 \otimes F_2)\ \mathcal{R}^{|\star}\ G_1 \otimes G_2$.
- $\Rightarrow_{\mathsf{c}} \mathcal{R}^{|\star}\ {}_{\mathsf{c}}\Leftarrow$ for the closure of $\mathcal{R}^{|\star}$ under reductions. That is, $F_1 \Rightarrow_{\mathsf{c}} \mathcal{R}^{|\star}\ {}_{\mathsf{c}}\Leftarrow F_2$ holds if there are F_i', $i = 1, 2$ with $F_i \Longrightarrow_{\mathsf{c}} F_i'$ and $F_1'\ \mathcal{R}^{|\star}\ F_2'$. (As $\Longrightarrow_{\mathsf{c}}$ is reflexive, we may have $F_i = F_i'$.)

▶ **Definition 21.** *A relation $\mathcal{R}$ on configurations is a* bisimulation up-to reduction and composition *if whenever $F_1\ \mathcal{R}\ F_2$:*

1. *if $F_1 \xrightarrow{\mu}_{\mathsf{c}} F_1'$ then there is F_2' such that $F_2 \stackrel{\widehat{\mu}}{\Longrightarrow}_{\mathsf{c}} F_2'$ and $F_1' \Rightarrow_{\mathsf{c}} \mathcal{R}^{|\star}\ {}_{\mathsf{c}}\Leftarrow F_2'$;*
2. *the converse, on the transitions from F_2.*

A variant of the technique in Definition 21, where the bisimulation game is played only on visible actions at the price of being defined on singleton configurations is presented in [21] and used in Section 7.5 to lift any eager normal form bisimulation to an OGS "bisimulation up-to".

▶ **Theorem 22.** *If $\mathcal{R}$ is bisimulation up-to reduction and composition then $\mathcal{R} \subseteq \approx_{\mathsf{c}}$.*

The theorem is proved by showing that the Iπ image of $\mathcal{R}$ is a bisimulation up-to context and up-to $({}_{\pi}\gtrsim, \approx_{\pi})$, and appealing to Lemma 15(2). See [21] for details.

▶ Remark 23. Results such as Corollary 17 and Theorem 22 might suggest that the equality between two configurations implies the equality of all their singleton components. That is, if $F \doteq_{\mathsf{c}} G$, with $[p \mapsto M]$ part of F and $[p \mapsto N]$ part of G, then also $\langle [p \mapsto M] \rangle \doteq_{\mathsf{c}} \langle [p \mapsto N] \rangle$. A counterexample is given by the configurations

$$F_1 \stackrel{\text{def}}{=} \langle [p_1 \mapsto M] \cdot [p_2 \mapsto \Omega] \rangle \qquad \text{where } M \stackrel{\text{def}}{=} (\lambda z.\, \Omega)(x \lambda y.\, \Omega)$$
$$F_2 \stackrel{\text{def}}{=} \langle [p_1 \mapsto \Omega] \cdot [p_2 \mapsto M] \rangle$$

Intuitively the reason why $F_1 \eqsim_{\mathsf{c}} F_2$ is that M can produce an output (along the variable x), but an observer will never obtain access to the name at which M is located (p_1 or p_2). That is, the term M can interrogate x, but it will never answer, neither at p_1 nor at p_2.

7.4 Relationship between Concurrent and Alternating OGS

In Section 6 we have proved that the trace-based and bisimulation-based semantics produced by A-OGS and by Iπ under the opLTS coincide. In Section 7.1 we have obtained the same result for C-OGS and Iπ under the ordinary LTS. In this section, we develop these results so to conclude that all such equivalences for λ-terms actually coincide. In other words, the equivalence induced on λ-terms by their representations in OGS and Iπ is the same, regardless of whether we adopt the alternating or concurrent flavour for OGS, the opLTS or the ordinary LTS in Iπ, a trace or a bisimulation semantics. For this, in one direction, we show that the trace semantics induced by C-OGS implies that induced by A-OGS. In the opposite direction, we lift a bisimulation over the alternating LTS on singleton configurations into a bisimulation up-to composition over the concurrent LTS.

▶ **Lemma 24.** *If F_1, F_2 are* A-OGS *singleton configurations and $F_1 \approx_{\mathsf{a}} F_2$, then also $F_1 \approx_{\mathsf{c}} F_2$.*

Details may be found in [21].

▶ **Corollary 25.** *For any λ-terms M, N, the following statements are the same: $\langle M\rangle \eqsim_{\mathsf{a}} \langle N\rangle$; $\langle M\rangle \approx_{\mathsf{a}} \langle N\rangle$; $\langle M\rangle \eqsim_{\mathsf{c}} \langle N\rangle$; $\langle M\rangle \approx_{\mathsf{c}} \langle N\rangle$; $\mathcal{V}[\![M]\!] \eqsim_{\mathsf{o}\pi} \mathcal{V}[\![N]\!]$; $\mathcal{V}[\![M]\!] \approx_{\mathsf{o}\pi} \mathcal{V}[\![N]\!]$; $\mathcal{V}[\![M]\!] \eqsim_{\pi} \mathcal{V}[\![N]\!]$; $\mathcal{V}[\![M]\!] \approx_{\pi} \mathcal{V}[\![N]\!]$.*

7.5 Eager Normal Form Bisimulations

We recall Lassen's *eager normal-form (enf) bisimilarity* [29].

▶ **Definition 26.** *An* enf-bisimulation *is a triple of relation on terms $\mathcal{R}_{\mathcal{M}}$, values $\mathcal{R}_{\mathcal{V}}$, and evaluation contexts $\mathcal{R}_{\mathcal{K}}$ that satisfies:*

- *$M_1\ \mathcal{R}_{\mathcal{M}}\ M_2$ if either:*
  - *both M_1, M_2 diverge;*
  - *$M_1 \Downarrow E_1[xV_1]$ and $M_2 \Downarrow E_2[xV_2]$ for some x, values V_1, V_2, and evaluation contexts E_1, E_2 with $V_1\ \mathcal{R}_{\mathcal{V}}\ V_2$ and $K_1\ \mathcal{R}_{\mathcal{K}}\ K_2$;*
  - *$M_1 \Downarrow V_1$ and $M_2 \Downarrow V_2$ for some values V_1, V_2 with $V_1\ \mathcal{R}_{\mathcal{V}}\ V_2$.*
- *$V_1\ \mathcal{R}_{\mathcal{V}}\ V_2$ if $V_1 x\ \mathcal{R}_{\mathcal{M}}\ V_2 x$ for some fresh x;*
- *$K_1\ \mathcal{R}_{\mathcal{V}}\ K_2$ if $K_1[x]\ \mathcal{R}_{\mathcal{M}}\ K_2[x]$ for some fresh x.*

The largest enf-bisimulation *is called* enf-bisimilarity*.*

From Corollary 25 and existing results in the π-calculus [11] we can immediately conclude that the semantics on λ-terms induced by OGS (Alternating or Concurrent) coincides with enf-bisimilarity (i.e., Lassen's trees).

In this section, we show a direct proof of the result, for C-OGS bisimilarity, as an example of application of the up-to composition technique for C-OGS.

Terms, Values, and Evaluations contexts can be directly lifted to singleton concurrent configurations, meaning that we can transform a relation on terms, values, and contexts $\mathcal{R}$ into a relation $\widehat{\mathcal{R}}$ on singleton concurrent configurations in the following way:

- If $(M_1, M_2) \in \mathcal{R}$ then $(\langle [p \mapsto M_1], \varepsilon, \phi\rangle, \langle [p \mapsto M_2], \varepsilon, \phi\rangle) \in \widehat{\mathcal{R}}$, with $\phi = \mathtt{fv}(M_1, M_2) \uplus \{p\}$
- If $(V_1, V_2) \in \mathcal{R}$ then $(\langle [x \mapsto V_1], \phi\rangle, \langle [x \mapsto V_2], \phi\rangle) \in \widehat{\mathcal{R}}$, with $\phi = \mathtt{fv}(V_1, V_2) \uplus \{x\}$;
- If $(E_1, E_2) \in \mathcal{R}$ then $(\langle [p \mapsto (E_1, q)], \phi\rangle, \langle [p \mapsto (E_2, q)], \phi\rangle) \in \widehat{\mathcal{R}}$, with $\phi = \mathtt{fv}(E_1, E_2) \uplus \{p, q\}$.

▶ **Theorem 27.** *Taking $(\mathcal{R}_{\mathcal{M}}, \mathcal{R}_{\mathcal{V}}, \mathcal{R}_{\mathcal{K}})$ an enf-bisimulation, then $(\widehat{\mathcal{R}_{\mathcal{M}}} \cup \widehat{\mathcal{R}_{\mathcal{V}}} \cup \widehat{\mathcal{R}_{\mathcal{K}}})$ is a singleton bisimulation up-to composition in* C-OGS *(as defined in [21]).*

Proof. Taking F_1, F_2 two singleton configurations s.t. $(F_1, F_2) \in \widehat{\mathcal{R}_{\mathcal{M}}}$, then we can write F_i as $\langle [p \mapsto M_i], \phi\rangle$ for $i = 1, 2$, such that $(M_1, M_2) \in \mathcal{R}$. suppose that $F_1 \Longrightarrow_{\mathtt{c}} \xrightarrow{\ell}_{\mathtt{c}} F_1'$, with ℓ a Player action. We only present the Player Question case, which is where "up-to composition" is useful. So writing ℓ as $\bar{x}(y, q)$, F_1' can be written as $\langle [y \mapsto V_1] \cdot [q \mapsto (E_1, p)]\rangle$, so that $M_1 \to_{\mathtt{v}}^* E_1[xV_1]$.

As $(M_1, M_2) \in \mathcal{R}$, there are E_2, V_2 s.t. $M_2 \to_{\mathtt{v}}^* E_2[xV_2]$, $(V_1, V_2) \in \mathcal{R}$ and $(E_1, E_2) \in \mathcal{R}$. Hence $F_2 \Longrightarrow_{\mathtt{c}} \xrightarrow{\ell}_{\mathtt{c}} F_2'$ with $F_2' = \langle [y \mapsto V_2] \cdot [q \mapsto (E_2, p)]\rangle$.

Finally, $(\langle [y \mapsto V_1]\rangle, \langle [y \mapsto V_2]\rangle) \in \widehat{\mathcal{R}}$ and $(\langle [q \mapsto (E_1, p)]\rangle, \langle [q \mapsto (E_2, p)]\rangle) \in \widehat{\mathcal{R}}$, so that $(F_1', F_2') \in \widehat{\mathcal{R}}^{\mid\star}$.

The case of passive singleton configurations is proved in a similar way. ◀

8 Compositionality of OGS via Iπ

We now present some compositionality results about C-OGS and A-OGS, that can be proved via the correspondence between OGS and Iπ.

Compositionality of OGS amounts to compute the set of traces generated by $\langle M\{V/x\}\rangle$ from the set of traces generated by $\langle M\rangle$ and $\langle [x \mapsto V]]\rangle$. This is the cornerstone of (denotational) game semantics, where the combination of $\langle M\rangle$ and $\langle [x \mapsto V]\rangle$ is represented via the so-called "parallel composition plus hiding". This notion of parallel composition of two processes P, Q plus hiding over a name x is directly expressible in Iπ as the process $\nu x(P \mid Q)$. Precisely, suppose F, G are two configurations that agree on their polarity functions, but on a name x; then we write

$$\nu x(F \mid G) \tag{$*$}$$

for the Iπ process $\nu x(\mathcal{V}[\![F]\!] \mid \mathcal{V}[\![G]\!])$, the *parallel composition plus hiding over* x. To define this operation directly at the level of OGS, we would have to generalize its LTS, allowing internal interactions over a name x used both in input and in output.

As the translation $\mathcal{V}$ from OGS configurations into Iπ validates the β_v rule, we can prove that the behaviour of $\langle M\{V/x\}\rangle$ (e.g., its set of traces) is the same as that of the parallel composition plus hiding over x of $\langle M\rangle$ and $\langle [x \mapsto V]\rangle$). We use the notation $(*)$ above to express the following two results.

▶ **Theorem 28.** *For $\bowtie \in \{\approx_{\mathtt{o}\pi}, \approx_{\pi}, \eqsim_{\mathtt{o}\pi}, \eqsim_{\pi}\}$, we have*
$\mathcal{V}[\![\langle [p \mapsto M\{V/x\}], \gamma, \phi \cup \phi'\rangle]\!] \bowtie \nu x(\langle [p \mapsto M], \varepsilon, \phi \uplus \{x\}\rangle \mid \langle \gamma \cdot [x \mapsto V], \phi' \uplus \{x\}\rangle)$

▶ **Corollary 29.** *For any trace s:*
1. $\langle M\{V/x\}, p, \gamma, \phi \cup \phi'\rangle \overset{s}{\Longrightarrow}_{\mathtt{a}}$ *iff* $\nu x(\langle M, p, \varepsilon, \phi \uplus \{x\}\rangle \mid \langle \gamma \cdot [x \mapsto V], \phi' \uplus \{x\}\rangle) \overset{s}{\Longrightarrow}_{\mathtt{o}\pi}$
2. $\langle [p \mapsto M\{V/x\}], \gamma, \phi \cup \phi'\rangle \overset{s}{\Longrightarrow}_{\mathtt{c}}$ *iff* $\nu x(\langle [p \mapsto M], \varepsilon, \phi \uplus \{x\}\rangle \mid \langle \gamma \cdot [x \mapsto V], \phi' \uplus \{x\}\rangle) \overset{s}{\Longrightarrow}_{\pi}$

Other important properties that we can import in OGS from Iπ are the congruence properties for the A-OGS and C-OGS semantics. We report the result for $\eqsim_{\mathtt{a}}$; the same result holds for $\approx_{\mathtt{a}}, \eqsim_{\mathtt{c}}, \approx_{\mathtt{c}}$.

▶ **Theorem 30.** *If $\langle M\rangle \eqsim_{\mathtt{a}} \langle N\rangle$ then for any Λ_{V} context C, $\langle C[M]\rangle \eqsim_{\mathtt{a}} \langle C[N]\rangle$.*

The result is obtained from the congruence properties of Iπ, Corollary 25, and the compositionality of the encoding $\mathcal{V}$.

9 Related and Future Work

Analogies between game semantics and π-calculus, as semantic frameworks in which *names* are central, have been pointed out from the very beginning of game semantics. In the pioneering work [19], the authors obtain a translation of PCF terms into the π-calculus from a game model of PCF by representing *strategies* (the denotation of PCF terms in the game model) as processes of the π-calculus. The encoding bears similarities with Milner's, though they are syntactically rather different ("it is clear that the two are conceptually quite unrelated", [19]). The connection has been developed in various papers, e.g., [8, 14, 17, 18, 46]. Milner's encodings into the π-calculus have sometimes been a source of inspiration in the definition of the game semantics models (e.g., transporting the work [19], in call-by-name, onto call-by-value [18]). In [46], a typed variant of π-calculus, influenced by differential linear logic [13], is introduced as a metalanguage to represent game models. In [9], games are defined using algebraic operations on sets of traces, and used to prove type soundness of a simply-typed call-by-value λ-calculus with effects. Although the calculus of traces employed is not a π-calculus (e.g., being defined from operators and relations over trace sets rather than from syntactic process constructs), there are similarities, which would be interesting to investigate.

Usually in the above papers the source language is a form of λ-calculus, that is interpreted into game semantics, and the π-calculus (or dialects of it) is used to represent the resulting strategies and games. Another goal has been to shed light into typing disciplines for π-calculus processes, by transplanting conditions on strategies such as well-bracketing and innocence into appropriate typings for the π-calculus (see, e.g., [6, 45]).

The results in this paper (e.g., operational correspondence and transfer of techniques) are not derivable from the above works, where analogies between game semantics and π-calculus are rather used to better understand one of the two models (i.e., explaining game semantics in terms of process interactions, or enhancing type systems for processes following structures in game semantics). Indeed, in the present paper we have carried out a *direct comparison* between the two models (precisely OGS and Iπ). For this we have started from the (arguably natural) representations of the λ-calculus into OGS and Iπ (the latter being Milner's encodings). Our goal was understanding the relation between the behaviours of the terms in the two models, and transferring techniques and results between them.

Technically, our work builds on [10, 11], where a detailed analysis of the behaviour of Milner's call-by-value encoding is carried out using proof techniques for π-calculus based on unique-solution of equations. Various results in [10, 11] are essential to our own (the observation that Milner's encodings should be interpreted in Iπ rather than the full π-calculus is also from [10, 11]).

Bisimulations over OGS terms, and tensor products of configurations, were introduced in [30], in order to provide a framework to study compositionality properties of OGS. In our case, the compositionality result of OGS is derived from the correspondence with the π-calculus. In [39], a correspondence between an i/o typed asynchronous π-calculus and a computational λ-calculus with channel communication is established, using a common categorical model (a compact closed Freyd category). It would be interesting to see if our concurrent operational game model could be equipped with this categorical semantics.

Normal form (or *open*) bisimulations [29, 43], as game semantics, manipulate open terms, and sometimes make use of environments or stacks of evaluation contexts (see, e.g., the recent work [7], where a fully abstract normal-form bisimulation for a λ-calculus with store is obtained).

There are also works that build game models directly for the π-calculus, i.e., [12, 26, 27, 38] A correspondence between a synchronous π-calculus with session types and concurrent game semantics [44] is given in [8], relating games (represented as arenas) to session types, and strategies (defined as coincident event structures) to processes. We have exploited the full abstraction results between OGS and Iπ to transport a few up-to techniques for bisimulation from Iπ onto OGS. However, in Iπ, there are various other such techniques, even a theory of bisimulation enhancements. We would like to see which other techniques could be useful in OGS, possibly transporting the theory of enhancements itself.

References

1 Samson Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1989.

2 Samson Abramsky. Game semantics for programming languages (abstract). In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science*, MFCS '97, pages 3–4, Berlin, Heidelberg, 1997. Springer-Verlag.

3 Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, LICS '98, page 334, USA, 1998. IEEE Computer Society.

4 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for pcf. *Inf. Comput.*, 163(2):409–470, December 2000. doi:10.1006/inco.2000.2930.

5 Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In *Algol-like languages*, pages 297–329. Springer, 1997.

6 Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *International Conference on Typed Lambda Calculi and Applications*, pages 29–45. Springer, 2001.

7 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity for state. In *International Conference on Foundations of Software Science and Computation Structures*, pages 98–114. Springer, 2019.

8 Simon Castellan and Nobuko Yoshida. Two sides of the same coin: Session types and game semantics: A synchronous side and an asynchronous side. *Proc. ACM Program. Lang.*, January 2019. doi:10.1145/3290340.

9 Tim Disney and Cormac Flanagan. Game semantics for type soundness. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, pages 104–114, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.20.

10 Adrien Durier. *Unique solution techniques for processes and functions*. PhD thesis, University of Lyon, France, 2020.

11 Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Eager functions as processes. In *33nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*. IEEE Computer Society, 2018.

12 Clovis Eberhart, Tom Hirschowitz, and Thomas Seiller. An intensionally fully-abstract sheaf model for pi. In *6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

13 Thomas Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Mathematical Structures in Computer Science*, 28(7):995–1060, 2018.

14 Marcelo Fiore and Kohei Honda. Recursive types in games: Axiomatics and process representation. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, LICS '98, page 345, USA, 1998. IEEE Computer Society.

15 Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. In *International Conference on Foundations of Software Science and Computation Structures*, pages 211–225. Springer, 2004.

16 Dan R. Ghica and Nikos Tzevelekos. A system-level game semantics. *Electron. Notes Theor. Comput. Sci.*, 286:191–211, September 2012. doi:10.1016/j.entcs.2012.08.013.

17 Kohei Honda. Processes and games. *Electron. Notes Theor. Comput. Sci.*, 71:40–69, 2002. doi:10.1016/S1571-0661(05)82528-9.

18 Kohei Honda and Nobuko Yoshida. Game-theoretic analysis of call-by-value computation. *Theor. Comput. Sci.*, 221(1–2):393–456, June 1999. doi:10.1016/S0304-3975(99)00039-0.

19 J. M. E. Hyland and C.-H. L. Ong. Pi-calculus, dialogue games and full abstraction PCF. In *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture*, FPCA '95, pages 96–107, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/224164.224189.

20 J.M.E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Inf. Comput.*, 163(2):285–408, December 2000. doi:10.1006/inco.2000.2917.

21 Guilhem Jaber and Davide Sangiorgi. Games, mobile processes, and functions (long version), 2021. URL: https://hal.archives-ouvertes.fr/hal-03407123.

22 Guilhem Jaber and Nikos Tzevelekos. Trace semantics for polymorphic references. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 585–594, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2933575.2934509.

23 Radha Jagadeesan, Corin Pitcher, and James Riely. Open bisimulation for aspects. In *Transactions on Aspect-Oriented Software Development V*, pages 72–132. Springer, 2009.

24 Alan Jeffrey and Julian Rathke. Java jr: Fully abstract trace semantics for a core java language. In *European symposium on programming*, pages 423–438. Springer, 2005.

25 James Laird. Full abstraction for functional languages with control. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, LICS '97, page 58, USA, 1997. IEEE Computer Society.

26 James Laird. A game semantics of the asynchronous π-calculus. In *International Conference on Concurrency Theory*, pages 51–65. Springer, 2005.

27 James Laird. Game semantics for higher-order concurrency. In *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'06, pages 417–428, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11944836.38.

28 James Laird. A fully abstract trace semantics for general references. In *Proceedings of the 34th International Conference on Automata, Languages and Programming*, ICALP'07, pages 667–679, Berlin, Heidelberg, 2007. Springer-Verlag.

29 Søren B. Lassen. Eager normal form bisimulation. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 345–354, 2005. doi:10.1109/LICS.2005.15.

30 Paul Blain Levy and Sam Staton. Transition systems over games. In *Proceedings of the 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, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2603088.2603150.

31 R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

32 R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

33 R. Milner. Functions as processes. *MSCS*, 2(2):119–141, 1992.

34 B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *MSCS*, 6(5):409–454, 1996.

35 G.D. Plotkin. Call by name, call by value and the λ-calculus. *TCS*, 1:125–159, 1975.

36 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.

37 Damien Pous and Davide Sangiorgi. Bisimulation and coinduction enhancements: A historical perspective. *Formal Asp. Comput.*, 31(6):733–749, 2019. doi:10.1007/s00165-019-00497-w.

38 Ken Sakayori and Takeshi Tsukada. A truly concurrent game model of the asynchronous π-calculus. In *International Conference on Foundations of Software Science and Computation Structures*, pages 389–406. Springer, 2017.

39 Ken Sakayori and Takeshi Tsukada. A categorical model of an $\mathbf{i}/\mathbf{o}$-typed π-calculus. In *European Symposium on Programming*, pages 640–667. Springer, 2019.

40 D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *TCS*, 155:39–83, 1996.

41 D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *TCS*, 167(2):235–274, 1996.

42 D. Sangiorgi. Lazy functions and mobile processes. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

43 Kristian Støvring and Soren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 161–172, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1190216.1190244.

44 Glynn Winskel, Silvain Rideau, Pierre Clairambault, and Simon Castellan. Games and strategies as event structures. *Logical Methods in Computer Science*, 13, 2017.

45 Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong normalisation in the π-Calculus. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 311–322. IEEE Computer Society, 2001.

46 Nobuko Yoshida, Simon Castellan, and Léo Stefanesco. Game semantics: Easy as pi. *arXiv preprint*, 2020. arXiv:2011.05248.

Parallelism in Soft Linear Logic

Paulin Jacobé de Naurois ✉
CNRS, Université Paris 13, Sorbonne Paris Cité, LIPN, UMR 7030, F-93430 Villetaneuse, France

Abstract

We extend the Soft Linear Logic of Lafont with a new kind of modality, called *parallel*. Contractions on parallel modalities are only allowed in the cut and the left $\multimap$ rules, in a controlled, uniformly distributive way. We show that SLL, extended with this parallel modality, is sound and complete for PSPACE. We propose a corresponding typing discipline for the λ-calculus, extending the STA typing system of Gaboardi and Ronchi, and establish its PSPACE soundness and completeness. The use of the parallel modality in the cut-rule drives a polynomial-time, parallel call-by-value evaluation strategy of the terms.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Complexity theory and logic

Keywords and phrases Implicit Complexity, Typing, Linear Logic, Functional Programming

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.26

Funding *Paulin Jacobé de Naurois*: ANR project ELICA – ANR-14-CE25-0005.

Introduction

Implicit Complexity aims at providing purely syntactical, machine independent criteria on programs, in order to ensure they respect some complexity bounds upon execution. In the context of functional programming, the use of tailored proof systems, and subsequent type systems for λ-calculus, has been very successful: using subsystems of Linear Logic [8], several proof systems have been proposed, where cut-elimination has a bounded complexity. Consequently, under the Curry-Howard isomorphism, type systems for λ-calculus based on these logics have been proposed, where β-normalization of the typed terms follows the same complexity bounds. Such results include, among many others, Bounded Linear Logic [10, 14] and Light Linear Logic [9, 2] for polynomial time computation, and Stratified Bounded Affine Logic [17, 15] for logarithmic space computations. Our interest in this paper lies in the Soft Linear Logic of Lafont [13], which proposes a simple and elegant approach for ensuring polynomial time bounds by controlling contractions on exponential formulas, and in the subsequent type systems for polynomial time λ-calculus [1, 7, 5].

At this point, it is relevant to note that the complexity classes captured thus far are all sequential, deterministic *in essence*. While Soft Linear Logic type systems have been extended to express the classes NP and PSPACE [6], it is important to note that the construction relies on Soft Type Assignment (STA), a deterministic, sequential polynomial time type system, by extending the λ-calculus with an additional construct (`if then else`), for which an ad hoc, alternating polynomial time evaluation strategy is imposed - the core of the language retaining its sequential polynomial time evaluation. While being indeed extensionally complete for PSPACE, this approach lacks intensionality: many natural algorithms, that are easily computable in parallel, are hardly expressible in this setting. Let us take as simple example the numerical evaluation of a balanced, arithmetic expression on bounded integer values. In order to compute it in alternating polynomial time with the (`if then else`) defined in [6], one would need to express the value of all bits of the result as boolean expressions on the bits of the input numbers, and use the alternating evaluation of the (`if then else`) construct to speed up the parallel computation time - not quite a practical method. Furthermore, this approach is no longer doable in real world functional programming languages, where

integers are given as a base type, and arithmetical operations as unitary functions of the language. Our approach, on the other hand, extends very naturally to such programs: indeed, our complexity bounds still hold in this context, and the encodings used in Lemma 27 and Theorem 28 can seamlessly be used to encode uniform families of algebraic formulae, or algebraic circuits, of polynomial depth, provided the base type for numbers (be they integers or floating numbers, or even real or complex numbers) and for the algebraic basic operations are given in the typing context.

A reason why these approaches are all essentially sequential, deterministic is that they use the typing discipline to control the *amount* of resources the calculus uses (e.g. by controlling contractions on exponentials), not the *way* these resources are distributed along the computation. In order to truly denote parallel computation in a functional programming language, our proposal here is to use a parallel, call-by-value evaluation strategy for the λ-calculus: in an application, both terms can be normalized in parallel, before the substitution of the redex takes place. If both terms share the same normalization time bound, the parallel evaluation strategy is efficient. Note that in first order functional programming, this is already the approach used by Leivant and Marion [16] with their safe recursion with substitutions: using sequential resource bounds from Ptime Safe Recursion [3], and a parallel call-by-value evaluation strategy, the authors characterize the class FPAR (Parallel polynomial time), which coincides with PSPACE. This approach has also been later on extended to sub-polynomial complexity classes [12, 4, 11]. For higher order functional programming, we rely on the Curry-Howard isomorphism: ensuring an homogeneous computation time on the parallel evaluation of both arguments of an application amounts to ensuring that both premises of a cut-rule share a homogeneous bound on the resource usage in the corresponding type system.

In order to achieve this, we can no longer rely on the usual linear cut-rule. We propose therefore a modification of the linear cut-rule, that internalizes a controlled number of contractions on some formulas, that are uniformly distributed among the premises. These formulas are decorated with a dedicated modality, called parallel modality. This approach is applied here on the Soft Type Assignment (STA) of Gaboardi and Ronchi [7], in order to propose a sound and complete type system for PSPACE, with a truly parallel evaluation strategy.

Of course, breaking linearity in the cut-rule comes with a price: while proof nets for this logic are still definable, the additional bureaucracy needed to deal with the side condition of the cut-rule makes them much less meaningful than those for simpler logical systems such as MLL or SLL.

The paper is organized as follows. Section 1 recalls the Soft Linear Logic rules, introduces the parallel modality $/\!\!/$, and the modified, parallel (cut) rules, yielding the system PSLL. Cut-elimination for PSLL is also shown. Section 2 provides a parallel, polynomial time normalization bound. Section 3 extends STA with the rules of PSLL, yielding PSTA. A parallel polynomial time call-by-value strategy for PSTA is described. FPAR completeness of PSTA is proven in Section 4.

1 Parallel Soft Linear Logic

1.1 Soft Linear Logic

Let us recall the SLL rules of Lafont [13], in its intuitionistic fashion. In the following, $!\Gamma$ stands for a multiset of formulae of the form $!F$, and $(A)^n$ stands for n copies of a formula A.

$$\frac{}{U \vdash U}\,(Id) \qquad \frac{\Gamma \vdash U \qquad \Delta, U \vdash V}{\Gamma, \Delta \vdash V}\,(cut) \qquad \frac{\Gamma, U \vdash V}{\Gamma \vdash U \multimap V}\,(\multimap R)$$

$$\frac{\Gamma \vdash U \qquad V, \Delta \vdash Z}{\Gamma, U \multimap V, \Delta \vdash Z}\,(\multimap L) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \& B}\,(\&R) \qquad \frac{\Gamma, A \vdash V}{\Gamma, A \& B \vdash V}\,(\&L_1)$$

$$\frac{\Gamma, B \vdash V}{\Gamma, A \& B \vdash V}\,(\&L_2) \qquad \frac{\Gamma \vdash U}{\Gamma \vdash \forall \alpha U}\,(\forall R) \qquad \frac{\Gamma, U[V/\alpha] \vdash Z}{\Gamma, \forall \alpha U \vdash Z}\,(\forall L)$$

$$\frac{\Gamma \vdash U}{!\Gamma \vdash !U}\,(sp) \qquad \frac{\Gamma, (U)^n \vdash V \qquad n \geq 0}{\Gamma, !U \vdash V}\,(m), \text{ of rank } n$$

where, in the $(\forall R)$-rule, there is no free occurrence of α in Γ. SLL proofs (of a given degree) normalize in polynomial time. Let the rank of a proof be the maximal rank of its (m) rules, and its degree the maximal nesting of its (sp) rules:

▶ **Theorem 1** ([13]). *A SLL proof of rank n and degree d normalizes in n^d steps.*

SLL is also complete for the class FP: inputs of size n are encoded with proofs of rank n, degree 1, and programs running in time $O(n^d)$ by proofs of degree d. Applying a program on an input amounts to performing a (cut) of the two proof derivations.

1.2 Parallel Modalities

PSLL is built upon SLL. An additional modality, called the parallel modality $/\!\!/$, is introduced, with corresponding elimination rules. Finally, the (sp), and the (cut) and $(\multimap L)$-rules are modified to accommodate this new modality, implementing the controlled contractions and homogeneous distribution of $/\!\!/$ formulas on the premises of the cut, as follows.

Polarities

Let us define as usual inductively the polarity of a sub-formula in an intuitionistic sequent $\Gamma \vdash V$. Polarities are either positive or negative, one being the opposite of the other.

1. in $\Gamma \vdash V$, every occurrence of a formula F in Γ is negative, and V is positive.
2. If F is $\forall \alpha A$, $!A$ or $/\!\!/ A$, the polarity of A is the polarity of F.
3. If F is $A \& B$, the polarity of A and the polarity of B are the polarity of F.
4. If F is $A \multimap B$, the polarity of A is the opposite of the polarity of F, and the polarity of B is the polarity of F.

In the sequel we only admit $/\!\!/ A$ sub-formulas with negative polarities in a sequent. An immediate consequence is that no $/\!\!/$ modality can appear in a cut formula, since a cut-formula has both a positive and a negative occurrence in a proof tree.

Rules for Parallel Modalities

$(/\!\!/ W)$ (weakening) and $(/\!\!/ D)$ (dereliction) rules eliminate the $/\!\!/$ modality, $(/\!\!/ sp)$ (soft promotion for the ! modality) and $(/\!\!/ ax)$ replace the linear (sp) and (Id) rules. Contraction for the $/\!\!/$ modality is not dealt with a dedicated rule, but is instead internalized in the side condition of the modified (cut) rule, as detailed in the next section.

$$\frac{\Gamma \vdash B}{\Gamma, /\!\!/ A \vdash B}\,(/\!\!/ W) \qquad \frac{\Gamma, A \vdash B}{\Gamma, /\!\!/ A \vdash B}\,(/\!\!/ D) \qquad \frac{/\!\!/ \Delta, \Gamma, \vdash U}{/\!\!/ \Delta, !\Gamma \vdash !U}\,(/\!\!/ sp) \qquad \frac{}{/\!\!/ \Gamma, A \vdash A}\,(/\!\!/ ax)$$

where $(/\!\!/ ax)$, is derivable from (Id) and $(/\!\!/ W)$, and used for convenience only.

1.2.1 (⅋Cut) and (⅋ ⊸) Rules

Contraction for parallel formulas is internalized into the PSLL ($\parr cut$)-rule and ($\parr \multimap L$)-rule, in a controlled fashion. As for the usual (cut) and ($\multimap L$)-rules in linear logic, linear and exponential formulas are linearly distributed among the two premises. Denote by $\subsetneq$ the strict inclusion relation on multisets. The (binary) ($\parr cut$) and ($\parr \multimap L$) rules are the following.

$$\frac{\parr\Delta_1, \Gamma_1 \vdash A_1 \qquad \parr\Delta_2, \Gamma_2, A_1 \vdash A_2}{\parr\Delta, \Gamma_1, \Gamma_2 \vdash A_2}\,(\parr cut) \qquad \frac{\parr\Delta_1, \Gamma_1 \vdash A_1 \qquad \parr\Delta_2, \Gamma_2, A_2 \vdash A_3}{\parr\Delta, \Gamma_1, A_1 \multimap A_2, \Gamma_2 \vdash A_3}\,(\parr \multimap L)$$

These two rules hold under the side condition $\mathcal{S}_P : (\parr\Delta_1 \subsetneq \parr\Delta,\ \parr\Delta_2 \subsetneq \parr\Delta)$. The ($\parr cut$)- rule has the principal cut-formula A_1 and the cut-pair of premises the pair $(\parr\Delta_1, \Gamma_1 \vdash A_1) \to (\parr\Delta_2, \Gamma_2, A_1 \vdash A_2)$. The ($\parr \multimap L$) rule has the principal $\multimap$-formula $A_1 \multimap A_2$ and the $\multimap$-pair of premises the pair $(\parr\Delta_1, \Gamma_1 \vdash A_1) \to (\parr\Delta_2, \Gamma_2, A_2 \vdash A_3)$.

In a proof tree consisting only in $(n-1)$ binary linear (cut)-rules, these (cut)-rules can be freely permuted, and a generalized, n-ary linear (cut)-rule can be derived. The non-linear distribution of parallel modalities in PSLL breaks this isomorphism: permuting two binary ($\parr cut$)-rules may come in conflict with the side condition $\parr\Delta_i \subsetneq \parr\Delta$. A similar remark can be made for $\multimap L$ rules as well. Since we want a *uniform* bound on the parallel normalization of the premises, we define a n-ary parallel (cut)-rule, exemplified in Example 5, as a parallel extension of the linear one, where the side condition for ⅋ modalities is adapted accordingly.

▶ **Definition 2** (n-ary ($cut/\ \multimap L$) rule). *We define the following n-ary ($cut/\ \multimap L$)-rule, together with its cut-pairs and $\multimap$-pairs, and principal formulae. To each cut-pair (respectively $\multimap$-pair) corresponds one principal cut-formula (resp. $\multimap$-formula).*

The following rule $\dfrac{\Gamma_1 \vdash A_1 \qquad \Gamma_2 \vdash A_2 \cdots \qquad \Gamma_d \vdash A_d}{\Gamma, \Lambda \vdash A_d}\,R$ *: ($cut/\ \multimap L$) is either a binary ($\multimap L$) or a binary (cut)-rule, or a n-ary rule obtained by several of the following proof tree ($cut/\ \multimap L$)-merge rewriting steps:*

$$\frac{T_1 \cdots \dfrac{T_2 \cdots T_n}{T_t}\,R_1 \cdots T_m}{\Gamma, \Lambda \vdash A_d}\,R_2 \qquad \to \qquad \frac{T_1 \cdots T_2 \cdots T_n \cdots T_m}{\Gamma, \Lambda \vdash A_d}\,R$$

provided the $\multimap$ principal formulae of R_1 are not sub-formulae of any principal formulae of R_2 corresponding T_t.

The multiset of $\multimap$ (respectively (cut)) principal formulae of R is then the union of those of R_1 and R_2.

The cut - and $\multimap$-pairs of R are obtained from the union of those of R_1 and R_2 with the following update procedure: whenever T_t belongs to a $\multimap$ or cut-pair of premises $T_t \to T_w$ (respectively $T_w \to T_t$) of R_2, with corresponding principal formula F belonging to one of the premises T_v of R_1, the pair $T_t \to T_w$ (resp. $T_w \to T_t$) is replaced by $T_v \to T_w$ (resp. $T_w \to T_v$), with the same corresponding principal formula.

We derive from this linear n-ary ($cut/\ \multimap L$) rule its parallel version ($\parr, \multimap cut$) as follows.

▶ **Definition 3** (n-ary ($\parr, \multimap cut$) rule). *A n-ary ($\parr, \multimap cut$) rule is*

$$\frac{\parr\Delta_1, \Gamma_1 \vdash A_1 \qquad \parr\Delta_2, \Gamma_2 \vdash A_2 \cdots \qquad \parr\Delta_d, \Gamma_d \vdash A_d}{\parr\Delta, \Gamma, \Lambda \vdash A_d}\,(\parr, \multimap cut)$$

where the side condition $\mathcal{S}_P : \forall i = 1, \cdots, d, \parr\Delta_i \subsetneq \parr\Delta$ holds, and the linear rule instance

$$\frac{\Gamma_1 \vdash A_1 \qquad \Gamma_2 \vdash A_2 \cdots \qquad \Gamma_d \vdash A_d}{\Lambda, \Gamma \vdash A_d}\,(cut/\ \multimap L)$$

holds as per Definition 2, with corresponding pairs of premises and principal formulae.

The following Lemma follows from the intuitionistic nature of the PSLL sequents, and will play a role in our elimination strategy.

▶ **Lemma 4.** *The cut-pairing relation on the premises of a* $(/\!\!/, cut/ \multimap L)$ *rule* R *defines a forest structure* $F(R)$*, called the* pairing forest *of the* $(/\!\!/, cut/ \multimap L)$*, on the premises of* R*; the edges of the pairing forest are the cut-pairs of the rule.*

▶ **Example 5.** A tree of linear $(\multimap L)$ and (cut) rules is

$$\dfrac{\Gamma_1 \vdash A_1 \qquad \dfrac{\dfrac{\Gamma_2 \vdash A_2 \qquad \Gamma_3, A_2 \vdash U}{\Gamma_2, \Gamma_3 \vdash U}\,(cut) \qquad \dfrac{\Gamma_4 \vdash A_3 \qquad \Gamma_5, A_1, A_3, V \vdash W}{\Gamma_4, \Gamma_5, A_1, V \vdash W}\,(cut)}{\Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V, A_1 \vdash W}\,(\multimap L)}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W}\,(cut)$$

The corresponding 5-ary linear $(cut/ \multimap L)$ rule is

$$\dfrac{\Gamma_1 \vdash A_1 \qquad \Gamma_2 \vdash A_2 \qquad \Gamma_3, A_2 \vdash U \qquad \Gamma_4 \vdash A_3 \qquad \Gamma_5, A_1, A_3, V \vdash W}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W}\,(cut, \multimap L)$$

A corresponding 5-ary $(/\!\!/, \multimap cut)$ rule, with $/\!\!/$-formulae satisfying the side condition, is

$$\dfrac{/\!\!/F, \Gamma_1 \vdash A_1 \qquad /\!\!/G, \Gamma_2 \vdash A_2 \qquad \Gamma_3, A_2 \vdash U \qquad /\!\!/G, \Gamma_4 \vdash A_3 \qquad /\!\!/F, \Gamma_5, A_1, A_3, V \vdash W}{/\!\!/F, /\!\!/G, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W}$$

The cut-pairs are

- $(/\!\!/F, \Gamma_1 \vdash A_1) \to (/\!\!/F, \Gamma_5, A_1, A_3, V \vdash W)$ with principal formula A_1,
- $(/\!\!/G, \Gamma_2 \vdash A_2) \to (\Gamma_3, A_2 \vdash U)$ with principal formula A_2, and
- $(/\!\!/G, \Gamma_4 \vdash A_3) \to (/\!\!/F, \Gamma_5, A_1, A_3, V \vdash W)$ with principal formula A_3,

which defines the pairing forest, with two roots $(/\!\!/F, \Gamma_5, A_1, A_3, V \vdash W)$ and $(\Gamma_3, A_2 \vdash U)$. The $\multimap$-pair is $(\Gamma_3, A_2 \vdash U) \to (/\!\!/F, \Gamma_5, A_1, A_3, V \vdash W)$ with principal formula $U \multimap V$.

We now define PSLL by the rules $(/\!\!/ax)$, $(\multimap R)$, $(\forall R)$, $(\forall L)$, $(\& R)$, $(\& L_i)$, (m) $(/\!\!/sp)$, $(/\!\!/W)$, $(/\!\!/D)$ and $(/\!\!/, \multimap cut)$.

A PSLL proof Π is said to be in *normal form* if it contains no cut: more precisely, no $(/\!\!/, \multimap cut)$-rule in Π admits any cut-pair of premises. Cut-elimination in this context amounts to rewrite the proof into a new equivalent proof in normal form. The cut-elimination procedure stems on the usual one for SLL, with some refinements.

1.3 Parallel Cut Elimination

▶ **Lemma 6.** *Sequent calculus rules preserve the polarities of subformulae.*

The proof is straightforward. This allows us to state the following rule commutation result.

▶ **Lemma 7.**

1. *A* $(/\!\!/, \multimap cut)$ *rule* (R_1)*, with premise* $\Gamma \vdash V$ *commutes with any non* $(/\!\!/, \multimap cut)$*, non* $(/\!\!/W)$*, non* $(/\!\!/D)$*, non* $(/\!\!/sp)$ *rule* (R_2) *with conclusion* $\Gamma \vdash V$*, provided the principal formula of* (R_2) *is not a sub-formula of any principal formula of* (R_1) *with respect to the premise* $\Gamma \vdash V$*.*
2. *A* $(/\!\!/W)$ *or a* $(/\!\!/D)$ *rule* (R_1)*, with premise* $\Gamma \vdash V$ *commutes with any non* $(/\!\!/ax)$ *rule* (R_2) *with conclusion* $\Gamma \vdash V$*, provided the principal formula of* (R_2) *is not a subformula of the principal formula of* (R_1)*.*

3. *A non* $(\parr, \multimap\ cut)$*, non* $(\parr sp)$ *rule* (R_1) *with premise* $\Gamma \vdash V$ *commutes with any non* $(\parr, \multimap\ cut)$*, non* $(\parr sp)$ *rule* (R_2) *with conclusion* $\Gamma \vdash V$*, provided the principal formula of* (R_2) *is not a sub-formula of the principal formula of* (R_1)*.*

▶ **Proposition 8.** *PSLL enjoys cut elimination.*

Proof. Let Π be a PSLL proof and R be a $(\parr, \multimap\ cut)$ rule in Π with cut-pair $(S = \Gamma \vdash A, T = \Lambda, A \vdash V)$. Since the cut formula A may not contain any $\parr$ modality, the commutation rules of Lemma 7 allow us to rewrite Π into an equivalent proof Π', where S is conclusion of a right rule with principal formula A, and T conclusion of a left rule with principal formula A. The cut-elimination cases are then the following, where, for all cases but $(\parr Sp), (m)$, the other premises of the rule are left unchanged, and omitted. Side conditions as well are omitted, but it is straightforward to see that they are preserved. The modification induced by each of the elimination cases below on the pairing forest is also detailed.

Rules $(\multimap L)$,$(\multimap R)$

$$\dfrac{\dfrac{\parr\Delta_1, \Gamma, B \vdash C}{\parr\Delta_1, \Gamma \vdash B \multimap C}\,(\multimap R) \quad \dfrac{\parr\Delta_3, \Phi \vdash B \qquad \parr\Delta_4, \Lambda, C \vdash V}{\parr\Delta_2, \Phi, \Lambda, B \multimap C \vdash V}\,(\parr \multimap L)}{\parr\Delta, \Gamma, \Phi, \Lambda \vdash V}\,(\parr, \multimap\ cut)$$

reduces to $\dfrac{\parr\Delta_1, \Gamma, B \vdash C \qquad \parr\Delta_3, \Phi \vdash B \qquad \parr\Delta_4, \Lambda, C \vdash V}{\parr\Delta, \Gamma, \Phi, \Lambda \vdash V}\,(\parr, \multimap\ cut)$.

In the pairing forest, the premise $\parr\Delta_1, \Gamma \vdash B \multimap C$ is replaced by $\parr\Delta_1, \Gamma, B \vdash C$, the premise $\parr\Delta_2, \Phi, \Lambda, B \multimap C \vdash V$ by $\parr\Delta_4, \Lambda, C \vdash V$, and a cut-pair $(\parr\Delta_3, \Phi \vdash B) \to (\parr\Delta_1, \Gamma, B \vdash C)$ is added.

Rule $(\parr ax)$

$$\dfrac{\dfrac{}{\parr\Delta_1, B \vdash B}\,(\parr ax) \qquad \parr\Delta_2, \Gamma, B \vdash V}{\parr\Delta, \Gamma, B \vdash V}\,(\parr, \multimap\ cut)$$

when no other premise exists, reduces to $\dfrac{\parr\Delta_2, \Gamma, B \vdash V}{\parr\Delta, \Gamma, B \vdash V}\,(\parr W^*)$, , Where $(\parr W)^*$ stands for several applications of the $(\parr W)$ rule.

Similarly,

$$\dfrac{\Pi_1 \cdots \quad \dfrac{}{\parr\Delta_1, B \vdash B}\,(\parr ax) \quad \Pi_t \cdots \quad \parr\Delta_2, \Gamma, B \vdash V \quad \cdots \Pi_n}{\parr\Delta, \Gamma, B \vdash V}\,(\parr, \multimap\ cut)$$

reduces to $\dfrac{\Pi_1 \cdots \Pi_t \cdots \qquad \parr\Delta_2, \Gamma, B \vdash V \quad \cdots \Pi_n}{\parr\Delta, \Gamma, B \vdash V}\,(\parr, \multimap\ cut)$.

In the pairing forest, the premise $\parr\Delta_1, B \vdash B$ is removed, and the paths in the forest are shortened accordingly, if necessary.

Rules $(\parr sp)$, (m)

$$\dfrac{S_1, \cdots, S_k \quad \dfrac{\parr\Delta_1, \Gamma \vdash B}{\parr\Delta_1, !\Gamma \vdash !B}\,(\parr sp) \quad \dfrac{\parr\Delta_2, \Lambda, B^n \vdash V}{\parr\Delta_2, \Lambda, !B \vdash V}\,(m)}{\parr\Delta, !\Gamma, \Lambda \vdash V}\,(\parr, \multimap\ cut)$$

reduces to $\dfrac{\dfrac{S_1^n, \cdots, S_k^n \quad \parr\Delta_1, \Gamma \vdash B \cdots \qquad \parr\Delta_1, \Gamma \vdash B \qquad \parr\Delta_2, \Lambda, B^n \vdash V}{\parr\Delta, \Gamma^n, \Lambda, \vdash V}\,(m)^*}{\parr\Delta, !\Gamma, \Lambda \vdash V}\,(\parr, \multimap\ cut)$

Where $S_1, \cdots, S_k$ are the premises of the $(/\!\!/, \multimap\ cut)$ belonging to the pairing tree rooted in $/\!\!/\Delta_1, !\Gamma \vdash !B$, and $S_1^n, \cdots, S_k^n$ are n copies of these premises. Then, in the pairing forest, the tree rooted in$/\!\!/\Delta_1, !\Gamma \vdash !B$ is copied n times, and the pair $(/\!\!/\Delta_1, !\Gamma \vdash !B) \to (/\!\!/\Delta_2, \Lambda, !B \vdash V)$ is replaced by n pairs $(/\!\!/\Delta_1, \Gamma \vdash B) \to (/\!\!/\Delta_2, \Lambda, B^n \vdash V)$, one connected to each of the copies above.

Rules $(\forall L)$, $(\forall R)$

$$\dfrac{\dfrac{/\!\!/\Delta_1, \Gamma \vdash U}{/\!\!/\Delta_1, \Gamma \vdash \forall \alpha U}\,(\forall R) \qquad \dfrac{/\!\!/\Delta_2, \Lambda, U[V/\alpha] \vdash V}{/\!\!/\Delta_2, \Lambda, \forall \alpha U \vdash V}\,(\forall L)}{/\!\!/\Delta, \Gamma, \Lambda \vdash V}\,(/\!\!/, \multimap\ cut)$$

reduces to $\dfrac{/\!\!/\Delta_1, \Gamma \vdash U[V/\alpha] \qquad /\!\!/\Delta_2, \Lambda, U[V/\alpha] \vdash V}{/\!\!/\Delta, \Gamma, \Lambda \vdash V}\,(/\!\!/, \multimap\ cut)$.

Rules $(\&L)$, $(\&R)$

$$\dfrac{\dfrac{/\!\!/\Delta_1, \Gamma \vdash A \qquad /\!\!/\Delta_1 \Gamma \vdash B}{/\!\!/\Delta_1, \Gamma \vdash A \& B}\,(\&R) \qquad \dfrac{/\!\!/\Delta_2, \Lambda, A \vdash V}{/\!\!/\Delta_2, \Lambda, A \& B \vdash V}\,(\&L_1)}{/\!\!/\Delta, \Gamma, \Lambda \vdash V}\,(/\!\!/, \multimap\ cut)$$

reduces to $\dfrac{/\!\!/\Delta_1, \Gamma \vdash A \qquad /\!\!/\Delta_2, \Lambda, A \vdash V}{/\!\!/\Delta, \Gamma, \Lambda \vdash V}\,(/\!\!/, \multimap\ cut)$, and, of course, the cut elimination rule for $(\&L_2)$, $(\&R)$ follows a similar pattern.

In each of the cases above, for each path in the pairing forest modified by the elimination case, the sum of the sizes of the sequents labelling the vertices along that path decreases strictly. As a consequence, the procedure terminates in a finite number of steps. ◀

2 Complexity Bounds

Let us now show that the contraction discipline ensures that PSLL admits cut-elimination in parallel polynomial time. The bounds are actually more straightforward than for SLL.

▶ **Definition 9.** *Let Π be a PSLL proof, with conclusion sequent $S = \Gamma \vdash V$. We define*

- *The size $|S|$ of S is the number of connectives in S.*
- *The size $|\Pi|$ of Π is the number of nodes in the proof-tree.*
- *The depth of a node R in Π is the length of the path from S to R in Π; the depth $d(\Pi)$ of Π is the maximal depth of its nodes.*
- *The rank $r(\Pi)$ of Π is the maximal rank of its (m) rules.*
- *The degree $d(f)$ of a formula f is the maximal nesting of its ! modalities. The degree $d(S)$ of a sequent S is the maximal degree of its formulas. The degree $d(\Pi)$ of a proof is the maximal degree of its sequents.*

PSLL proofs have bounded depth, and bounded number of (cut)-rules.

▶ **Lemma 10.** *Let Π be a PSLL cut-free proof, of rank n, with conclusion sequent S of degree d. The depth of Π is then bounded by $O(|S|.n^d)$.*

▶ **Lemma 11.** *Let Π be a PSLL proof, of rank n, with conclusion sequent S of degree d. Then, on any path from S to an axiom in Π, there are at most $O(|S|.n^d)$ $(/\!\!/, \multimap\ cut)$-rules with cut-pairs of premises.*

Combining these two lemmas, we obtain a bound on the depth of PSLL proofs.

▶ **Lemma 12.** *Let* Π *be a PSLL proof, of rank* n *and degree* d*, with conclusion sequent* S*.Let* M *be the maximal size of its cut-formulas. Then, the depth of* Π *is* $O(M.|S|.n^{2d})$*.*

▶ **Lemma 13.** *Let* R *be a* $(\parr, \multimap\ cut)$ *rule with cut-pairs* $(S_1 = \Gamma_1 \vdash A_1) \to S, \cdots, (S_t = \Gamma_t \vdash A_t) \to S$ *and cut-formulae* $A_1, \cdots, A_t$*. Assume moreover that*

- *each of the proof trees with conclusion* S_i*, for* $i = 1, \cdots, t$*, ends with the PSLL rule with right principal formula* A_i*, and*
- *the proof tree with conclusion* S *ends with the* t *PSLL rules with left principal formula* A_i*, for* $i = 1, \cdots, t$*.*

Then, the cut-elimination steps of Proposition 8 for the cut-pairs $(S_1, S), \cdots, (S_t, S)$ *can be performed in parallel.*

Proof. The cut-elimination steps of Proposition 8 act on distinct left sub-formulae of S, and distinct premises (other than S) of the $(\parr, \multimap\ cut)$ rule R. ◀

▶ **Definition 14** (Parallel elimination of an innermost cut)**.** *Le* Π *be a PSLL proof. A* $(\parr, \multimap\ cut)$ *rule* R *with cut-pairs is innermost in* Π *if there is no other* $(\parr, \multimap\ cut)$ *rule with cut-pairs along any path from* R *to the axioms of* Π*.*

Let R *be an innermost* $(\parr, \multimap\ cut)$ *rule in* Π*, and* $F(R)$ *be the pairing forest. The parallel elimination of* R *is then the following procedure:*

1. *For any premise* $S = \Lambda \vdash B$ *of* R *root in* $F(R)$*, with cut-pairs* $(S_1 = \Gamma_1 \vdash A_1, S), \cdots, (S_t = \Gamma_t \vdash A_t, S)$*, perform the rule permutations of Lemma 7 such that* S *is conclusion of a proof tree with deep most rules the left rules with principal formulae* $A_1, \cdots, A_t$*, and*
2. *perform the rule permutations of Lemma 7 such that, for* $i = 1, \cdots, t$*,* S_i *is conclusion of a proof tree ending with a right rule with principal formula* A_i*.*
3. *perform in parallel the cut-elimination steps of Lemma 13 for all cut-pairs* (S_i, S) *for all roots* S *in* $F(R)$*.*
4. *if* R *has at least one cut-pair left, go to step 1.*

▶ **Definition 15** (Innermost parallel cut-elimination)**.** *Let* Π *be a PSLL proof. The* Innermost parallel cut-elimination *procedure consists in applying in parallel, for all its innermost cuts, their parallel elimination, until no* $(\parr, \multimap\ cut)$ *rule with cut-pairs remains.*

The innermost parallel cut-elimination procedure ensures that the blow-up of the $(\parr, \multimap\ cut)$ rules remains under control:

▶ **Lemma 16.** *Let* Π *be a PSLL proof with conclusion* S*, degree* d*, and rank* n*. Let* M *be the maximal size of its cut-formulae and* w *the maximal indegree of its pairing forests. Then, the maximal indegree of the pairing forests of any proof* Π' *derived from* Π *by an innermost parallel partial evaluation is bounded by* $O(w.n^d + M)$*.*

▶ **Lemma 17.** *Let* Π *be a PSLL proof with conclusion* S*, degree* d*, and rank* n*. Let* M *be the maximal size of its cut-formulae, and* h *the maximal height of its pairing forests. The parallel elimination of an innermost cut takes at most* $O(M.h)$ *parallel steps.*

Proof. For each of the elimination steps of Proposition 8, for each path in the pairing forest containing the cut-pair eliminated by this step, the sum of the sizes of the cut-formulae along the path strictly decreases, hence the result. ◀

We now have a parallel, polynomial time cut-elimination procedure:

▶ **Theorem 18.** *Let* Π *be a PSLL proof, of rank* n *and degree* d*, with conclusion sequent* S*. Let* M *be the maximal size of its cut-formulae, and* h *the maximal height of its pairing forests. Then, an innermost parallel cut-elimination strategy takes* $O(|S|.M.h.n^{2d})$ *steps.*

Proof. By Lemma 12, the depth of the proof-tree is at most $O(M.|S|.n^{2d})$: this bounds applies therefore for the overall parallel time needed to parse the proof-tree and reach all innermost $(/\!\!/, \multimap\ cut)$-rules. These innermost $(/\!\!/, \multimap\ cut)$ rules belong to different branches of the proof tree: they can therefore be eliminated safely in parallel. Each of these parallel elimination steps takes at most $O(M.h)$ steps.

By Lemma 11, the number of $(/\!\!/, \multimap\ cut)$-rules with cut-pairs on any path in the proof tree is bounded by $O(|S|.n^d)$: this bounds the number of times one needs to fully eliminate the innermost $(/\!\!/, \multimap\ cut)$-rules, hence the overall bound. ◀

Note that, in Theorem 18, we have only counted the number of parallel cut-elimination steps. Lemma 16 ensures moreover that, for each of these cut-elimination steps, the number of rule permutations needed to compute it is also polynomially bounded.

▶ **Example 19.** Let us consider the following derivation proof, corresponding to the application of a function to two arguments, of types A and B respectively, in a curryfied fashion, with atomic resulting type C. Since the strategy is innermost, the four premises in the tree are conclusions of (cut)-free derivation trees.

$$\dfrac{\dfrac{\dfrac{/\!\!/\Delta_1, \Gamma, A, B \vdash C}{/\!\!/\Delta_1, \Gamma, A \vdash B \multimap C}(\multimap R)}{/\!\!/\Delta_1, \Gamma \vdash A \multimap B \multimap C}(\multimap R) \qquad \dfrac{/\!\!/\Delta_3, \Phi \vdash A \qquad \dfrac{/\!\!/\Delta_5, \Lambda \vdash B \qquad /\!\!/\Delta_6, \Theta, C \vdash C}{/\!\!/\Delta_4, \Lambda, \Theta, B \multimap C \vdash C}(/\!\!/ \multimap L)}{/\!\!/\Delta_2, \Phi, \Lambda, \Theta, A \multimap B \multimap C \vdash C}(/\!\!/ \multimap L)}{/\!\!/\Delta, \Gamma, \Phi, \Lambda, \Theta \vdash C}(/\!\!/ cut)$$

One parallel $(/\!\!/ cut)$ elimination step exhibits the application of the first argument, of type A,

$$\dfrac{\dfrac{/\!\!/\Delta_1, \Gamma, A, B \vdash C}{/\!\!/\Delta_1, \Gamma, A \vdash B \multimap C}(\multimap R) \qquad /\!\!/\Delta_3, \Phi \vdash A \qquad \dfrac{/\!\!/\Delta_5, \Lambda \vdash B \qquad /\!\!/\Delta_6, \Theta, C \vdash C}{/\!\!/\Delta_4, \Lambda, \Theta, B \multimap C \vdash C}(/\!\!/ \multimap L)}{/\!\!/\Delta, \Gamma, \Phi, \Lambda, \Theta \vdash C}(/\!\!/ cut)$$

And a second one exhibits the application of the second argument, of type B.

$$\dfrac{/\!\!/\Delta_1, \Gamma, A, B \vdash C \qquad /\!\!/\Delta_3, \Phi \vdash A \qquad /\!\!/\Delta_5, \Lambda \vdash B \qquad /\!\!/ \Delta_6, \Theta, C \vdash C}{/\!\!/\Delta, \Gamma, \Phi, \Lambda, \Theta \vdash C}(/\!\!/ cut)$$

The premise $/\!\!/\Delta_6, \Theta, C \vdash C$ is the root of the pairing forest, and the atomic type C is eliminated first.

$$\dfrac{/\!\!/\Delta_1, \Gamma, A, B \vdash C \qquad /\!\!/\Delta_3, \Phi \vdash A \qquad /\!\!/\Delta_5, \Lambda \vdash B}{/\!\!/\Delta, \Gamma, \Phi, \Lambda, \Theta \vdash C}(/\!\!/ cut)$$

Finally, the two arguments types A and B are then eliminated in parallel, with elimination steps corresponding to the substitution of the corresponding values in the function term in the Curry-Howard isomorphism, as detailed in the next section.

3 A Parallel Polynomial Time Type Assignment for λ-calculus

3.1 Parallel Soft Types

We take insipiration from the STA type assignment of Gaboardi and Ronchi [7]. We add the parallel modalities in a restricted way, as follows.

▶ **Definition 20** (Parallel Soft types (PSTA)). *In the following, α, β, etc stand for base type variables, A, B, C, etc stand for types with linear output, and σ, τ, etc stand for PSTA types. PSTA types are given by the following grammar:*

$$\begin{aligned} A, B, C &:= \alpha \mid \sigma \multimap A \mid \forall \alpha A \mid A \& B \\ \sigma, \tau, \rho, \mu, \nu &:= A \mid !\sigma \mid \mathbin{/\!\!/} \sigma \end{aligned}$$

A PSTA Typing context is a set of type assignments $x : \sigma$, where x is a variable and σ a PSTA type. A PSTA Typing judgment is $\Gamma \vdash M : \sigma$, where Γ is a PSTA Typing context, M is a λ-term, and σ is a PSTA type.

3.2 Typing Rules

Our PSTA typing rules are the following.

$$\frac{}{\mathbin{/\!\!/}\Delta,\ x :\ A \vdash x :\ A}\ (\mathbin{/\!\!/} Id) \qquad \frac{\mathbin{/\!\!/}\Delta,\ \Gamma \vdash M :\ \sigma}{\mathbin{/\!\!/}\Delta,\ !\Gamma \vdash M : !\sigma}\ (\mathbin{/\!\!/} Sp) \qquad \frac{\Gamma \vdash M :\ A}{\Gamma \vdash M :\ \forall \alpha A}\ (\forall R)$$

$$\frac{\Gamma,\ x_0 :\ \tau, \cdots, x_n :\ \tau \vdash M :\ \sigma}{\Gamma,\ x : !\tau \vdash M[x/x_0, \cdots, x/x_n] :\ \sigma}\ (m) \qquad \frac{\Gamma \vdash M :\ \sigma_1 \qquad \Gamma \vdash M :\ \sigma_2}{\Gamma \vdash M :\ \sigma_1 \& \sigma_2}\ (\& R)$$

$$\frac{\Gamma,\ x :\ A[B/\alpha] \vdash M :\ \sigma}{\Gamma,\ x :\ \forall \alpha A \vdash M :\ \sigma}\ (\forall L) \qquad \frac{\Gamma,\ x_1 : \tau \vdash M :\ \sigma}{\Gamma,\ x : \mathbin{/\!\!/}\tau \vdash M[x/x_1] :\ \sigma}\ (\mathbin{/\!\!/} D)$$

$$\frac{\Gamma,\ x_1 : \tau_1 \vdash M :\ \sigma}{\Gamma,\ x : \tau_1 \& \tau_2 \vdash M[x/x_1] :\ \sigma}\ (\& L_1) \qquad \frac{\Gamma,\ x_2 : \tau_2 \vdash M :\ \sigma}{\Gamma,\ x : \tau_1 \& \tau_2 \vdash M[x/x_2] :\ \sigma}\ (\& L_2)$$

$$\frac{\Gamma,\ x :\ \sigma \vdash M :\ A}{\Gamma \vdash \lambda x.M :\ \sigma \multimap A}\ (\multimap R) \qquad \frac{\mathbin{/\!\!/}\Delta_1,\ \Gamma \vdash M :\ \tau \qquad \mathbin{/\!\!/}\Delta_2,\ \Lambda,\ x :\ \tau \vdash N :\ \sigma}{\mathbin{/\!\!/}\Delta,\ \Gamma,\ \Lambda, \vdash N[M/x] :\ \sigma}\ (\mathbin{/\!\!/} cut)$$

$$\frac{\Gamma \vdash M :\ \sigma}{\Gamma,\ x : \mathbin{/\!\!/}\tau \vdash M :\ \sigma}\ (\mathbin{/\!\!/} W) \qquad \frac{\mathbin{/\!\!/}\Delta_1,\ \Gamma \vdash M :\ \tau \qquad \mathbin{/\!\!/}\Delta_2,\ \Lambda,\ x :\ A \vdash N :\ \sigma}{\mathbin{/\!\!/}\Delta,\ \Gamma,\ \Lambda,\ y :\ \tau \multimap A \vdash N[yM/x] :\ \sigma}\ (\mathbin{/\!\!/} \multimap L)$$

As exemplified in the subject reduction property, typing an application term (MN) is done with the $(\mathbin{/\!\!/} \multimap L)$ rule. In the typing rules above, we also add the following side conditions:

- Parallel types occur only with negative polarity in the typing judgments,
- In rules $(\mathbin{/\!\!/} cut)$ and $(\mathbin{/\!\!/} \multimap L)$, the domain of contexts Γ and Λ are disjoint, and finally
- In rules $(\mathbin{/\!\!/} cut)$ and $(\mathbin{/\!\!/} \multimap L)$, the side condition $\mathcal{S}_P : \mathbin{/\!\!/}\Delta_1 \subsetneq \mathbin{/\!\!/}\Delta, \mathbin{/\!\!/}\Delta_2 \subsetneq \mathbin{/\!\!/}\Delta$ holds.

Moreover, we also define a generalized $(\mathbin{/\!\!/}, \multimap cut)$ rule similar to that of PSLL, with the appropriate nesting of substitutions for all (cut) and $\multimap$ pairs of terms.

These rules being literal translations of that of PSLL, the rule permutations, and (cut)-elimination steps of PSLL apply to PSTA.

The grammar of our types, and the typing rules, together with the side conditions above, ensure that sharing does not occur in our typing system. More precisely, we have

▶ **Proposition 21.** *Let Π be a typing derivation with conclusion $\Gamma \vdash M : !\sigma$. Then, the context Γ is $\mathbin{/\!\!/}\Delta,\ !\Lambda$.*

A corollary of Proposition 21 is

▶ **Corollary 22.** *Any typing derivation with conclusion $\mathbin{/\!\!/}\Delta,\ !\Gamma \vdash M : !\sigma$ ends with a $(\mathbin{/\!\!/} Sp)$, (m), $(\mathbin{/\!\!/} D)$, $(\mathbin{/\!\!/} W)$ or a $(\mathbin{/\!\!/} cut)$. Moreover, in this context, the rules (m), $(\mathbin{/\!\!/} D)$, and $(\mathbin{/\!\!/} W)$ can be commuted to the top (since the premise needs to have a modal context as well), and the derivation can w.l.o.g. be considered to end with a $(\mathbin{/\!\!/} Sp)$ or a $(\mathbin{/\!\!/} cut)$-rule.*

From the absence of sharing, we derive

▶ **Proposition 23.** *PSTA enjoys the subject reduction property: if* $\Gamma \vdash M : \sigma$ *and* $M \to_\beta M'$, *then* $\Gamma \vdash M'$.

Proof. By structural induction on the cut-type σ of the term $\lambda y.P$ in the redex $(\lambda y.P\ Q)$. The terms M and M' can be written as $M = N[(x\ Q)/z][\lambda y.P/x]$ and $M' = N[P[Q/y]/z]$. Two cases arise:

1. $\sigma = \tau \to A$. The derivation is

$$\dfrac{\dfrac{/\!\!/\Delta_1,\ \Gamma_1,\ y :\ \tau \vdash P :\ A}{/\!\!/\Delta_1,\ \Gamma_1 \vdash \lambda y.P :\ \tau \to A} \qquad \dfrac{/\!\!/\Delta_3,\ \Gamma_2, \vdash Q :\ \tau \qquad /\!\!/\Delta_4,\ \Gamma_3,\ z :\ A \vdash N :\ \sigma}{/\!\!/\Delta_2,\ \Gamma_2,\ \Gamma_3,\ x :\ \tau \to A \vdash N[(x\ Q)/z] :\ \sigma}\,(/\!\!/ \multimap L)}{/\!\!/\Delta,\ \Gamma_1,\ \Gamma_2,\ \Gamma_3 \vdash N[(x\ Q)/z][\lambda y.P/x] :\ \sigma}\,(/\!\!/cut)$$

   Cut elimination yields then the following derivation tree

$$\dfrac{/\!\!/\Delta_1,\ \Gamma_1,\ y :\ \tau \vdash P :\ A \qquad /\!\!/\Delta_3,\ \Gamma_2, \vdash Q :\ \tau \qquad /\!\!/\Delta_4,\ \Gamma_3,\ z :\ A \vdash N :\ \sigma}{/\!\!/\Delta,\ \Gamma_1,\ \Gamma_2,\ \Gamma_3 \vdash N[P[Q/y]/z] :\ \sigma}\,(/\!\!/cut)$$

   which proves the subject reduction property. If $\sigma = \forall\alpha\tau$ or $\sigma = \tau \& \tau'$, the cut-elimination steps eventually reduce to the case above.

2. $\sigma = !\tau$. By Proposition 21, and modulo rule permutations the derivation is

$$\dfrac{\dfrac{/\!\!/\Delta_1,\ \Gamma_1 \vdash \lambda y.P :\ \tau}{/\!\!/\Delta_1,\ !\Gamma_1 \vdash \lambda y.P : !\tau}\,(/\!\!/Sp) \qquad \dfrac{/\!\!/\Delta_2,\ \Gamma_2,\ \cdots x_i :\ \tau \cdots \vdash N[\cdots (x_i\ Q)/z_i \cdots] :\ \sigma}{/\!\!/\Delta_2,\ \Gamma_2,\ x : !\tau \vdash N[(x\ Q)/z_1, \cdots, (x\ Q)/z_n] :\ \sigma}\,(m)}{/\!\!/\Delta,\ !\Gamma_{1,;}\Gamma_2 \vdash N[(x\ Q)/z_1, \cdots, (x\ Q)/z_n][\lambda y.P/x] :\ \sigma}\,(/\!\!/cut)$$

   Cut elimination yields then the following derivation tree

$$\dfrac{\dfrac{\overbrace{\cdots /\!\!/\ \Delta_1,\ \Gamma_i' \vdash \lambda y.P_i :\ \tau \cdots}^{n \text{ copies}} \qquad /\!\!/\Delta_2,\ \Gamma_2,\ \cdots x_i :\ \tau \cdots \vdash N[\cdots (x_i\ Q)/z_i \cdots] :\ \sigma}{/\!\!/\Delta, \Gamma_1', \cdots, \Gamma_n', \Gamma_2 \vdash N[(x_1 Q)/z_1, \cdots, (x_n Q)/z_n][\lambda y.P_1/x_1, \cdots, \lambda y.P_n/x_n] : \sigma}\,(/\!\!/cut)}{/\!\!/\Delta, !\Gamma_1, \Gamma_2 \vdash N[(x_1\ Q)/z_1, \cdots, (x_n\ Q)/z_n][\lambda y.P_1/x_1, \cdots, \lambda y.P_n/x_n] :\ \sigma}\,(m)$$

   and the induction hypothesis applies to the n cut-types τ. ◀

3.3 A Parallel, Polynomial Time Evaluation Strategy

▶ **Theorem 24.** *Let* T *be a* λ*-term, typable in PSTA. Then,* T *normalizes in polynomial parallel time.*

Proof. The proof follows from Theorem 18: cut-elimination in parallel polynomial time, and subject-reduction, induce a parallel polynomial number of β-reduction steps for the term. The overall complexity bound is however a bit more subtle: while PSTA type derivations have exponential size and polynomial depth, the corresponding right-hand side λ-terms may have syntactic trees of exponential depth as well. Performing the substitutions for each β-reduction step in parallel polynomial time requires then to use an appropriate, polynomial space representation of the terms: the explicit representation is clearly unsuitable.

Let us first introduce some definitions and observations.

Let T be a λ-term. Its Böhm-like tree $B(T)$ is defined as follows:

1. If T is a variable x, $B(T)$ is a single vertex labelled with x.
2. If T is an abstraction $\lambda x.U$, $B(T)$ is obtained adding $B(U)$ as a leftmost child of a root labelled with λx.

3. If T is an application UV, $B(T)$ is obtained by adding $B(V)$ as a new rightmost child of the root of $B(U)$.

Clearly, a Böhm-like tree $B(T)$ uniquely defines a term T. Therefore, in the sequel we identify the two notions, and focus on the computation of the Böhm-like tree of the normal-form of a given term.

Let T be a λ-term, typable in PSTA with a typing derivation Π. We define the *pseudo*-derivation $D(\Pi)$ associated to Π as the tree obtained from Π by removing all right-hand side λ-terms (while keeping the corresponding type).

Then, the following observations hold.

1. In each typing judgment in Π (and therefore in $D(\Pi)$), the typing context contains type assignments for variable terms only.
2. Erasing the variable names in the contexts of $D(\Pi)$ (while keeping the corresponding types) yields a PSLL proof $L(\Pi)$, with types as formulae,
3. All right-hand side λ-terms in Π are uniquely determined by $D(\Pi)$, and finally,
4. The variable type assignments in $D(\Pi)$ are preserved by the subject-reduction property: If T_1 is a λ-term with PSTA type derivation Π_1 , $T_1 \to_\beta T_2$, and Π_2 is the type derivation of T_2 obtained by the subject reduction steps of Proposition 23, then the variable type assignments in $D(\Pi_1)$ and $D(\Pi_2)$ coincide.

As a consequence, the following reduction strategy holds: from a λ-term T with PSTA typing derivation Π, perform the innermost parallel cut-elimination strategy on $L(\Pi)$, while keeping the variable type assignments given by $D(\Pi)$. The observations above ensure that the pseudo derivation $D(\Pi')$ thus obtained is that of the typing derivation Π' of the normal form T' of T. The additional information stored in the contexts of $D(\Pi')$ (the variable names) takes polynomial space (polynomially many variable names among an exponential number of possible names), and the reduction can be performed in parallel polynomial time. It remains to show how to compute the normal form T' from its pseudo-derivation $D(\Pi')$, in parallel polynomial time. We do this by actually computing a succinct representation of its Böhm-like tree $B(T')$.

Let $D(\Pi)$ be the pseudo-derivation of a PSTA derivation Π, with corresponding term T with Böhm-like tree $B(T)$. A first observation is the following: For any typing judgment $\Gamma \vdash t : \sigma$ in Π, if the explicit substitution $[M/x]$ (respectively $[yM/x]$) occurs in t, then there exists a judgment $\Gamma' \vdash M : \sigma'$ above in Π. Since Π has polynomial depth, and polynomial indegree by Lemma 16, the substitution term M can then be described in polynomial space by the path from the conclusion of Π to this typing judgment $\Gamma' \vdash M : \sigma'$.

For each typing judgment $\Gamma \vdash t : \sigma$ in Π, we associate to the right-hand term the following:

1. the path p from the conclusion of Π to this judgment, and
2. the list $s(p)$ of explicit substitutions occurring along p, computed as follows:
   - assume p chooses the rightmost premise N in a $(⅋, \multimap\ cut)$ rule R (i.e. the premise p' s.t. R has no (cut) or $\multimap$-pair $p' \to p''$): this cut-rule introduces a polynomial number of substitutions $[M_i/x_i]$ (or $[y_i M_i/x_i]$) in its conclusion term. Then, we add to $s(p)$ the pairs (p_i, x_i) (or $(y_i p_i, x_i)$), where p_i is the path to the corresponding premise with right hand term M_i.
   - assume p passes through a (m) $(⅋D)$ or $(\&L_i)$. Then, we add to $s(p)$ the pairs (x, x_i).

Clearly, for a path p, the list $s(p)$ has polynomial size, and can be computed in polynomial time. Now, for a given path p, the computation of the corresponding vertex $v(p)$ in $B(T)$ proceeds co-recursively on $D(\Pi)$ as follows:

- if p is conclusion of a $(⅋ax)$ rule, $v(p)$ is a leaf in $B(T)$, with label x.

- if p is conclusion of a $(/\!\!/ Sp)$, $(\forall R)$, $(\forall L)$, $(/\!\!/ W)$ or $(\&R)$ rule, with premise p', then $v(p)$ is $v(p')$.
- if p is conclusion of a (m), $(/\!\!/ D)$ or $(\&L_i)$ rule with premise p', two cases arise:
  1. $v(p')$ is labelled x_i: then, (x/x_i) belongs to $s(p)$. In that case $v(p)$ is labelled x, and its successors are those of $v(p')$.
  2. otherwise, $v(p)$ is $v(p')$.
- if p is conclusion of a $(\multimap R)$ rule with premise p', $v(p)$ is an inner node labelled λx, with left successor node p'.
- if p is conclusion of a $(/\!\!/, \multimap cut)$-rule R: let p' be its rightmost premise. Then, three cases arise:
  1. $v(p')$ is labelled x, (p'', x) belongs to $s(p')$: then, $v(p)$ is obtained from $v(p'')$ by adding to its root the successor vertices of $v(p')$.
  2. $v(p')$ is labelled x, (yp'', x) belongs to $s(p')$: then, $v(p)$ is a vertex labelled y, with right successor $v(p'')$.
  3. otherwise, $v(p)$ is $v(p')$.

Performing the procedure above in parallel for all paths in $D(\Pi)$ provides then a succinct description of $B(T)$ in parallel polynomial time. ◀

4 Completeness of PSTA

We now prove that PSTA is complete for the class FPAR of functions computable in parallel, polynomial time. In order to do so, we first encode parallel, polynomial time recursive functions with substitutions, à la Leivant and Marion [16], and then use them to simulate the computation of a P-uniform family of boolean circuits of polynomial depth. Extending these encodings to the setting of algebraic complexity amounts then simply to replace the base type $\mathbf{B}$ by a base type for the underlying algebraic structure (e.g. real numbers), and to provide the type of the algebraic constants and operations in the typing context.

First, PSTA captures (obviously) STA.

▶ **Lemma 25.** *Let Π be a SLL proof of degree d and rank n, with conclusion $\Gamma \vdash A$ of size s. Let W_Π be its weight, as defined in [13]. Then, any path in from the conclusion of Π to an axiom contains at most $s + W_\Pi(1)$ $(\multimap L)$ rules, and at most $W_\Pi(1).n^d$ (cut) rules.*

▶ **Corollary 26.** *Let Π be a SLL proof of degree d and rank n, with conclusion $\Gamma \vdash A$ of size s. Then, there exists a PSLL proof Π' with conclusion $\Delta, \Gamma \vdash A$, of degree d and rank n.*

Proof. Take $\Delta = !^d /\!\!/ A_1, \ldots, !^d /\!\!/ A_k$, with $k = W_\Pi(1) + s$, for any $A_1, \cdots, A_k$. ◀

An immediate consequence is that all λ-terms typable in STA are also typable in PSTA, with the same rank and degree. As a consequence, following [7], Theorem 19, we immediately have that PSTA is complete for FPTIME. This allows us to prove its FPAR completeness more easily. Denote by $\mathbf{B}$ the STA (hence PSTA) type for booleans, $\mathbf{L}$ the STA type for binary strings, and $\mathbf{N}$ the STA type for Church Integers.

The following lemma allows us to encode some sort of polynomial recursion with substitutions a la Leivant and Marion [16] in PSTA.

▶ **Lemma 27.** *Assume we have the following sequential, polynomial time functions, with PSTA type derivations:*

- $\mathtt{op}$ *with derivation* $\Pi_{\mathtt{op}}$ *with conclusion* $\Gamma_{\mathtt{op}} \vdash \mathtt{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L}$.
- $\mathtt{s}_1$ *and* $\mathtt{s}_2$ *with derivation* Π_i*, for* $i \in \{1, 2\}$*, with conclusion* $\Gamma_i \vdash \mathtt{s}_i : \mathbf{L} \multimap \mathbf{L}$.

- *and, for any univariate polynomial P of degree d, a function $\overline{P}$, encoding the church integer $P(|L|)$ for a binary list L, with derivation $\Pi_{\overline{P}}$ with conclusion $\Gamma_{\overline{P}} \vdash \overline{P} : !^d\mathbf{L} \multimap \mathbf{N}$.*

We now consider the following recursive function with substitutions, on binary lists: $f(v) = \mathtt{op}(v, f(\mathtt{s}_1(v)), f(\mathtt{s}_2(v)))$. Moreover, we assume that on any input v, the recursive computation of f reaches a fixed point after $P(|v|)$ steps. Then, $f(v)$ is PSTA definable with degree d.

Proof. Following a similar encoding in [6], each recursion step in the computation of f is encoded by the following function $\mathtt{Step} = \lambda h.\lambda v.\mathtt{op}\ v\ (h\ (\mathtt{s}_1\ v))\ (h\ (\mathtt{s}_2\ v))$. Let $\mathbf{L2} = (\mathbf{L} \multimap \mathbf{L}) \multimap \mathbf{L} \multimap \mathbf{L}$, and $\Gamma = \Gamma_{\mathtt{op}}, \Gamma_1, \Gamma_2, (X : /\!\!/A)^2, h : /\!\!/(\mathbf{L} \multimap \mathbf{L}), v : /\!\!/\mathbf{L}$ Then, $\mathtt{Step}$ admits a PSTA proof derivation $\Pi_{\mathtt{Step}}$ with conclusion $\Gamma \vdash \mathtt{Step} : \mathbf{L2}$.

Indeed, $\Pi_{\mathtt{Step}}$ is

$$\dfrac{\dfrac{A_{\mathtt{op}} \quad A_1 \quad A_2 \quad A_{\mathtt{s}}1 \quad A_{\mathtt{s}}2 \quad A_{\mathtt{v}}1 \quad A_{\mathtt{v}}2 \quad A_{\mathtt{step}}}{\Gamma, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash \mathtt{op}\ v\ (h\ (\mathtt{s}_1\ v))\ (h\ (\mathtt{s}_2\ v)) : \mathbf{L}}\,(/\!\!/cut)}{\Gamma \vdash \mathtt{Step} : \mathbf{L2}}\,(\multimap R)^2$$

where $A_{\mathtt{op}}$ is $\dfrac{\Pi_{\mathtt{op}}}{\Gamma_{\mathtt{op}} \vdash \mathtt{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L}}$, A_i is $\dfrac{\Pi_i}{\Gamma_i \vdash \mathtt{s}_i : \mathbf{L} \multimap \mathbf{L}}$,

$A_{\mathtt{s}}i$ is $\dfrac{\dfrac{\dfrac{}{v : \mathbf{L} \vdash v : \mathbf{L}}\,(/\!\!/Id) \quad \dfrac{}{x : \mathbf{L} \vdash x : \mathbf{L}}\,(/\!\!/Id)}{X : /\!\!/A, \mathtt{t}_i : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash \mathtt{t}_i\ v : \mathbf{L}}\,(/\!\!/ \multimap L)}{X : /\!\!/A, \mathtt{t}_i : \mathbf{L} \multimap \mathbf{L}, v : /\!\!/\mathbf{L} \vdash \mathtt{t}_i\ v : \mathbf{L}}\,(/\!\!/D)$,

$A_{\mathtt{v}}1$ is $\dfrac{\dfrac{\dfrac{}{v : \mathbf{L} \vdash v : \mathbf{L}}\,(/\!\!/Id) \quad \dfrac{}{x : \mathbf{L} \vdash x : \mathbf{L}}\,(/\!\!/Id)}{X : /\!\!/A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h\ v : \mathbf{L}}\,(/\!\!/ \multimap L)}{X : /\!\!/A, h : /\!\!/\mathbf{L} \multimap \mathbf{L}, v : /\!\!/\mathbf{L} \vdash h\ v : \mathbf{L}}\,(/\!\!/D)^2$,

$A_{\mathtt{v}}2$ is $\dfrac{\dfrac{\dfrac{}{v : \mathbf{L} \vdash v : \mathbf{L}}\,(/\!\!/Id) \quad \dfrac{}{x : \mathbf{L} \vdash x : \mathbf{L}}\,(/\!\!/Id)}{X : /\!\!/A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h\ v : \mathbf{L}}\,(/\!\!/ \multimap L)}$,

and $A_{\mathtt{step}}$ is

$$\dfrac{\dfrac{}{V_1 : \mathbf{L} \vdash V_1 : \mathbf{L}}\,(/\!\!/Id) \quad \dfrac{}{x_1 : \mathbf{L} \vdash x_1 : \mathbf{L}}\,(/\!\!/Id) \cdots \quad \dfrac{}{x_3 : \mathbf{L} \vdash x_3 : \mathbf{L}}\,(/\!\!/Id)}{X : /\!\!/A, V_1 : \mathbf{L}, V_2 : \mathbf{L}, V_3 : \mathbf{L}, \mathtt{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \vdash \mathtt{op}\ V_1\ V_2\ V_3 : \mathbf{L}}\,(/\!\!/, \multimap cut)$$

The value $f(v)$ is reached after $P(|v|)$ recursion steps. It is given by $\mathtt{Value}\ v$, where $\mathtt{Value} = \lambda v.((\overline{P}\ v)\ \mathtt{Step}\ \lambda y.y)\ v)$. Let $\Gamma' = \Gamma_{\mathtt{op}}, \Gamma_1, \Gamma_2, (X : /\!\!/A)^5, h : /\!\!/(\mathbf{L} \multimap \mathbf{L}), v : /\!\!/\mathbf{L}, \Gamma_{\overline{P}}, v :!^d\mathbf{L}$. Then, $\mathtt{Value}$ admits a PSTA proof derivation $\Pi_{\mathtt{Value}}$ with conclusion $\Gamma' \vdash \mathtt{Value} : \mathbf{L} \multimap \mathbf{L}$. Indeed, let $\mathbf{L3} = (!\mathbf{L2} \multimap \mathbf{L2})$. Recall that $\mathbf{N} = \forall\alpha!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$ and consider the following proof derivations.

$\Pi_{\overline{P}\ v}$:

$$\dfrac{\dfrac{\Pi_{\overline{P}}}{\Gamma_{\overline{P}} \vdash \overline{P} :!^d\mathbf{L} \multimap \mathbf{N}} \quad \dfrac{\dfrac{}{v :!^d\mathbf{L} \vdash v :!^d\mathbf{L}}\,(/\!\!/Id) \quad \dfrac{}{x :!^d\mathbf{L} \vdash x :!^d\mathbf{L}}\,(/\!\!/Id)}{x : /\!\!/A, \overline{P} :!^d\mathbf{L} \multimap \mathbf{N}, v :!^d\mathbf{L} \vdash \overline{P}\ v : \mathbf{N}}\,(/\!\!/ \multimap L)}{(x : /\!\!/A)^2, \Gamma_{\overline{P}}, v :!^d\mathbf{L} \vdash \overline{P}\ v : \mathbf{N}}\,(/\!\!/cut)$$

Π_s:

$$\dfrac{\Pi_{\overline{P}\ v} \qquad !\Pi_{\texttt{Step}} \qquad \dfrac{\dfrac{!\texttt{Step} :!\mathbf{L2} \vdash !\texttt{Step} :!\mathbf{L2} \qquad x :!\mathbf{L2} \vdash x :!\mathbf{L2}}{X : /\!\!/ A, (\overline{P}\ v) : \mathbf{L3}, !\texttt{Step} :!\mathbf{L2} \vdash \overline{P}\ v\ \texttt{Step} : \mathbf{L2}}\ (/\!\!/ \multimap L)}{X : /\!\!/ A, (\overline{P}\ v) : \mathbf{N}, !\texttt{Step} :!\mathbf{L2} \vdash \overline{P}\ v\ \texttt{Step} : \mathbf{L2}}\ (\forall L)}{\Gamma_{\texttt{op}}, \Gamma_1, \Gamma_2, (X : /\!\!/ A)^3, h : /\!\!/(\mathbf{L} \multimap \mathbf{L}), v : /\!\!/ \mathbf{L}, \Gamma_{\overline{P}}, v :!^d\mathbf{L} \vdash \overline{P}\ v\ \texttt{Step} : \mathbf{L2}}\ (/\!\!/ cut)$$

Π_{sl}:

$$\dfrac{\Pi_s \qquad \dfrac{\lambda y.y : \mathbf{L} \multimap \mathbf{L} \vdash \lambda y.y : \mathbf{L} \multimap \mathbf{L} \qquad f : \mathbf{L} \multimap \mathbf{L} \vdash f : \mathbf{L} \multimap \mathbf{L}}{X : /\!\!/ A, \overline{P}\ v\ \texttt{Step} : \mathbf{L2} \vdash \overline{P}\ v\ \texttt{Step}\ \lambda y.y : \mathbf{L} \multimap \mathbf{L}}\ (/\!\!/ \multimap L)}{\Gamma_{\texttt{op}}, \Gamma_1, \Gamma_2, (X : /\!\!/ A)^4, h : /\!\!/(\mathbf{L} \multimap \mathbf{L}), v : /\!\!/ \mathbf{L}, \Gamma_{\overline{P}}, v :!^d\mathbf{L} \vdash \overline{P}\ v\ \texttt{Step}\ \lambda y.y : \mathbf{L} \multimap \mathbf{L}}\ (/\!\!/ cut)$$

and finally $\Pi_{\texttt{Value}}$:

$$\dfrac{\dfrac{\Pi_{sl} \qquad \dfrac{v : \mathbf{L} \vdash v : \mathbf{L} \qquad z : \mathbf{L} \vdash z : \mathbf{L}}{X : /\!\!/ A, \overline{P}\ v\ \texttt{Step}\ \lambda y.y : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash (\overline{P}\ v\ \texttt{Step}\ \lambda y.y)v : \mathbf{L}}\ (/\!\!/ \multimap L)}{\Gamma', v : \mathbf{L} \vdash (\overline{P}\ v\ \texttt{Step}\ \lambda y.y)v : \mathbf{L}}\ (/\!\!/ cut)}{\Gamma' \vdash \texttt{Value} : \mathbf{L} \multimap \mathbf{L}}\ (\multimap R)$$

◀

▶ **Theorem 28.** *PSTA is complete for FPAR.*

Proof. Using the usual encodings for binary strings, booleans, integers and pairs, we use Lemma 27 to prove our completeness result. Let g be a function computed in FPAR. For the sake of simplicity let us assume that g outputs a single boolean. Then, there exists a P-uniform family $\mathcal{C}$ of succinctly described boolean circuits, of polynomial depth, computing g. More precisely, there exist a univariate polynomial p, and polynomial time functions $\texttt{and}$, $\texttt{or}$, $\texttt{node}$, $\texttt{input}$, $\texttt{s}_1$ and $\texttt{s}_2$ such that:

- On any input $x = x_1, \cdots, x_n$ of size n, $g(x_1, \cdots, x_n)$ is computed by a boolean circuit C_n of depth $p(n)$, with output node t.
- For each node s in C_n, there exists a binary list n_s, encoding a path from t to s in C_n, of length less than $p(n)$. Each node will be identified by these paths (there may be several paths for a given node).
- $\texttt{and}(x, y)$ (respectively $\texttt{or}(x, y)$, resp. $\texttt{not}(x, y)$) is $\texttt{true}$ if the path y encodes a $\texttt{and}$ (resp. $\texttt{or}$, resp. $\texttt{not}$) node of $C_{|x|}$, and $\texttt{false}$ otherwise.
- $\texttt{input}(x, y)$ is $(x_i, \texttt{true})$ if y encodes the i^{th} input node of $C_{|x|}$, and $(0, \texttt{false})$ otherwise.
- $\texttt{s}_1(y) = 0.y$ encodes a path to the left parent of the node encoded by y, if it exists.
- $\texttt{s}_2(y) = 1.y$ encodes a path to the right parent of the node encoded by y, if it exists.

Define now $f(x, y) = \texttt{op}(x, y, f(x, \texttt{s}_1(y)), f(x, \texttt{s}_1(y)))$, where y denotes a path in $C_{|x|}$, and $\texttt{op}(x, y, v_1, v_2)$ computes, using the functions defined above, the boolean value of the node y in $C_{|x|}$, provided v_1 and v_2 are the boolean values of its two parents nodes. Then, Lemma 27 applies: f is definable in PSTA, and recursively computes the value of all nodes in $C_{|x|}$. The output $g(x)$ is then given by $f(x, \epsilon)$, where ϵ is the empty binary list. ◀

5 Concluding Remarks

In this paper we have only investigated one of the many possible choices for the way the parallel $(/\!\!/, \multimap cut)$ rule allows contraction on $/\!\!/$ formulas, and allows its distribution among the premises of the cut, and we have applied this approach to one example (STA) of linear typing system. Among the questions now worth investigating are the following: Is it possible to tune differently the side condition of the $(/\!\!/, \multimap cut)$-rule to capture other complexity classes? Such obvious candidates are the classes NC^i, which we could hope to capture

by taking a fully linear ($/\!\!/, \multimap$ *cut*)-rule (for ensuring sequential polynomial time), with an additional side condition ensuring parallel polylogarithmic time cut-elimination. Is it also possible to use this approach on type systems capturing other sequential complexity classes, for instance Logspace [17, 15], and to obtain other interesting results?

References

1 Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: A language for polynomial time computation. In Igor Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004. doi:10.1007/978-3-540-24727-2_4.

2 Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1):41–62, January 2009. doi:10.1016/j.ic.2008.08.005.

3 Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

4 Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Two function algebras defining functions in NC^{k} boolean circuits. *Inf. Comput.*, 248:82–103, 2016. doi:10.1016/j.ic.2015.12.009.

5 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of pspace. In George C. Necula and Philip Wadler, editors, *POPL*, pages 121–131. ACM, 2008. doi:10.1145/1328438.1328456.

6 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. An implicit characterization of PSPACE. *ACM Transactions on Computational Logic*, 13(2):1–36, April 2012. doi:10.1145/2159531.2159540.

7 Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for *ambda* -calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2007. doi:10.1007/978-3-540-74915-8_21.

8 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

9 Jean-Yves Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.

10 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992.

11 Paulin Jacobé De Naurois. Pointers in recursion: Exploring the tropics. In *FSCD*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2019. doi:10.4230/LIPICS.FSCD.2019.29.

12 Satoru Kuroda. Recursion schemata for slowly growing depth circuit classes. *Computational Complexity*, 13(1-2):69–89, 2004. doi:10.1007/s00037-004-0184-4.

13 Yves Lafont. Soft linear logic and polynomial time. *Theor. Comput. Sci.*, 318(1-2):163–180, 2004. doi:10.1016/j.tcs.2003.10.018.

14 Ugo Dal Lago and Martin Hofmann. Bounded linear logic, revisited. *Logical Methods in Computer Science*, 6(4), December 2010. doi:10.2168/lmcs-6(4:7)2010.

15 Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *Programming Languages and Systems*, pages 205–225. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-11957-6_12.

16 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity ii: Substitution and poly-space. In Leszek Pacholski and Jerzy Tiuryn, editors, *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 1994. doi:10.1007/BFb0022242.

17 Ulrich Schöpp. Stratified bounded affine logic for logarithmic space. In *LICS*, pages 411–420. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.45.

Encoding Tight Typing in a Unified Framework

Delia Kesner
Université de Paris, CNRS, IRIF, France
Institut Universitaire de France (IUF), France

Andrés Viso
Inria, Paris, France

Abstract

This paper explores how the intersection type theories of call-by-name (CBN) and call-by-value (CBV) can be *unified* in a more general framework provided by call-by-push-value (CBPV). Indeed, we propose *tight* type systems for CBN and CBV that can be both encoded in a unique *tight* type system for CBPV. All such systems are quantitative, *i.e.* they provide *exact* information about the length of normalization sequences to normal form as well as the size of these normal forms. Moreover, the length of reduction sequences are discriminated according to their multiplicative and exponential nature, a concept inherited from linear logic. Last but not least, it is possible to extract quantitative measures for CBN and CBV from their corresponding encodings in CBPV.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Call-by-Push-Value, Call-by-Name, Call-by-Value, Intersection Types

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.27

Related Version *Full Version*: `https://arxiv.org/abs/2105.00564`

Funding This work has been partially supported by IRP SINFIN.

Acknowledgements We are specially grateful to G. Guerrieri for fruitful discussions. We also thank B. Accattoli, A. Bucciarelli, and A. Ríos for constructive remarks.

1 Introduction

Every programming language implements a particular evaluation strategy which specifies when and how parameters are evaluated during function calls. For example, in **call-by-value (CBV)**, the argument is evaluated before being passed to the function, while in **call-by-name (CBN)** the argument is substituted directly into the function body, so that the argument may never be evaluated, or may be re-evaluated several times. CBN and CBV have always been studied independently, by developing different techniques for one and the other, until the remarkable observation that they are two different instances of a more general framework introduced by Girard's **Linear Logic (LL)**. Their (logical) duality ("CBN is *dual* to CBV") was understood later [18, 17]. And their rewriting semantics were finally *unified* by the **call-by-push-value (CBPV)** paradigm – introduced by P.B. Levy [41, 42] – a formalism being able to capture different functional languages/evaluation strategies.

A typical aspect that one wants to compare between two different evaluation strategies is the *number of steps* that are necessary to get a result. Such numbers are extracted from different models of computation and should be then measured by *compatible* instruments, either by means of common quantitative tools, or by a precise transformation between them[1]. Thus, we provide a *uniform* tool to measure quantitative information extracted from the evaluation of programs in different programming languages (namely CBN and CBV).

[1] Think for example about cm and inches.

More concretely, we introduce typing systems capturing **qualitative** and **quantitative** information of programs. From a qualitative point of view, the typing systems characterize termination of programs, *i.e.* a program p is typable if and only if p terminates. From a quantitative point of view, the typing systems provide **exact/tight** information about the **number of steps** needed to get a result, as well as the **size** of this result. All these questions are addressed in the general framework of CBPV, being able to encode CBN and CBV, both from a static and a dynamic point of view. Dynamically, CBN and CBV evaluation strategies are known to be encodable by the rewriting semantics of CBPV. Statically, we define tight typing systems providing quantitative information for CBN and CBV, that can be seen as particular cases of the tight quantitative typing system behind the unified framework of CBPV. We now explain all these concepts in more detail.

Quantitative Types. Quantitative typing systems are often specified by non-idempotent intersection types inspired by the relational semantics of LL [30, 11], as pioneered by [28]. This connection makes non-idempotent types not only a qualitative typing tool to reason about programming languages, but mainly a quantitative one, being able to specify properties related to the consumption of resources, a remarkable investigation pioneered by the seminal de Carvalho's PhD thesis [19] (see also [21]). Thus, qualitatively, a non-idempotent typing system is able to fully **characterise** normalisation, in the sense that a term t is typable if and only if t is normalising. More interestingly, quantitative typing systems also provide **upper bounds**, in the sense that the length of any reduction sequence from t to normal form *plus* the size of this normal form is *bounded* by the size of the type derivation of t. Therefore, typability characterises normalisation in a qualitative as well as in a quantitative way, but only provides upper bounds. Several papers explore bounded measures of non-idempotent types for different higher order languages. Some references are [24, 31, 4, 3, 35, 14, 20, 33, 23, 22, 13, 44].

In this satisfactory enough? A major observation concerning β-reduction in λ-calculus is that the size of normal forms can be *exponentially* bigger than the number of steps needed to reach these normal forms. This means that bounding the sum of these two integers *at the same* time is too rough, and not very relevant from a quantitative point of view. Fortunately, it is possible to extract better (*i.e.* independent and exact) measures from a non-idempotent intersection type system. A crucial point to obtain **exact measures**, instead of upper bounds, is to consider *minimal* type derivations [19, 9, 23]. Therefore, *upper bounds* for time *plus* size can be refined into *independent exact measures* for time *and* size [1]. More precisely, the quantitative typing systems are now equipped with constants and *counters*, together with an appropriate notion of **tightness**, which encodes minimality of type derivations. For any *tight* type derivation Φ of a term t ending with (independent) counters (b, s), it is now possible to show that t is normalisable in b steps and its normal form has size s, so that the type system is able to *guess* the number of steps to normal form as well as the size of this normal form. The opposite direction also holds: if t normalises in b steps to a normal form size s, then it is possible to tightly type t by using (independent) counters (b, s).

In this paper we design tight quantitative type systems that are also capable to discriminate between **multiplicative** and **exponential** evaluation steps to normal form, two conceptual notions coming from LL: *multiplicative* steps are essentially those that (linearly) reconfigure proofs/terms/programs, while *exponential* steps are the only ones that are potentially able to erase/duplicate other objects. As a consequence, for any *tight* type derivation Φ of t ending with (independent) counters (m, e, s), the term t is normalisable in m multiplicative and e exponential steps to a normal form having size s. The opposite direction also holds.

Call-by-push-value. CBPV extends the λ-calculus with two primitives `thunk` and `force` distinguishing between *values* and *computations*: the former freezes the execution of a term (*i.e.* turns a computation into a value) while the latter fires again a frozen term (*i.e.* turns a value into a computation). These primitives allow to capture the duality between CBN and CBV by conveniently labelling a λ-term with `thunk`/`force` to pause/resume the evaluation of a subterm. Thus, CBPV provides a *unique* and *general* formalism capturing different functional strategies, and allowing to *uniformly* study operational and denotational semantics of different programming languages through a single tool.

In this paper we model the CBPV paradigm by using the $\lambda!$-calculus [12], based on the *bang calculus* introduced in [26], which in turn extends ideas by T. Ehrhard [25]. The granularity of the $\lambda!$-calculus, expressed with both **explicit substitutions** and **reduction at a distance** (details in Sec. 5), clearly allows to differentiate between *multiplicative* and *exponential* steps, as in LL. The corresponding CBN and CBV strategies also follow this pattern: it is possible to distinguish the multiplicative steps that only reconfigure pieces of syntax, from the exponential steps used to implement erasure and duplication of terms.

Contributions. We first define deterministic strategies for CBN and CBV that are able to discriminate between multiplicative and exponential steps (Sec. 2).

We then formulate tight typing systems for both CBN (Sec. 3) and CBV (Sec. 4), called respectively $\mathcal{N}$ and $\mathcal{V}$. System $\mathcal{N}$ is a direct extension of Gardner's system [28], while system $\mathcal{V}$ is completely new, and constitutes one of the major contributions of this paper. A key feature of system $\mathcal{V}$ is its ability to distinguish between the two different roles that variables may play in CBV depending on the context where they are placed, *i.e.* to be a placeholder for a *value* or the head of a *neutral* term. We show that both systems implement independent measures for time and size, and that they are quantitatively sound and complete. More precisely, we show that tight (*i.e.* minimal) typing derivations in such systems (exactly) quantitatively characterise normalisation, *i.e.* if Φ is a *tight* type derivation of t in system $\mathcal{N}$ (resp. $\mathcal{V}$), ending with counters (m, e, s), then there exists a CBN (resp. CBV) normal form p of size s such that t reduces to p by using exactly m multiplicative steps and e exponential steps. The converse, giving quantitative completeness of the approach, also holds.

Sec. 5 recalls the bang calculus at a distance $\lambda!$ and Sec. 6 presents its associated tight type system $\mathcal{B}$, together with their respective quantitative sound and complete properties. The untyped CBN/CBV translations into $\lambda!$ are recalled in Sec. 7, while the typed translations are defined and discussed in Sec. 8, the other major contribution of this work. Through these typed encodings, the counters of the source and target derivations are related. This makes it possible to give the precise cost of our typed translations, as well as to extract quantitative measures for CBN and CBV from their corresponding encodings in CBPV.

Detailed proofs can be found in [37].

Related Work. For CBN, non-idempotent types were introduced by [28], their quantitative power was extensively studied in [19, 20], and their tight extensions in [9, 1]. For CBV, non-idempotent types were introduced in [24], and extensively studied, *e.g.* [31, 3]. A tight extension being able to count reduction steps was recently defined in [4, 40] for a special version of (closed) CBV, but it is not clear how this could be encoded in a linear logic based CBPV framework. Another non-idempotent type system was also recently introduced for CBV [43, 34], it is not tight and does not translate to CBPV.

A (non-tight) quantitative type system for the bang calculus, based on a relational model, can be found in [32]. Another relational model that can be seen as a non-tight system for CBPV was introduced by [16]. Following ideas in [19, 9, 1], a type system $\mathcal{E}$ was

proposed in [12] to fully exploit tight quantitative aspects of the $\lambda!$-calculus: *independent exact measures* for time and size are guessed by the types system. However, no relation between tightness for CBN/CBV and tightness for the $\lambda!$-calculus are studied in *op.cite.* This papers fills this gap.

The discrimination between multiplicative and exponential steps by means of tight quantitative types can be found *e.g.* in CBN [1], call-by-need [4], and languages with pattern-matching primitives [7].

2 Call-by-Name and Call-by-Value

This section introduces the CBN and CBV specifications being able to distinguish between multiplicative and exponential steps, as in linear logic. Given a countably infinite set $\mathcal{X}$ of variables $x, y, z, \ldots$, we consider the following grammars for terms ($\mathcal{T}_\lambda$), values and contexts:

$$\begin{array}{rrcl}
\textbf{\textit{(Terms)}} & t,u,r & ::= & v \mid t\,u \mid t[x\backslash u] \\
\textbf{\textit{(Values)}} & v & ::= & x \in \mathcal{X} \mid \lambda x.t \\
\textbf{\textit{(Contexts)}} & \mathtt{C} & ::= & \mathtt{L} \mid \mathtt{N} \mid \mathtt{V} \\
\textbf{\textit{(List Contexts)}} & \mathtt{L} & ::= & \Box \mid \mathtt{L}[x\backslash t] \\
\textbf{\textit{(CBN Contexts)}} & \mathtt{N} & ::= & \Box \mid \mathtt{N}\,t \mid \lambda x.\mathtt{N} \mid \mathtt{N}[x\backslash u] \\
\textbf{\textit{(CBV Contexts)}} & \mathtt{V} & ::= & \Box \mid \mathtt{V}\,t \mid t\,\mathtt{V} \mid \mathtt{V}[x\backslash u] \mid t[x\backslash \mathtt{V}]
\end{array}$$

A terms of the form $t[x\backslash u]$ is a ***closure***, and $[x\backslash u]$ an ***explicit substitution*** (ES). Special terms are $\mathtt{I} = \lambda z.z$, $\mathtt{K} = \lambda x.\lambda y.x$, $\Delta = \lambda x.x\,x$, and $\Omega = \Delta\,\Delta$. We use $\mathtt{C}\langle t\rangle$ for the term obtained by replacing the hole $\Box$ of $\mathtt{C}$ by t. ***Free*** and ***bound*** variables, as well as α-conversion, are defined as expected. In particular, $\mathtt{fv}(t[x\backslash u]) \stackrel{def}{=} \mathtt{fv}(t)\setminus\{x\}\cup\mathtt{fv}(u)$, $\mathtt{fv}(\lambda x.t) \stackrel{def}{=} \mathtt{fv}(t)\setminus\{x\}$, $\mathtt{bv}(t[x\backslash u]) \stackrel{def}{=} \mathtt{bv}(t)\cup\{x\}\cup\mathtt{bv}(u)$ and $\mathtt{bv}(\lambda x.t) \stackrel{def}{=} \mathtt{bv}(t)\cup\{x\}$. The notation $t\,\{x\backslash u\}$ is used for the (capture-free) ***meta-level*** substitution operation, defined, as usual, modulo α-conversion. Special predicates are used to distinguish different kinds of terms surrounded by ES: $\mathsf{abs}(t)$ iff $t = \mathtt{L}\langle\lambda x.u\rangle$, $\mathsf{app}(t)$ iff $t = \mathtt{L}\langle r\,u\rangle$ and $\mathsf{var}(t)$ iff $t = \mathtt{L}\langle x\rangle$. Finally, $\mathsf{val}(t)$ iff $\mathsf{abs}(t)$ or $\mathsf{var}(t)$.

As mentioned in the introduction, our aim is to count the reduction steps by distinguishing their multiplicative and exponential nature. To achieve this, the standard specifications of CBN/CBV are not adequate, so we need to consider alternative appropriate definitions [6] making use of the following three different rewriting rules:

$$\begin{array}{rrcl}
(\textbf{d}\text{istant }\texttt{B}\text{eta}) & \mathtt{L}\langle\lambda x.t\rangle\,u & \mapsto_{\mathtt{dB}} & \mathtt{L}\langle t[x\backslash u]\rangle \\
(\textbf{s}\text{ubstitute term}) & t[x\backslash u] & \mapsto_{\mathtt{sn}} & t\,\{x\backslash u\} \\
(\textbf{s}\text{ubstitute value}) & t[x\backslash \mathtt{L}\langle v\rangle] & \mapsto_{\mathtt{sv}} & \mathtt{L}\langle t\,\{x\backslash v\}\rangle
\end{array}$$

Rule $\mathtt{dB}$ fires β-reduction *at a distance* by combining the two more elementary rules: $(\lambda x.t)\,u \mapsto t[x\backslash u]$ and $\mathtt{L}\langle t\rangle\,u \mapsto \mathtt{L}\langle t\,u\rangle$, where the second one is a structural/permutation rule pushing out ES that may block β-redexes. Rule $\mathtt{sn}$ implements standard substitution, while $\mathtt{sv}$ restricts substitution to values and acts *at a distance* by combining the two more elementary rules: $t[x\backslash v] \mapsto t\,\{x\backslash v\}$ and $t[x\backslash \mathtt{L}\langle v\rangle] \mapsto \mathtt{L}\langle t[x\backslash v]\rangle$. The ***call-by-name*** reduction relation $\to_{\mathtt{n}}$ is the closure by contexts $\mathtt{N}$ of the rules $\mathtt{dB}$ and $\mathtt{sn}$, while the ***call-by-value*** reduction relation $\to_{\mathtt{v}}$ is the closure by contexts $\mathtt{V}$ of the rules $\mathtt{dB}$ and $\mathtt{sv}$. Equivalently,

$$\to_{\mathtt{n}} := \mathtt{N}(\mapsto_{\mathtt{dB}} \cup \mapsto_{\mathtt{sn}}) \quad \text{and} \quad \to_{\mathtt{v}} := \mathtt{V}(\mapsto_{\mathtt{dB}} \cup \mapsto_{\mathtt{sv}})$$

The resulting CBN/CBV formulations are now based on distinguished ***multiplicative*** (*cf.* $\mathtt{dB}$) and ***exponential*** (*cf.* $\mathtt{sn}$ and $\mathtt{sv}$) steps, called resp. $\mathtt{m}$-steps and $\mathtt{e}$-steps, thus inheriting the nature of cut elimination rules in LL. Notice that the number of $\mathtt{m}$ and $\mathtt{e}$-steps in a

normalization sequence is not always the same: *e.g.* $x[x\backslash y] \to_{\mathtt{n}} y$ has only one $\mathtt{e}$-step, and $(\lambda x.x)\,(z\,\mathtt{I}) \to_{\mathtt{v}} x[x\backslash z\,\mathtt{I}]$ has only one $\mathtt{m}$-step. We write $t \not\to_{\mathtt{n}}$ (resp. $t \not\to_{\mathtt{v}}$), and call t an ***n-normal form*** (resp. ***v-normal form***), if t cannot be reduced by means of $\to_{\mathtt{n}}$ (resp. $\to_{\mathtt{v}}$). Both CBN and CBV are non-deterministic: $t \to_{\mathtt{n}} u$ and $t \to_{\mathtt{n}} s$ does not necessarily implies $u = s$. But both calculi enjoy confluence, notably because their rules are orthogonal [39, 38]. Moreover, in each calculus, it is easy to show that any two different reduction paths to normal form have the same number of multiplicative and exponential steps.

CBN is to be understood as *head* reduction [8], *i.e.* reduction does not take place in arguments of applications, while CBV corresponds to *open* CBV reduction [6, 2], *i.e.* reduction does not take place inside abstractions. The sets of $\mathtt{n}$/$\mathtt{v}$-normal forms can be alternatively characterised by the following grammars [12]:

$$\begin{array}{ll} \textbf{\textit{(CBN Neutral)}} & \mathsf{ne}_{\mathtt{n}} ::= x \in \mathcal{X} \mid \mathsf{ne}_{\mathtt{n}}\, t \\ \textbf{\textit{(CBN Normal)}} & \mathsf{no}_{\mathtt{n}} ::= \lambda x.\mathsf{no}_{\mathtt{n}} \mid \mathsf{ne}_{\mathtt{n}} \end{array} \qquad \begin{array}{ll} \textbf{\textit{(CBV Variable)}} & \mathsf{vr}_{\mathtt{v}} ::= x \in \mathcal{X} \mid \mathsf{vr}_{\mathtt{v}}[x\backslash \mathsf{ne}_{\mathtt{v}}] \\ \textbf{\textit{(CBV Neutral)}} & \mathsf{ne}_{\mathtt{v}} ::= \mathsf{vr}_{\mathtt{v}}\,\mathsf{no}_{\mathtt{v}} \mid \mathsf{ne}_{\mathtt{v}}\,\mathsf{no}_{\mathtt{v}} \mid \mathsf{ne}_{\mathtt{v}}[x\backslash \mathsf{ne}_{\mathtt{v}}] \\ \textbf{\textit{(CBV Normal)}} & \mathsf{no}_{\mathtt{v}} ::= \lambda x.t \mid \mathsf{vr}_{\mathtt{v}} \mid \mathsf{ne}_{\mathtt{v}} \mid \mathsf{no}_{\mathtt{v}}[x\backslash \mathsf{ne}_{\mathtt{v}}] \end{array}$$

In contrast to CBN, variables are left out of the definition of neutral terms for the CBV case, since they are now considered as values. However, even if CBV variables are not neutral terms, neutral terms are necessarily headed by a variable, so that variables play a double role which is difficult to be distinguished by means of an intersection type system. We will come back to this point in Sec. 4. Excluding variables from the set of values brings a remarkable speed up in implementations of CBV [40], but goes beyond the logical Girard's translation of CBV into LL, which is the main topic of this paper. Our chosen approach allows both CBN and CBV neutral terms to translate to neutral terms of the λ!-calculus (*cf.* Sec. 5).

Deterministic Strategies for CBN and CBV. As a technical tool, in order to count the reduction steps of CBN/CBV we first fix a deterministic version for them. The reduction relation $\to_{\mathtt{dn}}$ is a deterministic version of $\to_{\mathtt{n}}$ defined as:

$$\frac{}{\mathtt{L}\langle\lambda x.t\rangle\, u \to_{\mathtt{dn}} \mathtt{L}\langle t[x\backslash u]\rangle} \qquad \frac{}{t[x\backslash u] \to_{\mathtt{dn}} t\,\{x\backslash u\}} \qquad \frac{t \to_{\mathtt{dn}} s \text{ and } \neg\mathsf{abs}(t)}{t\,u \to_{\mathtt{dn}} s\,u} \qquad \frac{t \to_{\mathtt{dn}} s}{\lambda x.t \to_{\mathtt{dn}} \lambda x.u}$$

Similarly, the reduction relation $\to_{\mathtt{dv}}$ is a deterministic version of $\to_{\mathtt{v}}$ defined as:

$$\frac{}{\mathtt{L}\langle\lambda x.t\rangle\, u \to_{\mathtt{dv}} \mathtt{L}\langle t[x\backslash u]\rangle} \qquad \frac{}{t[x\backslash \mathtt{L}\langle v\rangle] \to_{\mathtt{dv}} \mathtt{L}\langle t\,\{x\backslash v\}\rangle} \qquad \frac{t \to_{\mathtt{dv}} s \quad \neg\mathsf{abs}(t)}{t\,u \to_{\mathtt{dv}} s\,u}$$

$$\frac{t \to_{\mathtt{dv}} s \quad u \in \mathsf{ne}_{\mathtt{v}} \cup \mathsf{vr}_{\mathtt{v}}}{u\,t \to_{\mathtt{dv}} u\,s} \qquad \frac{t \to_{\mathtt{dv}} s \quad \neg\mathsf{val}(t)}{u[x\backslash t] \to_{\mathtt{dv}} u[x\backslash s]} \qquad \frac{t \to_{\mathtt{dv}} s \quad u \in \mathsf{ne}_{\mathtt{v}}}{t[x\backslash u] \to_{\mathtt{dv}} s[x\backslash u]}$$

As a matter of notation, for $\mathcal{X} \in \{\mathtt{dn}, \mathtt{dv}\}$, we write $t \twoheadrightarrow_{\mathcal{X}}^{(m,e)} u$ if $t \twoheadrightarrow_{\mathcal{X}} u$ using m multiplicative steps and e exponential steps.

The normal forms of the non-deterministic and the deterministic versions of CBN/CBV are the same, in turn characterised by the grammars $\mathsf{no}_{\mathtt{n}}/\mathsf{no}_{\mathtt{v}}$ [12].

▶ **Proposition 1.** *Let $t \in \mathcal{T}_\lambda$. Then, $t \not\to_{\mathtt{n}}$ iff $t \not\to_{\mathtt{dn}}$ iff $t \in \mathsf{no}_{\mathtt{n}}$ and $t \not\to_{\mathtt{v}}$ iff $t \not\to_{\mathtt{dv}}$ iff $t \in \mathsf{no}_{\mathtt{v}}$.*

(Head) CBN ignores reduction inside arguments of applications, while (Open) CBV ignores reduction inside abstractions, then CBN (resp. CBV) normal forms are measured by the following ***n-size*** (resp. ***v-size***) function:

$$\begin{array}{llll} |x|_{\mathtt{n}} := 0 & |\lambda x.t|_{\mathtt{n}} := |t|_{\mathtt{n}} + 1 & |t\,u|_{\mathtt{n}} := |t|_{\mathtt{n}} + 1 & |t[x\backslash u]|_{\mathtt{n}} := |t|_{\mathtt{n}} \\ |x|_{\mathtt{v}} := 0 & |\lambda x.t|_{\mathtt{v}} := 0 & |t\,u|_{\mathtt{v}} := |t|_{\mathtt{v}} + |u|_{\mathtt{v}} + 1 & |t[x\backslash u]|_{\mathtt{v}} := |t|_{\mathtt{v}} + |u|_{\mathtt{v}} \end{array}$$

3 Tight Call-by-Name

We now introduce a tight type system $\mathcal{N}$ for CBN which captures independent exact measures for $\to_{\mathtt{n}}$-reduction sequences. This result is not surprising, since it extends the one in [1] from the pure λ-calculus to our CBN calculus. However, we revisit the op.cit. approach by appropriately splitting reduction into multiplicative and exponential steps, a reformulation necessary to establish a precise correspondence with the tight type system for the $\lambda!$-calculus. In particular, our *counting* mechanism slightly differs from [1] (details below).

In system $\mathcal{N}$ there are two base types: $\mathtt{a}$ types terms whose normal form is an abstraction, and $\mathtt{n}$ types terms whose normal form is CBN neutral. The grammar of types is given by:

$$\begin{array}{rrcl} \textbf{\textit{(Tight Types)}} & \mathtt{tt} & ::= & \mathtt{n} \mid \mathtt{a} \\ \textbf{\textit{(Types)}} & \sigma, \tau & ::= & \mathtt{tt} \mid \mathcal{M} \mid \mathcal{M} \to \sigma \\ \textbf{\textit{(Multitypes)}} & \mathcal{M}, \mathcal{N} & ::= & [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set} \end{array}$$

Multitypes are multisets of types. The ***empty multitype*** is denoted by $[\,]$, $\sqcup$ denotes multitype union, and $\sqsubseteq$ multitype inclusion. Also, $|\mathcal{M}|$ denotes the size of the multitype, thus if $\mathcal{M} = [\sigma_i]_{i \in I}$ then $|\mathcal{M}| = \#(I)$. Notice that the grammar for types slightly differs from [1], in particular types are now allowed to be just multitypes. The main reason to adopt this change is that this (unique) grammar is used for our three formalisms CBN, CBV, and CBPV, changing only the definition of tight types for each case[2].

Typing contexts (or just ***contexts***), written Γ, Δ, are functions from variables to multitypes, assigning the empty multitype to all but a finite set of variables. The domain of Γ is given by $\mathtt{dom}(\Gamma) \stackrel{def}{=} \{x \mid \Gamma(x) \neq [\,]\}$. The ***union of contexts***, written $\Gamma + \Delta$, is defined by $(\Gamma + \Delta)(x) \stackrel{def}{=} \Gamma(x) \sqcup \Delta(x)$. An example is $(x : [\sigma], y : [\tau]) + (x : [\sigma], z : [\tau]) = (x : [\sigma, \sigma], y : [\tau], z : [\tau])$. This notion is extended to several contexts as expected, so that $+_{i \in I}\, \Gamma_i$ denotes a finite union of contexts (particularly the empty context when $I = \emptyset$). We write $\Gamma \setminus\!\setminus x$ for the context $(\Gamma \setminus\!\setminus x)(x) = [\,]$ and $(\Gamma \setminus\!\setminus x)(y) = \Gamma(y)$ if $y \neq x$.

Type judgements have the form $\Gamma \vdash^{(m,e,s)} t : \sigma$, where Γ is a typing context, t is a term, σ is a type, and the ***counters*** (m, e, s) are expected to provide the following information: m (resp. e) indicates the number of multiplicative $\mathtt{m}$-steps (resp. exponential $\mathtt{e}$-steps) to normal form, while s indicates the $\mathtt{n}$-size of this normal form. It is worth noticing that the λ-calculus hides both multiplicative and exponential steps in one single β-reduction rule [1], so that only two counters suffice, one for the number of β-reduction steps, and another for the size of normal forms. Here we want to discriminate between multiplicative/exponential steps, in the sense that the execution of an ES generates an exponential step, but not a multiplicative one. This becomes possible due to the CBN/CBV alternative specifications with ES that we have adopted, and that is why we need three independent counters, in contrast to the λ-calculus.

The type system $\mathcal{N}$ for CBN is given in Fig. 1 (*persistent* rules) and 2 (*consuming* rules). A constructor is consuming (resp. persistent) if it is consumed (resp. not consumed) during $\mathtt{n}$-reduction. For instance, in $\mathtt{K}\,\mathtt{I}\,\Omega$ the two abstractions of $\mathtt{K}$ are consuming, while the abstraction of $\mathtt{I}$ is persistent, and all the other constructors are also consuming, except those of Ω that turns out to be an untyped subterm. The persistent rules (Fig. 1) are those typing persistent constructors, so that none of them increases the first two counters, but only possibly the third one, which contributes to the size of the normal form. The consuming rules (Fig. 2), in contrast, type consuming constructors, so that they may increase the first

[2] In the intersection type literature CBN and CBV do always adopt different grammars.

two counters, contributing to the length of the normalisation sequence. Notice in particular that there are only two rules contributing to the multiplicative/exponential counting:

(1) rule $(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$ types a *consuming* application, meaning that its left-hand side subterm reduces to an abstraction, then causing a multiplicative step (rule $\mathtt{dB}$) [3] followed later by an exponential step; while

(2) rule $(\mathtt{es}_{\mathtt{c}}^{\mathcal{N}})$ types a *consuming* substitution causing an exponential step. In both cases, it is the constructor application/substitution which is considered to be consumed, without any further hypothesis on the form of the subterm that appears inside this consuming constructor (recall that any term can be substituted in CBN). This phenomenon facilitates in particular the identification of exponential steps in CBN, in contrast to CBV and CBPV.

$$\frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{n}}{\Gamma \vdash^{(m,e,s+1)} t\, u : \mathtt{n}}\,(\mathtt{app}_{\mathtt{p}}^{\mathcal{N}}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{tt} \quad \mathsf{tight}(\Gamma(x))}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s+1)} \lambda x.t : \mathtt{a}}\,(\mathtt{abs}_{\mathtt{p}})$$

Figure 1 System $\mathcal{N}$ for the Call-by-Name Calculus: Persistent Typing Rules.

$$\frac{}{x : [\sigma] \vdash^{(0,0,0)} x : \sigma}\,(\mathtt{var}_{\mathtt{c}}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \tau}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s)} \lambda x.t : \Gamma(x) \to \tau}\,(\mathtt{abs}_{\mathtt{c}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : [\sigma_i]_{i\in I} \to \tau \quad (\Delta_i \vdash^{(m_i,e_i,s_i)} u : \sigma_i)_{i\in I}}{\Gamma + \Delta \vdash^{(1+m+_{i\in I} m_i,\, 1+e+_{i\in I} e_i,\, s+_{i\in I} s_i)} t\, u : \tau}\,(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$$

$$\frac{\Gamma; x : [\sigma_i]_{i\in I} \vdash^{(m,e,s)} t : \tau \quad (\Delta_i \vdash^{(m_i,e_i,s_i)} u : \sigma_i)_{i\in I}}{(\Gamma \setminus\!\setminus x) +_{i\in I} \Delta_i \vdash^{(m+_{i\in I} m_i,\, 1+e+_{i\in I} e_i,\, s+_{i\in I} s_i)} t[x\backslash u] : \tau}\,(\mathtt{es}_{\mathtt{c}}^{\mathcal{N}})$$

Figure 2 System $\mathcal{N}$ for the Call-by-Name Calculus: Consuming Typing Rules.

This dichotomy between consuming/persistent constructors has been first used in [36] for the λ and $\lambda\mu$-calculi, and adapted here for our distant versions of CBN/CBV as well as for the λ!-calculus. We write $\rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ if there is a ***(tree) type derivation*** of the judgement $\Gamma \vdash^{(m,e,s)} t : \sigma$ in system $\mathcal{N}$. The term t is typable in system $\mathcal{N}$, or $\mathcal{N}$-typable, iff there is a context Γ, a type σ and counters (m, e, s) such that $\rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$. We use the capital Greek letters $\Phi, \Psi, \ldots$ to name type derivations, by writing for example $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$. As (local) counters of judgements in a given derivation Φ contribute to the global counters of the derivation itself, there is an alternative way to define counters associated to a derivation Φ: the first counter m_Φ is given by the number of rules $(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$ in Φ, the second counter e_Φ is the number of rules $(\mathtt{es}_{\mathtt{c}}^{\mathcal{N}})$ in Φ and finally the third counter s_Φ is the number of rules $(\mathtt{app}_{\mathtt{p}}^{\mathcal{N}})$ and $(\mathtt{abs}_{\mathtt{p}})$ in Φ. We prefer however to systematically write counters in judgements to easy the understanding of the examples and proofs.

A ***multitype*** $[\sigma_i]_{i\in I}$ is ***tight***, written $\mathsf{tight}([\sigma_i]_{i\in I})$, if $\sigma_i \in \mathtt{tt}$ for all $i \in I$. A ***context*** Γ is said to be ***tight*** if it assigns tight multitypes to all variables. A ***type derivation*** $\Phi \rhd_{\mathcal{B}} \Gamma \vdash^{(m,e,s)} t : \sigma$ is ***tight*** if Γ is tight and $\sigma \in \mathtt{tt}$.

The proofs of soundness and completeness related to our CBN type system are respectively based on subject reduction and expansion properties, and they are very similar to those in [1]. The most important point to be mentioned is that system $\mathcal{N}$ is now counting *separately* the multiplicative and exponential steps of $\to_{\mathtt{n}}$-reductions to normal-form.

[3] In both [1] and [36], it is the consuming abstraction which contributes to the multiplicative steps.

▶ **Theorem 2** (Soundness and Completeness).

- *If* $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight, then there exists* p *such that* $p \in \mathsf{no}_{\mathtt{n}}$ *and* $t \twoheadrightarrow_{\mathtt{n}}^{(m,e)} p$ *with* m $\mathtt{m}$*-steps,* e $\mathtt{e}$*-steps, and* $|p|_{\mathtt{n}} = s$.
- *If* $t \twoheadrightarrow_{\mathtt{n}}^{(m,e)} p$ *with* $p \in \mathsf{no}_{\mathtt{n}}$*, then there exists a tight type derivation* $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,|p|_{\mathtt{n}})} t : \sigma$.

Notice that the previous theorem is stated by using the general notion of reduction $\to_{\mathtt{n}}$, but the proofs are done using the deterministic reduction $\to_{\mathtt{dn}}$. The equivalence holds because any two different reduction paths to normal form have the same number of multiplicative and exponential steps, as already remarked.

▶ **Example 3.** Consider $t_0 = \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})$ that $\to_{\mathtt{n}}$-reduces in 2 $\mathtt{m}$-steps and 2 $\mathtt{e}$-steps to the term $z\,\mathtt{I} \in \mathsf{no}_{\mathtt{n}}$, whose $\mathtt{n}$-size is 1:

$$t_0 = \underline{\mathtt{K}\,(z\,\mathtt{I})}\,(\mathtt{I}\,\mathtt{I}) \to_{\mathtt{dB}} \underline{(\lambda y.x)[x\backslash z\,\mathtt{I}]\,(\mathtt{I}\,\mathtt{I})} \to_{\mathtt{dB}} \underline{x[y\backslash \mathtt{I}\,\mathtt{I}][x\backslash z\,\mathtt{I}]} \to_{\mathtt{sn}} \underline{(z\,\mathtt{I})[y\backslash \mathtt{I}\,\mathtt{I}]} \to_{\mathtt{sn}} z\,\mathtt{I}$$

System $\mathcal{N}$ admits a tight type derivation for t_0 with the expected final counter $(2,2,1)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{n}] \vdash^{(0,0,0)} x : \mathtt{n}}\,(\mathtt{var_c})}{x : [\mathtt{n}] \vdash^{(0,0,0)} \lambda y.x : [\,] \to \mathtt{n}}\,(\mathtt{abs_c})}{\vdash^{(0,0,0)} \mathtt{K} : [\mathtt{n}] \to [\,] \to \mathtt{n}}\,(\mathtt{abs_c}) \quad \dfrac{\dfrac{}{z : [\mathtt{n}] \vdash^{(0,0,0)} z : \mathtt{n}}\,(\mathtt{var_c})}{z : [\mathtt{n}] \vdash^{(0,0,1)} z\,\mathtt{I} : \mathtt{n}}\,(\mathtt{app}_{\mathtt{p}}^{\mathcal{N}})}{z : [\mathtt{n}] \vdash^{(1,1,1)} \mathtt{K}\,(z\,\mathtt{I}) : [\,] \to \mathtt{n}}\,(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})}{z : [\mathtt{n}] \vdash^{(2,2,1)} \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I}) : \mathtt{n}}\,(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$$

4 Tight Call-by-Value

In this section we introduce a tight system $\mathcal{V}$ for CBV which captures exact measures for $\to_{\mathtt{v}}$-reduction sequences. Here we do not only distinguish length of reduction sequences from size of normal forms, but also discriminate multiplicative from exponential steps. Moreover, we establish a precise relation between the counters in CBV and their counterparts in the $\lambda!$-calculus (*cf.* Sec. 8). This constitutes one of the main contributions of the present work.

The grammar of types of system $\mathcal{V}$ is given by:

(Tight Types)	$\mathtt{tt}$	$::=$	$\mathtt{n} \mid \mathtt{vl} \mid \mathtt{vr}$
(Types)	σ, τ	$::=$	$\mathtt{tt} \mid \mathcal{M} \mid \mathcal{M} \to \sigma$
(Multitypes)	$\mathcal{M}, \mathcal{N}$	$::=$	$[\sigma_i]_{i \in I}$ where I is a finite set

Indeed, apart from the constant $\mathtt{n}$ also used in CBN, we now introduce constants $\mathtt{vl}$ and $\mathtt{vr}$, typing respectively, terms reducing to values; and terms reducing to persistent variables that are not acting as values (*i.e.* they are applied to some argument to produce neutral terms). Notice that we remove the base type $\mathtt{a}$ from CBN, since abstractions are in particular values, so they will be typed with $\mathtt{vl}$. The notions of ***tightness*** for multitypes, contexts and derivations are exactly the same we used for CBN.

A type system for CBV must type variables without knowing yet the future role they are going to play (head of neutral term or value). This is one of the main difficulties behind the definition of such a system. Besides that, the system must also be able to count multiplicative and exponential steps independently. The resulting type system $\mathcal{V}$ for CBV is given by the typing rules in Fig. 3 and 4, distinguishing between persistent and consuming rules resp. As a matter of notation, given a tight type $\mathtt{tt}_0$ we write $\overline{\mathtt{tt}_0}$ to denote a tight type different from $\mathtt{tt}_0$. Thus for instance, $\overline{\mathtt{vr}} \in \{\mathtt{vl}, \mathtt{n}\}$.

$$\frac{}{x : [\mathtt{vr}] \vdash^{(0,0,0)} x : \mathtt{vr}}\,(\mathtt{var_p}) \qquad \frac{}{\vdash^{(0,0,0)} x : \mathtt{vl}}\,(\mathtt{val_p}) \qquad \frac{}{\vdash^{(0,0,0)} \lambda x.t : \mathtt{vl}}\,(\mathtt{abs}^{\mathcal{V}}_{\mathtt{p}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \overline{\mathtt{vl}} \quad \Delta \vdash^{(m',e',s')} u : \overline{\mathtt{vr}}}{\Gamma + \Delta \vdash^{(m+m',e+e',s+s'+1)} t\,u : \mathtt{n}}\,(\mathtt{app}^{\mathcal{V}}_{\mathtt{p}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \tau \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\Gamma(x))}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x\backslash u] : \tau}\,(\mathtt{es_p})$$

Figure 3 System $\mathcal{V}$ for the Call-by-Value Calculus: Persistent Typing Rules.

$$\frac{}{x : \mathcal{M} \vdash^{(0,1,0)} x : \mathcal{M}}\,(\mathtt{var}^{\mathcal{V}}_{\mathtt{c}}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : [\mathcal{M} \to \tau] \quad \Delta \vdash^{(m',e',s')} u : \mathcal{M}}{\Gamma + \Delta \vdash^{(m+m'+1,e+e'-1,s+s')} t\,u : \tau}\,(\mathtt{app}^{\mathcal{V}}_{\mathtt{c}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : [\mathcal{M} \to \tau] \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\mathcal{M})}{\Gamma + \Delta \vdash^{(m+m'+1,e+e'-1,s+s')} t\,u : \tau}\,(\mathtt{appt}^{\mathcal{V}}_{\mathtt{c}})$$

$$\frac{(\Gamma_i \vdash^{(m_i,e_i,s_i)} t : \tau_i)_{i\in I}}{+_{i\in I}\,\Gamma_i \setminus\!\setminus x \vdash^{(+_{i\in I} m_i, 1+_{i\in I} e_i, +_{i\in I} s_i)} \lambda x.t : [\Gamma_i(x) \to \tau_i]_{i\in I}}\,(\mathtt{abs}^{\mathcal{V}}_{\mathtt{c}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \sigma \quad \Delta \vdash^{(m',e',s')} u : \Gamma(x)}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x\backslash u] : \sigma}\,(\mathtt{es_c})$$

Figure 4 System $\mathcal{V}$ for the Call-by-Value Calculus: Consuming Typing Rules.

Some rules deserve a comment. A difficult property to be statically captured by the counters is that an exponential step can only be generated by the meeting of a substitution constructor with an appropriate value argument. This remark leads to the introduction of different rules for typing variables, depending on the role they play (to be a value or not). Indeed, there are three axioms for variables: $(\mathtt{var_p})$ typing variables that will persist as the head of a neutral term; $(\mathtt{val_p})$ typing variable values that may be substituted by other values, *i.e.* they are *placeholders* for future persistent values; and $(\mathtt{var}^{\mathcal{V}}_{\mathtt{c}})$ typing variable values that, in particular, may be consumed as arguments. Consuming values are always typed with multitypes. Rule $(\mathtt{app}^{\mathcal{V}}_{\mathtt{p}})$ types neutral applications, *i.e.* the left premise has type $\mathtt{vr}$ or $\mathtt{n}$. Rule $(\mathtt{abs}^{\mathcal{V}}_{\mathtt{c}})$ increases the second counter, just like $(\mathtt{var}^{\mathcal{V}}_{\mathtt{c}})$, typing a value that is consumed as an argument. Rules $(\mathtt{app}^{\mathcal{V}}_{\mathtt{c}})$ and $(\mathtt{appt}^{\mathcal{V}}_{\mathtt{c}})$ increment the first counter because the (consuming) application will be used to perform a $\mathtt{dB}$-step, while they decrement the second counter to compensate for the left-hand-side value that is not being consumed after all. In other words, consuming values are systematically typed by incrementing their exponential counter by one (rules $(\mathtt{var}^{\mathcal{V}}_{\mathtt{c}})$ and $(\mathtt{abs}^{\mathcal{V}}_{\mathtt{c}})$), but they can finally act as computations instead as values, notably when they are placed in a head position, so that their exponential counter needs to be adjusted correctly (*cf.* Ex. 11). The decrement of the exponential counters in rules $(\mathtt{app}^{\mathcal{V}}_{\mathtt{c}})$ and $(\mathtt{appt}^{\mathcal{V}}_{\mathtt{c}})$ can also be understood by means of the subtle translation from CBV to the $\lambda!$-calculus that we introduce in Sec. 7. Rule $(\mathtt{appt}^{\mathcal{V}}_{\mathtt{c}})$ is particularly useful to type $\mathtt{dB}$-redexes whose reduction does not create an exponential redex, because the argument of the substitution created by the $\mathtt{dB}$-step does not reduce to a value.

In spite of the decrements in rules $(\mathtt{app}^{\mathcal{V}}_{\mathtt{c}})$ and $(\mathtt{appt}^{\mathcal{V}}_{\mathtt{c}})$, the counters are positive.

▶ **Lemma 4.** *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *then* $e \geq 0$*. Moreover, if* σ *is a multitype then* $e > 0$*.*

Soundness. The soundness property is based on a series of auxiliary results that enables to reason about tight type derivations, we only state here the more important ones.

▶ **Lemma 5** (Tight Spreading). *Let* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *such that* Γ *is tight.*
1. *If* $t \in \mathsf{ne}_{\mathtt{v}}$ *or* $m = e = 0$*, then* $\sigma \in \mathtt{tt}$*.*
2. *If* $\sigma = [\mathcal{M} \to \tau]$*, then* $t \notin \mathsf{vr}_{\mathtt{v}}$*.*

▶ **Lemma 6** (Exact Subject Reduction). *Let* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *be a tight derivation. If* $t \to_{\mathtt{dv}} t'$*, then there is* $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash^{(m',e',s)} t' : \sigma$ *such that*
(1) $m' = m - 1$ *and* $e' = e$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{m}$*-step;*
(2) $e' = e - 1$ *and* $m' = m$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{e}$*-step.*

An interesting remark is that types of subterms in tight derivations may change during CBV reduction, unlike other approaches [1, 40]. To illustrate this phenomenon, consider the following reduction $x[x\backslash \mathtt{I}] \to_{\mathtt{dv}} \mathtt{I}$. The terms on the left and right hand side can be resp. tightly typed by the following derivations (we omit the counters):

$$\dfrac{\dfrac{}{\vdash x : \mathtt{vl}}\,(\mathtt{val_p}) \qquad \dfrac{}{\vdash \mathtt{I} : [\,]}\,(\mathtt{abs}^{\mathcal{V}}_{\mathtt{c}})}{\vdash x[x\backslash \mathtt{I}] : \mathtt{vl}}\,(\mathtt{es_c}) \qquad\qquad \dfrac{}{\vdash \mathtt{I} : \mathtt{vl}}\,(\mathtt{abs}^{\mathcal{V}}_{\mathtt{p}})$$

Notice that the identity function $\mathtt{I}$ is typed differently in each derivation: the substitution introduced by rule $(\mathtt{es_c})$ is a consuming constructor, which disappears when $\mathtt{sv}$-reduction consumes its value argument, a phenomenon that is captured by means of a multitype for the argument of the substitution. Indeed, $\mathtt{I}$ on the left-hand side derivation is typed with the multitype $[\,]$. This value $\mathtt{I}$ substitutes a variable x typed with $\mathtt{vl}$, which is just a placeholder for a persistent value. Thus, once the substitution is performed, the identity $\mathtt{I}$ becomes persistent on the right-hand side, a phenomenon which is naturally captured by the tight type $\mathtt{vl}$. This observation also applies to the tight typing system of CBPV in Sec. 6.

▶ **Theorem 7** (Soundness). *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight, then there exists* p *such that* $p \in \mathsf{no}_{\mathtt{v}}$ *and* $t \twoheadrightarrow_{\mathtt{v}}^{(m,e)} p$ *with* m $\mathtt{m}$*-steps,* e $\mathtt{e}$*-steps, and* $|p|_{\mathtt{v}} = s$*.*

As in CBN, the previous theorem is stated by using the general notion of reduction $\to_{\mathtt{v}}$, but the proofs (notably Lem. 6) are done using the deterministic reduction $\to_{\mathtt{dv}}$. Similar comment applies to the forthcoming Thm. 10.

Completeness. The completeness result is also based on intermediate lemmas, we only state here the so-called *tight typing of normal forms* and *subject expansion* properties.

▶ **Lemma 8** (Tight Typing of Normal Forms). *If* $t \in \mathsf{no}_{\mathtt{v}}$*, then there is a tight derivation* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(0,0,|t|_{\mathtt{v}})} t : \sigma$*.*

▶ **Lemma 9** (Exact Subject Expansion). *Let* $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash^{(m',e',s)} t' : \sigma$ *be a tight derivation. If* $t \to_{\mathtt{dv}} t'$*, then there is* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *such that*
(1) $m' = m - 1$ *and* $e' = e$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{m}$*-step;*
(2) $e' = e - 1$ *and* $m' = m$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{e}$*-step.*

Notice that tight derivations properly counts, separately, $\mathtt{m}$-steps and $\mathtt{e}$-steps. As a consequence, completeness follows.

▶ **Theorem 10** (Completeness). *If* $t \twoheadrightarrow_{\mathsf{v}}^{(m,e)} p$ *with* $p \in \mathsf{no}_{\mathsf{v}}$*, then there exists a tight type derivation* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,|p|_{\mathsf{v}})} t : \sigma$*.*

▶ **Example 11.** Consider $t_0 = \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})$ from Ex. 3 but now in the CBV setting. It $\to_{\mathsf{v}}$-reduces in 3 $\mathtt{m}$-steps and 2 e-steps to $x[x\backslash z\,\mathtt{I}] \in \mathsf{no}_{\mathsf{v}}$, whose v-size is 1, as follows:

$$\begin{array}{lllllll} t_0 & = & \underline{\mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})} & \to_{\mathtt{dB}} & \underline{(\lambda y.x)[x\backslash z\,\mathtt{I}]\,(\mathtt{I}\,\mathtt{I})} & \to_{\mathtt{dB}} & x[y\backslash \underline{\mathtt{I}\,\mathtt{I}}][x\backslash z\,\mathtt{I}] \\ & \to_{\mathtt{dB}} & \underline{x[y\backslash w[w\backslash \mathtt{I}]][x\backslash z\,\mathtt{I}]} & \to_{\mathtt{sv}} & \underline{x[w\backslash \mathtt{I}][x\backslash z\,\mathtt{I}]} & \to_{\mathtt{sv}} & x[x\backslash z\,\mathtt{I}] \end{array}$$

The reader may check that system $\mathcal{V}$ indeed admits a proper tight type derivation for t_0 with final counters $(3, 2, 1)$, as expected.

5 The λ!-Calculus

This section briefly presents the ***bang calculus at a distance*** [12], called λ!-calculus. It is a (conservative) extension of the original bang calculus [25, 26], it uses ES operators and *reduction at a distance* [5], thus integrating commutative conversions without jeopardising confluence (see [12] for a discussion). Indeed, T. Ehrhard [25] studies the CBPV from a Linear Logic (LL) point of view by extending the λ-calculus with two new unary constructors *bang* (!) and *dereliction* (der), playing the role of the CBPV primitives `thunk`/`force` respectively. His calculus suffers from the absence of *commutative conversions* [45, 15], making some redexes to be syntactically blocked when open terms are considered. As a consequence, some normal forms are semantically equivalent to non-terminating programs, a situation which is clearly unsound. The bang calculus [26] adds commutative conversions specified by means of σ-reduction rules, which are crucial to unveil hidden (blocked) redexes. This approach, however, presents a major drawback since the resulting combined reduction relation is not confluent. The λ!-calculus [12] fixes these two problems at the same time. Indeed, the syntax of the bang calculus is enriched with explicit substitutions (ES), and σ-equivalence is integrated in the primary reduction system by using the *distance* paradigm [5], without any need to unveil hidden redexes by means of an independent relation.

We consider the following grammar for terms (denoted by $\mathcal{T}$) and contexts:

$$\begin{array}{rrcl} \textbf{\textit{(Terms)}} & t, u & ::= & x \in \mathcal{X} \mid t\,u \mid \lambda x.t \mid !\,t \mid \mathrm{der}\,t \mid t[x\backslash u] \\ \textbf{\textit{(List contexts)}} & \mathtt{L} & ::= & \Box \mid \mathtt{L}[x\backslash t] \\ \textbf{\textit{(Surface contexts)}} & \mathtt{S} & ::= & \Box \mid \mathtt{S}\,t \mid t\,\mathtt{S} \mid \lambda x.\mathtt{S} \mid \mathrm{der}\,\mathtt{S} \mid \mathtt{S}[x\backslash u] \mid t[x\backslash \mathtt{S}] \end{array}$$

Special terms are $\Delta_! = \lambda x.x\,!\,x$, and $\Omega_! = \Delta_!\,!\,\Delta_!$. Surface contexts do not allow the symbol $\Box$ to occur inside the bang constructor $!$. This is similar to *weak* contexts in λ-calculus, where $\Box$ cannot occur inside λ-abstractions. As we will see in Sec. 7, surface reduction in the λ!-calculus is perfectly sufficient to capture head reduction in CBN, disallowing reduction inside arguments, as well as open CBV, disallowing reduction inside abstractions. Finally, we define the ***f-size*** of terms as follows:

$$\begin{array}{llllllllll} |x|_{\mathtt{f}} & := & 0 & |\mathrm{der}\,t|_{\mathtt{f}} & := & |t|_{\mathtt{f}} & |t[x\backslash u]|_{\mathtt{f}} & := & |t|_{\mathtt{f}} + |u|_{\mathtt{f}} \\ |!\,t|_{\mathtt{f}} & := & 0 & |\lambda x.t|_{\mathtt{f}} & := & 1 + |t|_{\mathtt{f}} & |t\,u|_{\mathtt{f}} & := & 1 + |t|_{\mathtt{f}} + |u|_{\mathtt{f}} \end{array}$$

The ***λ!-calculus*** is given by the set of terms $\mathcal{T}$ and the ***(surface) reduction relation*** $\to_{\mathtt{f}}$, which is defined as the *union* of $\to_{\mathtt{dB}}$, $\to_{\mathtt{s!}}$ (**s**ubstitute bang) and $\to_{\mathtt{d!}}$ (**d**istant bang), defined respectively as the closure by contexts $\mathtt{S}$ of the following three rewriting rules:

$$\begin{array}{lll} \mathtt{L}\langle \lambda x.t\rangle\,u & \mapsto_{\mathtt{dB}} & \mathtt{L}\langle t[x\backslash u]\rangle \\ t[x\backslash \mathtt{L}\langle !\,u\rangle] & \mapsto_{\mathtt{s!}} & \mathtt{L}\langle t\,\{x\backslash u\}\rangle \\ \mathrm{der}\,(\mathtt{L}\langle !\,t\rangle) & \mapsto_{\mathtt{d!}} & \mathtt{L}\langle t\rangle \end{array}$$

The rules are defined *at a distance*, as in CBN/CBV, in the sense that the list context $\mathtt{L}$ allows the main constructors involved in the rules to be separated by an arbitrary finite list of substitutions. This new formulation integrates commutative conversions inside the main (logical) reduction rules of the calculus, thus inheriting the benefits enumerated in Sec. 1. Indeed, rule $\mathtt{s}!$ can be decomposed in two different rules $t[x\backslash !\, u] \mapsto t\,\{x\backslash u\}$ and $t[x\backslash \mathtt{L}\langle !\, u\rangle] \mapsto \mathtt{L}\langle t[x\backslash !\, u]\rangle$, while $\mathtt{d}!$ can be decomposed in $\mathrm{der}\,(!\, t) \mapsto t$ and $\mathrm{der}\,(\mathtt{L}\langle !\, t\rangle) \mapsto \mathtt{L}\langle \mathrm{der}\,!\, t\rangle$. We write $\twoheadrightarrow_{\mathtt{f}}$ for the reflexive-transitive closure of $\rightarrow_{\mathtt{f}}$. Given the translation of the bang calculus into LL proof-nets [25], we refer to $\mathtt{dB}$-steps as $\mathtt{m}$***ultiplicative*** and $\mathtt{s}!/\mathtt{d}!$-steps as $\mathtt{e}$***xponential*** steps. We write $t \twoheadrightarrow_{\mathtt{f}}^{(m,e)} u$ if $t \twoheadrightarrow_{\mathtt{f}} u$ using m $\mathtt{m}$-steps and e $\mathtt{e}$-steps.

▶ **Example 12.** Consider the following reduction sequence from $t'_0 = \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I}))$:

$$t'_0 = \underline{\mathtt{K}\,(!\,(z\,!\,\mathtt{I}))}\,(!\,(\mathtt{I}\,!\,\mathtt{I})) \quad \underset{\underset{\rightarrow_{\mathtt{s}!}}{}}{\rightarrow_{\mathtt{dB}}} \quad \underline{(\lambda y.x)[x\backslash !\,(z\,!\,\mathtt{I})]\,(!\,(\mathtt{I}\,!\,\mathtt{I}))} \quad \underset{\rightarrow_{\mathtt{s}!}}{\rightarrow_{\mathtt{dB}}} \quad \underline{x[y\backslash !\,(\mathtt{I}\,!\,\mathtt{I})][x\backslash !\,(z\,!\,\mathtt{I})]}$$
$$\underline{(z\,!\,\mathtt{I})[y\backslash !\,(\mathtt{I}\,!\,\mathtt{I})]} \qquad\qquad z\,!\,\mathtt{I}$$

Notice that the second $\mathtt{dB}$-step uses action at a distance, where $\mathtt{L}$ is $\Box[x\backslash !\,(z\,!\,\mathtt{I})]$.

The relation $\rightarrow_{\mathtt{f}}$ enjoys a weak diamond property, *i.e.* one-step divergence can be closed in one step if the diverging terms are different. This property has two important consequences.

▶ **Theorem 13** (Confluence [12])**.** *The reduction relation $\rightarrow_{\mathtt{f}}$ is confluent. Moreover, any two different $\rightarrow_{\mathtt{f}}$-reduction paths to normal form have the same length.*

The second point relies essentially on the fact that reductions are disallowed under bangs. An important consequence is that we can focus on any particular *deterministic* strategy for the $\lambda!$-calculus, without changing the number of steps to $\mathtt{f}$-normal form.

Neutral, Normal, and Clash-Free Terms. A term is said to be $\mathtt{f}$***-normal*** if there is no t' such that $t \rightarrow_{\mathtt{f}} t'$, in which case we write $t \not\rightarrow_{\mathtt{f}}$. However, some ill-formed $\mathtt{f}$-normal terms are not still the ones that represent a desired result for a computation, they are called ***clashes*** (meta-variable c), and take one of the following forms: $\mathtt{L}\langle !\, t\rangle\, u$, $t[y\backslash \mathtt{L}\langle \lambda x.u\rangle]$, $\mathrm{der}\,(\mathtt{L}\langle \lambda x.u\rangle)$, or $t\,(\mathtt{L}\langle \lambda x.u\rangle)$. Remark that in the three first kind of clashes, replacing $\lambda x.$ by !, and inversely, creates a (root) redex, namely $(\mathtt{L}\langle \lambda x.t\rangle)\, u$, $t[x\backslash \mathtt{L}\langle !\, t\rangle]$ and $\mathrm{der}\,(\mathtt{L}\langle !\, t\rangle)$, respectively.

A term is ***clash free*** if it does not reduce to a term containing a clash, it is ***surface clash free***, written $\mathtt{scf}$, if it does not reduce to a term containing a clash outside the scope of any constructor !. Thus, t is not $\mathtt{scf}$ if and only if there exist a surface context $\mathtt{S}$ and a clash c such that $t \twoheadrightarrow_{\mathtt{f}} \mathtt{S}\langle c\rangle$. Surface clash free normal terms can be characterised as follows:

$$\begin{array}{rlcl}
\textit{(Neutral } \mathtt{scf}\textit{)} & \mathsf{ne}_{\mathtt{scf}} & ::= & x \in \mathcal{X} \mid \mathsf{ne}_{\mathtt{scf}}\,\mathsf{na}_{\mathtt{scf}} \mid \mathrm{der}\,(\mathsf{ne}_{\mathtt{scf}}) \mid \mathsf{ne}_{\mathtt{scf}}[x\backslash \mathsf{ne}_{\mathtt{scf}}] \\
\textit{(Neutral-Abs } \mathtt{scf}\textit{)} & \mathsf{na}_{\mathtt{scf}} & ::= & !\, t \mid \mathsf{ne}_{\mathtt{scf}} \mid \mathsf{na}_{\mathtt{scf}}[x\backslash \mathsf{ne}_{\mathtt{scf}}] \\
\textit{(Neutral-Bang } \mathtt{scf}\textit{)} & \mathsf{nb}_{\mathtt{scf}} & ::= & \mathsf{ne}_{\mathtt{scf}} \mid \lambda x.\mathsf{no}_{\mathtt{scf}} \mid \mathsf{nb}_{\mathtt{scf}}[x\backslash \mathsf{ne}_{\mathtt{scf}}] \\
\textit{(Normal } \mathtt{scf}\textit{)} & \mathsf{no}_{\mathtt{scf}} & ::= & \mathsf{na}_{\mathtt{scf}} \mid \mathsf{nb}_{\mathtt{scf}}
\end{array}$$

▶ **Proposition 14** (Clash-Free Normal Terms [12])**.** *Let $t \in \mathcal{T}$. Then t is a surface clash free $\mathtt{f}$-normal term iff $t \in \mathsf{no}_{\mathtt{scf}}$.*

6 A Tight Type System for the λ!-Calculus

The methodology used to define the type system $\mathcal{B}$ for the $\lambda!$-calculus is based on [12], inspired in turn from [19, 9, 1], which defines non-idempotent intersection type systems to count reduction lengths for different evaluation strategies in the λ-calculus. In the case of the

λ!-calculus, however, Thm. 13 guarantees that all reduction paths to normal form have the same length, so that it is not necessary to reason w.r.t. any particular evaluation strategy.

The grammar of types of system $\mathcal{B}$ is given by:

$$\begin{array}{rrcl} \textbf{\textit{(Tight Types)}} & \mathtt{tt} & ::= & \mathtt{n} \mid \mathtt{a} \mid \mathtt{vl} \\ \textbf{\textit{(Types)}} & \sigma, \tau & ::= & \mathtt{tt} \mid \mathcal{M} \mid \mathcal{M} \to \sigma \\ \textbf{\textit{(Multitype)}} & \mathcal{M}, \mathcal{N} & ::= & [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set} \end{array}$$

The constant $\mathtt{a}$ (resp. $\mathtt{vl}$) types terms whose normal form has the shape $\mathtt{L}\langle \lambda x.t \rangle$ (resp. $\mathtt{L}\langle !\, t \rangle$, *i.e. values* in the λ!-calculus sense), and the constant $\mathtt{n}$ types terms whose normal form is a neutral $\mathtt{scf}$. As before, the notions of ***tightness*** for multitypes, contexts and derivations are exactly the same used for CBN. Typing rules are split in two groups: the *persistent* rules (Fig. 5) and the *consuming* ones (Fig. 6).

$$\frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{n} \quad \Delta \vdash^{(m',e',s')} u : \overline{\mathtt{a}}}{\Gamma + \Delta \vdash^{(m+m',e+e',s+s'+1)} t\, u : \mathtt{n}}\,(\mathtt{app_p}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{tt} \quad \mathsf{tight}(\Gamma(x))}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s+1)} \lambda x.t : \mathtt{a}}\,(\mathtt{abs_p})$$

$$\frac{}{\vdash^{(0,0,0)} !\, t : \mathtt{vl}}\,(\mathtt{bg_p}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{n}}{\Gamma \vdash^{(m,e,s)} \mathrm{der}\, t : \mathtt{n}}\,(\mathtt{dr_p}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \tau \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\Gamma(x))}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x \backslash u] : \tau}\,(\mathtt{es_p})$$

Figure 5 System $\mathcal{B}$ for the λ!-Calculus: Persistent Typing Rules.

$$\frac{}{x : [\sigma] \vdash^{(0,0,0)} x : \sigma}\,(\mathtt{var_c}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathcal{M} \to \tau \quad \Delta \vdash^{(m',e',s')} u : \mathcal{M}}{\Gamma + \Delta \vdash^{(m+m'+1,e+e',s+s')} t\, u : \tau}\,(\mathtt{app_c})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \mathcal{M} \to \tau \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\mathcal{M})}{\Gamma + \Delta \vdash^{(m+m'+1,e+e',s+s')} t\, u : \tau}\,(\mathtt{appt_c})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \tau}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s)} \lambda x.t : \Gamma(x) \to \tau}\,(\mathtt{abs_c}) \qquad \frac{(\Gamma_i \vdash^{(m_i,e_i,s_i)} t : \sigma_i)_{i \in I}}{+_{i \in I}\, \Gamma_i \vdash^{(+_{i \in I} m_i, 1 +_{i \in I} e_i, +_{i \in I} s_i)} !\, t : [\sigma_i]_{i \in I}}\,(\mathtt{bg_c})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : [\sigma]}{\Gamma \vdash^{(m,e,s)} \mathrm{der}\, t : \sigma}\,(\mathtt{dr_c}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \sigma \quad \Delta \vdash^{(m',e',s')} u : \Gamma(x)}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x \backslash u] : \sigma}\,(\mathtt{es_c})$$

Figure 6 System $\mathcal{B}$ for the λ!-Calculus: Consuming Typing Rules.

As in CBV, the $\mathtt{s}!$ exponential steps do not only depend on a (consuming) substitution constructor, but on the (bang) form of its argument. This makes the exponential counting more subtle. Notice also that rule $(\mathtt{dr_p})$ does not count der constructors, according to the definition of $|\_|_{\mathtt{f}}$ given in Sec. 5 and in contrast to [12]. This is to keep a more intuitive relation with the CBN/CBV translations (Sec. 2), where der plays a silent role.

As in [12], system $\mathcal{B}$ is quantitatively sound and complete. More precisely,

▶ **Theorem 15** (Soundness and Completeness).

1. *If* $\Phi \rhd_{\mathcal{B}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight, then there exists* p *such that* $p \in \mathsf{no}_{\mathtt{scf}}$ *and* $t \twoheadrightarrow_{\mathtt{f}}^{(m,e)} p$ *with* m $\mathtt{m}$*-steps,* e $\mathtt{e}$*-steps, and* $|p|_{\mathtt{f}} = s$.
2. *If* $t \twoheadrightarrow_{\mathtt{f}}^{(m,e)} p$ *with* $p \in \mathsf{no}_{\mathtt{scf}}$*, then there exists a tight type derivation* $\Phi \rhd_{\mathcal{B}} \Gamma \vdash^{(m,e,|p|_{\mathtt{f}})} t : \sigma$.

▶ **Example 16.** Consider $t'_0 = \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I}))$ from Ex. 12, which normalises in 2 $\mathtt{m}$-steps and 2 $\mathtt{e}$-steps to $z\,!\,\mathtt{I} \in \mathtt{no}_{\mathtt{scf}}$ of $\mathtt{f}$-size 1. A tight derivation for t'_0 with appropriate final counters $(2,2,1)$ is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{n}] \vdash^{(0,0,0)} x : \mathtt{n}}\,(\mathtt{var_c})}{x : [\mathtt{n}] \vdash^{(0,0,0)} \lambda y.x : [\,] \to \mathtt{n}}\,(\mathtt{abs_c})}{\vdash^{(0,0,0)} \mathtt{K} : [\mathtt{n}] \to [\,] \to \mathtt{n}}\,(\mathtt{abs_c}) \qquad \dfrac{\dfrac{\dfrac{}{z : [\mathtt{n}] \vdash^{(0,0,0)} z : \mathtt{n}}\,(\mathtt{var_c}) \quad \dfrac{}{\vdash^{(0,0,0)} !\,\mathtt{I} : \mathtt{vl}}\,(\mathtt{bg_p})}{z : [\mathtt{n}] \vdash^{(0,0,1)} z\,!\,\mathtt{I} : \mathtt{n}}\,(\mathtt{app_p})}{z : [\mathtt{n}] \vdash^{(0,1,1)} !\,(z\,!\,\mathtt{I}) : [\mathtt{n}]}\,(\mathtt{bg_c})}{z : [\mathtt{n}] \vdash^{(1,1,1)} \mathtt{K}\,(!\,(z\,!\,\mathtt{I})) : [\,] \to \mathtt{n}}\,(\mathtt{app_c}) \qquad \dfrac{}{\vdash^{(0,1,0)} !\,(\mathtt{I}\,!\,\mathtt{I}) : [\,]}\,(\mathtt{bg_c})}{z : [\mathtt{n}] \vdash^{(2,2,1)} \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I})) : \mathtt{n}}\,(\mathtt{app_c})$$

Notice that the only persistent rules are $(\mathtt{bg_p})$ and $(\mathtt{app_p})$, used to type $z\,!\,\mathtt{I}$. Indeed, $z\,!\,\mathtt{I}$ is the $\mathtt{f}$-normal form of t'_0.

7 Untyped Translations

CBN/CBV (untyped) encodings into the bang calculus [32], inspired from Girard's encodings, establish two translations *cbn* and *cbv*, such that when t reduces to u in CBN (resp. CBV), $\mathit{cbn}(t)$ reduces to $\mathit{cbn}(u)$ (resp. $\mathit{cbv}(t)$ reduces to $\mathit{cbv}(u)$) in the bang calculus. These two encodings are dual: CBN forbids reduction inside arguments, which are translated to bang terms, while CBV forbids reduction under λ-abstractions, also translated to bang terms.

In this paper we use alternative encodings. For CBN, we slightly adapt to explicit substitutions Girard's translation into LL [29]. The resulting encoding preserves normal forms and is sound and complete with respect to the standard (quantitative) type system in [28]. For CBV, we discard the original encoding in [32] for two reasons: CBV normal forms are not necessarily translated to normal forms in the bang calculus (see [32]), and levels of terms (the level of t is the number of ! surrounding t) are not preserved either (see [27]). We thus adopt the CBV encoding in [12] which preserves normal forms as well as levels.

The CBN and CBV embedding into the λ!-calculus, written $\_^{\mathtt{n}}$ and $\_^{\mathtt{v}}$ resp., are inductively defined as:

$$\begin{array}{rcl} x^{\mathtt{n}} & \stackrel{def}{=} & x \\ (\lambda x.t)^{\mathtt{n}} & \stackrel{def}{=} & \lambda x.t^{\mathtt{n}} \\ (t\,u)^{\mathtt{n}} & \stackrel{def}{=} & t^{\mathtt{n}}\,!\,u^{\mathtt{n}} \\ (t[x\backslash u])^{\mathtt{n}} & \stackrel{def}{=} & t^{\mathtt{n}}[x\backslash !\,u^{\mathtt{n}}] \end{array} \qquad \begin{array}{rcl} x^{\mathtt{v}} & \stackrel{def}{=} & !\,x \\ (\lambda x.t)^{\mathtt{v}} & \stackrel{def}{=} & !\,\lambda x.t^{\mathtt{v}} \\ (t\,u)^{\mathtt{v}} & \stackrel{def}{=} & \begin{cases} \mathtt{L}\langle s\rangle\,u^{\mathtt{v}} & \text{if } t^{\mathtt{v}} = \mathtt{L}\langle !\,s\rangle \\ \mathrm{der}\,(t^{\mathtt{v}})\,u^{\mathtt{v}} & \text{otherwise} \end{cases} \\ (t[x\backslash u])^{\mathtt{v}} & \stackrel{def}{=} & t^{\mathtt{v}}[x\backslash u^{\mathtt{v}}] \end{array}$$

Both translations extend to list contexts $\mathtt{L}$ as expected. Remark that there are no two consecutive ! constructors in the image of the translation. The CBN embedding extends Girard's translation to ES, while the CBV one is different. Indeed, the translation of an application $t\,u$ is usually defined as $\mathrm{der}\,(t^{\mathtt{v}})\,u^{\mathtt{v}}$ (see *e.g.* [26]). This definition does not preserve normal forms, *i.e.* $x\,y$ is a $\mathtt{v}$-normal form but its translated version $\mathrm{der}\,(!\,x)\,!\,y$ is not a $\mathtt{f}$-normal form. We restore this fundamental property by using the well-known notion of superdevelopment [10], so that $\mathtt{d}$!-reductions are applied by the translation on the fly. Moreover, simulation of CBN/CBV in the λ!-calculus also holds.

▶ **Lemma 17** (Simulation [12])**.** *Let $t \in \mathcal{T}_\lambda$.*

1. *$t \not\to_{\mathtt{n}}$ implies $t^{\mathtt{n}} \not\to_{\mathtt{f}}$, and $t \to_{\mathtt{n}} s$ implies $t^{\mathtt{n}} \to_{\mathtt{f}} s^{\mathtt{n}}$.*
2. *$t \not\to_{\mathtt{v}}$ implies $t^{\mathtt{v}} \not\to_{\mathtt{f}}$, and $t \to_{\mathtt{v}} s$ implies $t^{\mathtt{v}} \twoheadrightarrow_{\mathtt{f}} s^{\mathtt{v}}$.*

▶ **Example 18.** Consider again $t_0 = \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})$. To illustrate the CBN case (Lem. 17:1), notice that the reduction sequence $t_0 \twoheadrightarrow_{\mathtt{cbn}} z\,\mathtt{I} = s_0$ given in Ex. 3 is translated to the sequence ${t_0}^{\mathtt{n}} = \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I})) = t'_0 \twoheadrightarrow_{\mathtt{f}} z\,!\,\mathtt{I} = {s_0}^{\mathtt{n}}$ given in Ex. 12.

To illustrate the CBV case (Lem. 17:2), consider the sequence $t_0 \twoheadrightarrow_{\mathtt{cbv}} x[x\backslash z\,\mathtt{I}] = s_1$ in Ex. 11. Then for $\mathtt{I}' = \lambda w.!\,w$ we have:

$$\begin{array}{lll}
{t_0}^{\mathtt{v}} = & \mathtt{der}\,(\underline{(\lambda x.!\,\lambda y.!\,x)\,(z\,!\,\mathtt{I}')})\,(\mathtt{I}'\,!\,\mathtt{I}') & \to_{\mathtt{dB}} \underline{\mathtt{der}\,((!\,\lambda y.!\,x)[x\backslash z\,!\,\mathtt{I}'])\,(\mathtt{I}'\,!\,\mathtt{I}')} \\
\to_{\mathtt{d!}} \underline{(\lambda y.!\,x)[x\backslash z\,!\,\mathtt{I}']\,(\mathtt{I}'\,!\,\mathtt{I}')} & & \to_{\mathtt{dB}} (!\,x)[y\backslash\underline{\mathtt{I}'\,!\,\mathtt{I}'}][x\backslash z\,!\,\mathtt{I}'] \\
\to_{\mathtt{dB}} \underline{(!\,x)[y\backslash(!\,w)[w\backslash!\,\mathtt{I}']]}[x\backslash z\,!\,\mathtt{I}'] & & \to_{\mathtt{s!}} \underline{(!\,x)[w\backslash!\,\mathtt{I}']}[x\backslash z\,!\,\mathtt{I}'] \\
\to_{\mathtt{s!}} (!\,x)[x\backslash z\,!\,\mathtt{I}'] & & = {s_1}^{\mathtt{v}}
\end{array}$$

Notice how this sequence requires extra reduction steps with respect to the one given in Ex. 11. Indeed, the $\mathtt{e}$-step $\to_{\mathtt{d!}}$ in the $\lambda!$-calculus has no counterpart in CBV.

8 Typed Translations

Call-by-Name. We study the correspondence between derivations in CBN and their encodings in the $\lambda!$-calculus. First we inject the set of types for $\mathcal{N}$ (generated by the base types $\mathtt{n}$ and $\mathtt{a}$) into the set of types of $\mathcal{B}$ (generated also by the base type $\mathtt{vl}$) by means of the function: $\mathtt{n}^{\mathtt{n}} \stackrel{def}{=} \mathtt{n}$, $\mathtt{a}^{\mathtt{n}} \stackrel{def}{=} \mathtt{a}$, $(\mathcal{M} \to \sigma)^{\mathtt{n}} \stackrel{def}{=} \mathcal{M}^{\mathtt{n}} \to \sigma^{\mathtt{n}}$ and $[\sigma_i]_{i\in I}{}^{\mathtt{n}} \stackrel{def}{=} [\sigma_i{}^{\mathtt{n}}]_{i\in I}$. Then we translate terms, using the function $\_^{\mathtt{n}}$ from Sec. 7. Translation of contexts is defined as expected: $\Gamma^{\mathtt{n}} = \{x_i : \mathcal{M}_i{}^{\mathtt{n}}\}_{i\in I}$. Another notion is needed to restrict $\mathcal{B}$ derivations to those that come from the translation of some $\mathcal{N}$ derivation. Indeed, a $\mathcal{B}$ derivation Φ is ***$\mathtt{n}$-relevant*** if all the contexts and types involved in Φ are in the image of the translation $\_^{\mathtt{n}}$. We then obtain:

▶ **Theorem 19.** *$\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ if and only if $\Phi' \rhd_{\mathcal{B}} \Gamma^{\mathtt{n}} \vdash^{(m,e,s)} t^{\mathtt{n}} : \sigma^{\mathtt{n}}$ is $\mathtt{n}$-relevant.*

This result is illustrated by the tight type derivations in Ex. 3 and 16 for the terms t_0 and t'_0 resp. Moreover, tightness is preserved by the translation of contexts and types, hence:

▶ **Corollary 20.** *If $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ is tight, then there exists $p \in \mathsf{no}_{\mathtt{scf}}$ such that $t^{\mathtt{n}} \twoheadrightarrow_{\mathtt{f}}^{(m,e)} p$ with m $\mathtt{m}$-steps, e $\mathtt{e}$-steps, and $|p|_{\mathtt{f}} = s$. Conversely, if $\Phi' \rhd_{\mathcal{B}} \Gamma^{\mathtt{n}} \vdash^{(m,e,s)} t^{\mathtt{n}} : \sigma^{\mathtt{n}}$ is tight and $\mathtt{n}$-relevant, then there exists $p \in \mathsf{no}_{\mathtt{n}}$ such that $t \twoheadrightarrow_{\mathtt{n}}^{(m,e)} p$ with m $\mathtt{m}$-steps, e $\mathtt{e}$-steps, and $|p|_{\mathtt{n}} = s$.*

This result shows that not only from the tight type system $\mathcal{N}$ it is possible to extract exact measures for the image of the CBN in the $\lambda!$-calculus, but more interestingly, also that from the tight type system $\mathcal{B}$ for the $\lambda!$-calculus it is possible to extract exact measures for CBN. In this sense, the goal of encoding tight typing in a unified framework is achieved.

Call-by-Value. In contrast with the CBN case, the set of type for system $\mathcal{V}$ is not a subset of that for $\mathcal{B}$, and we need to properly translate types. To that end, we introduce two mutually dependent translations $\_^{\overline{\mathtt{v}}}$ and $\_^{\mathtt{v}}$:

$$\begin{array}{rcll}
\mathtt{tt}^{\overline{\mathtt{v}}} & \stackrel{def}{=} & \mathtt{tt} & \text{if } \mathtt{tt} \neq \mathtt{vr} \\
\mathtt{vr}^{\overline{\mathtt{v}}} & \stackrel{def}{=} & \mathtt{n} & \\
(\mathcal{M} \to \sigma)^{\overline{\mathtt{v}}} & \stackrel{def}{=} & \mathcal{M}^{\overline{\mathtt{v}}} \to \sigma^{\mathtt{v}} & \\
([\sigma]_{i\in I})^{\overline{\mathtt{v}}} & \stackrel{def}{=} & [\sigma_i{}^{\overline{\mathtt{v}}}]_{i\in I} &
\end{array}
\qquad
\begin{array}{rcll}
\mathtt{tt}^{\mathtt{v}} & \stackrel{def}{=} & \mathtt{tt} & \text{if } \mathtt{tt} \neq \mathtt{vr} \\
\mathtt{vr}^{\mathtt{v}} & \stackrel{def}{=} & [\mathtt{n}] & \\
(\mathcal{M} \to \sigma)^{\mathtt{v}} & \stackrel{def}{=} & \mathcal{M}^{\overline{\mathtt{v}}} \to \sigma^{\mathtt{v}} & \\
([\sigma]_{i\in I})^{\mathtt{v}} & \stackrel{def}{=} & [\sigma_i{}^{\overline{\mathtt{v}}}]_{i\in I} &
\end{array}$$

Remark that $\mathcal{M}^{\overline{\mathtt{v}}} = \mathcal{M}^{\mathtt{v}}$ for every multitype $\mathcal{M}$. Translation $\_^{\overline{\mathtt{v}}}$ for a context Γ is defined as expected: $\Gamma^{\overline{\mathtt{v}}} = \{x_i : \mathcal{M}_i{}^{\overline{\mathtt{v}}}\}_{i\in I}$. To translate terms, we resort to the function $\_^{\mathtt{v}}$ defined in Sec. 7. We also restrict $\mathcal{B}$ derivations to those that come from the translation of some $\mathcal{V}$ derivation. Indeed, a $\mathcal{B}$ derivation Φ is ***$\mathtt{v}$-relevant*** if:

(1) all the contexts and types involved in Φ are in the image of the translations $\_^{\overline{\mathtt{v}}}$ and $\_^{\mathtt{v}}$ respectively; and

(2) rule $(\mathtt{dr_c})$ is only applied to terms having a type of the form $[\mathcal{M} \to \tau]$.

To state the preservation of typing derivations between systems $\mathcal{V}$ and $\mathcal{B}$ we define measures over type derivations in both systems to conveniently capture the relationship between the exponential steps in the source and the target derivation. The intuition is that we need to compensate for those $\mathtt{d}!$-redexes of the $\lambda!$-calculus that might be introduced when translating. For all the typing rules with two premises, we write Φ_t and Φ_u for the first and second premise respectively. The first measure for system $\mathcal{V}$ is given by induction on Φ as follows:

(1) for $(\mathtt{var_p})$, $\mathtt{e}(\Phi) \stackrel{def}{=} 1$;

(2) for $(\mathtt{val_p})$, $(\mathtt{abs}^{\mathcal{V}}_{\mathtt{p}})$ and $(\mathtt{var}^{\mathcal{V}}_{\mathtt{c}})$, $\mathtt{e}(\Phi) \stackrel{def}{=} 0$;

(3) for $(\mathtt{app}^{\mathcal{V}}_{\mathtt{p}})$, $\mathtt{e}(\Phi) \stackrel{def}{=} \mathtt{e}(\Phi_t) + \mathtt{e}(\Phi_u) - 1$ if $\mathsf{val}(t)$;

(4) for $(\mathtt{app}^{\mathcal{V}}_{\mathtt{c}})$ and $(\mathtt{appt}^{\mathcal{V}}_{\mathtt{c}})$, $\mathtt{e}(\Phi) \stackrel{def}{=} \mathtt{e}(\Phi_t) + \mathtt{e}(\Phi_u) + 1$ if $\neg\mathsf{val}(t)$; and

(5) in any other case $\mathtt{e}(\Phi)$ is defined as the sum of the recursive calls over all premises.

The second measure for system $\mathcal{B}$ is also defined by induction on Φ as follows:

(1) for $(\mathtt{bg_p})$ and $(\mathtt{var_c})$, $\widehat{\mathtt{e}}(\Phi) \stackrel{def}{=} 0$;

(2) for $(\mathtt{app_p})$, $\widehat{\mathtt{e}}(\Phi) \stackrel{def}{=} \widehat{\mathtt{e}}(\Phi_t) + \widehat{\mathtt{e}}(\Phi_u) - 1$ if $\mathsf{val}(t)$;

(3) for $(\mathtt{app_c})$ and $(\mathtt{appt_c})$, $\widehat{\mathtt{e}}(\Phi) \stackrel{def}{=} \widehat{\mathtt{e}}(\Phi_t) + \widehat{\mathtt{e}}(\Phi_u) + 1$ if $\neg\mathsf{val}(t)$; and

(4) in any other case $\widehat{\mathtt{e}}(\Phi)$ is defined as the sum of the recursive calls over all premises.

Then, we obtain:

▶ **Theorem 21.**

1. *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$, *then* $\Phi' \rhd_{\mathcal{B}} \Gamma^{\overline{\mathtt{v}}} \vdash^{(m,e',s)} t^{\mathtt{v}} : \sigma^{\mathtt{v}}$ *is* $\mathtt{v}$*-relevant with* $e' = e + \mathtt{e}(\Phi)$.
2. *If* $\Phi' \rhd_{\mathcal{B}} \Gamma^{\overline{\mathtt{v}}} \vdash^{(m,e',s)} t^{\mathtt{v}} : \sigma^{\mathtt{v}}$ *is* $\mathtt{v}$*-relevant, then* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *with* $e = e' - \widehat{\mathtt{e}}(\Phi')$.

As an example, consider the CBV term $(\lambda x.x)\, y$ (whose derivation Φ is on the left) and its translation $(\lambda x.!\, x)\,!\, y$ into the $\lambda!$-calculus (whose derivation Φ' is on the right):

$$\dfrac{\dfrac{\dfrac{}{x : [\mathtt{vr}] \vdash^{(0,0,0)} x : \mathtt{vr}}}{\vdash^{(0,1,0)} \lambda x.x : [[\mathtt{vr}] \to \mathtt{vr}]} \quad \dfrac{}{y : [\mathtt{vr}] \vdash^{(0,1,0)} y : [\mathtt{vr}]}}{y : [\mathtt{vr}] \vdash^{(1,1,0)} (\lambda x.x)\, y : \mathtt{vr}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{n}] \vdash^{(0,0,0)} x : \mathtt{n}}}{x : [\mathtt{n}] \vdash^{(0,1,0)} !\, x : [\mathtt{n}]}}{\vdash^{(0,1,0)} \lambda x.!\, x : [\mathtt{n}] \to [\mathtt{n}]} \quad \dfrac{\dfrac{}{y : [\mathtt{n}] \vdash^{(0,0,0)} y : \mathtt{n}}}{y : [\mathtt{n}] \vdash^{(0,1,0)} !\, y : [\mathtt{n}]}}{y : [\mathtt{n}] \vdash^{(1,2,0)} (\lambda x.!\, x)\,!\, y : [\mathtt{n}]}$$

Notice that $\mathsf{tight}(\Gamma)$ if and only if $\mathsf{tight}(\Gamma^{\overline{\mathtt{v}}})$. Unfortunately, this is not sufficient to translate a tight derivation in $\mathcal{V}$ into a tight derivation in $\mathcal{B}$. Indeed, the variable x of type $\mathtt{vr}$ translates to $!\, x$ of type $[\mathtt{n}]$. However, this only happens if the derived type is $\mathtt{vr}$, which is an auxiliary type of system $\mathcal{V}$ used to identify variables that are not used as values because they will be applied to some argument. In the case of ***$\mathtt{f}$-relevant $\mathcal{V}$ derivations***, defined as not deriving type $\mathtt{vr}$, tightness is indeed preserved. As a consequence, as for CBN, is it possible to study CBV in the unified framework of the $\lambda!$-calculus and extract exact measures for it by resorting to relevant tight derivations:

▶ **Corollary 22.** *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight and* $\mathtt{f}$*-relevant, then there exists* $p \in \mathsf{no}_{\mathtt{scf}}$ *such that* $t^{\mathtt{v}} \twoheadrightarrow^{(m,e)}_{\mathtt{f}} p$ *with* m $\mathtt{m}$*-steps,* $e + \mathtt{e}(\Phi)$ $\mathtt{e}$*-steps, and* $|p|_{\mathtt{f}} = s$. *Conversely, if* $\Phi' \rhd_{\mathcal{B}} \Gamma^{\overline{\mathtt{v}}} \vdash^{(m,e',s)} t^{\mathtt{v}} : \sigma^{\mathtt{v}}$ *is tight and* $\mathtt{v}$*-relevant, then there exists* $p \in \mathsf{no}_{\mathtt{v}}$ *such that* $t \twoheadrightarrow^{(m,e)}_{\mathtt{v}} p$ *with* m $\mathtt{m}$*-steps,* $e' - \widehat{\mathtt{e}}(\Phi')$ $\mathtt{e}$*-steps, and* $|p|_{\mathtt{v}} = s$.

9 Conclusion

Following recent works exploring the power of CBPV, we develop a technique for deriving tight type systems for CBN/CBV as special cases of a single tight type system for the $\lambda!$-calculus, a subcalculus of CBPV inspired by Linear Logic.

The idea to study semantical and operational properties in a CBPV framework in order to transfer them to CBN/CBV has so far be exploited in different works [42, 25, 26, 32, 16, 12, 46]. Moreover, relational models for CBN/CBV can be derived from the relational model for CBPV, resulting in non-idempotent intersection type systems for them, that provide upper bounds for the length of normalization sequences [12]. However, the challenging quest of a (tight) quantitative type system for CBV, giving *exact measures* for the length of normalization sequences instead of *upper bounds*, and being at the same time encodable in CBPV, has been open. None of the existing proposals [3, 40] could be defined/explained within such an approach. In particular, the tight type systems that we propose for CBN/CBV give independent exact measures for the length of multiplicative and exponential reduction to normal form, as well as the size of these normal forms.

Different topics deserve future attention. One of them is the study of *strong* reduction for the $\lambda!$-calculus, which allows to reduce terms under *all* the constructors, including bang. Appropriate encodings of strong CBN and strong CBV should follow. Linear (head) reduction, as well as other more sophisticated semantics like GOI also deserve some attention.The tight systems presented in this work could also be used to understand bounded computation.

References

1 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *PACMPL*, 2(ICFP):94:1–94:30, 2018. doi:10.1145/3236789.

2 Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. In Atsushi Igarashi, editor, *Programming Languages and Systems – 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016. doi:10.1007/978-3-319-47958-3_12.

3 Beniamino Accattoli and Giulio Guerrieri. Types of fireballs. In Sukyoung Ryu, editor, *Programming Languages and Systems – 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, volume 11275 of *Lecture Notes in Computer Science*, pages 45–66. Springer, 2018. doi:10.1007/978-3-030-02768-1_3.

4 Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In Luís Caires, editor, *Programming Languages and Systems – 28th European Symposium on Programming, ESOP 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 11423 of *Lecture Notes in Computer Science*, pages 410–439. Springer, 2019. doi:10.1007/978-3-030-17184-1_15.

5 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. 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 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

6 Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming – 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16. Springer, 2012. doi:10.1007/978-3-642-29822-6_4.

7 Sandra Alves, Delia Kesner, and Daniel Ventura. A quantitative understanding of pattern matching. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, volume 175 of *LIPIcs*, pages 3:1–3:36. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.TYPES.2019.3.

8 Hendrik P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*, volume 103. North Holland, revised edition, 1984.

9 Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:3)2013.

10 Marc Bezem, Jan Willem Klop, and Vincent van Oostrom. *Term Rewriting Systems (TeReSe)*. Cambridge University Press, 2003.

11 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Logic*, 109(3):205–241, 2001. doi:10.1016/S0168-0072(00)00056-7.

12 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. In Keisuke Nakano and Konstantinos Sagonas, editors, *Functional and Logic Programming – 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*, volume 12073 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2020. doi:10.1007/978-3-030-59025-3_2.

13 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Solvability = typability + inhabitation. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: https://lmcs.episciences.org/7141.

14 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017. doi:10.1093/jigpal/jzx018.

15 Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures – 17th International Conference, FOSSACS 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 8412 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2014. doi:10.1007/978-3-642-54830-7_7.

16 Jules Chouquet and Christine Tasson. Taylor expansion for call-by-push-value. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CSL.2020.16.

17 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, pages 233–243. ACM, 2000. doi:10.1145/351240.351262.

18 Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Sequent calculi for second order logic. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*. Cambridge University Press, 1995.

19 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. PhD thesis, Université Aix-Marseille II, 2007.

20 Daniel de Carvalho. The relational model is injective for multiplicative exponential linear logic. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 – September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 41:1–41:19. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.41.

21 Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, 28(7):1169–1203, 2018. doi:10.1017/S0960129516000396.

22 Daniel de Carvalho and Lorenzo Tortora de Falco. A semantic account of strong normalization in linear logic. *Inf. Comput.*, 248:104–129, 2016. doi:10.1016/j.ic.2015.12.010.

23 Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. A semantic measure of the execution time in linear logic. *Theor. Comput. Sci.*, 412(20):1884–1902, 2011. doi:10.1016/j.tcs.2010.12.017.

24 Thomas Ehrhard. Collapsing non-idempotent intersection types. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) – 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.259.

25 Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In Peter Thiemann, editor, *Programming Languages and Systems – 25th European Symposium on Programming, ESOP 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 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016. doi:10.1007/978-3-662-49498-1_9.

26 Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In James Cheney and Germán Vidal, editors, *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016*, pages 174–187. ACM, 2016. doi:10.1145/2967973.2968608.

27 Claudia Faggian and Giulio Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures – 24th International Conference, FOSSACS 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*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021. doi:10.1007/978-3-030-71995-1_11.

28 Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 1994. doi:10.1007/3-540-57887-0_115.

29 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

30 Jean-Yves Girard. Normal functors, power series and λ-calculus. *Ann. Pure Appl. Logic*, 37(2):129–177, 1988. doi:10.1016/0168-0072(88)90025-5.

31 Giulio Guerrieri. Towards a semantic measure of the execution time in call-by-value lambda-calculus. In Michele Pagani and Sandra Alves, editors, *Proceedings Twelfth Workshop on Developments in Computational Models and Ninth Workshop on Intersection Types and Related Systems, DCM/ITRS 2018, Oxford, UK, 8th July 2018*, volume 293 of *EPTCS*, pages 57–72, 2018. doi:10.4204/EPTCS.293.5.

32 Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA), Oxford, UK, 7-8 July 2018.*, EPTCS, pages 15–30, 2019.

33 Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. Computing connected proof(-structure)s from their taylor expansion. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.20.

34 Axel Kerinec, Giulio Manzonetto, and Simona Ronchi Della Rocca. Call-by-value, again! In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.7.

35 Delia Kesner and Pierre Vial. Types as resources for classical natural deduction. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 24:1–24:17. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.24.

36 Delia Kesner and Pierre Vial. Consuming and persistent types for classical logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 619–632. ACM, 2020. doi:10.1145/3373718.3394774.

37 Delia Kesner and Andrés Viso. Encoding tight typing in a unified framework. *CoRR*, abs/2105.00564, 2021. arXiv:2105.00564.

38 Zurab Khasidashvili. The Church-Rosser theorem in orthogonal combinatory reduction systems. Technical Report 1825, INRIA Rocquencourt, France, 1992.

39 Jan Willem Klop. *Combinatory reduction systems*. PhD thesis, Univ. Utrecht, 1980.

40 Maico Leberle. *Dissecting call-by-need by customizing multi type systems*. PhD thesis, Institut Polytechnique de Paris, 2021.

41 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.

42 Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006. doi:10.1007/s10990-006-0480-6.

43 Giulio Manzonetto, Michele Pagani, and Simona Ronchi Della Rocca. New semantical insights into call-by-value λ-calculus. *Fundam. Informaticae*, 170(1-3):241–265, 2019. doi:10.3233/FI-2019-1862.

44 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *Proc. ACM Program. Lang.*, 2(POPL):6:1–6:28, 2018. doi:10.1145/3158094.

45 Laurent Regnier. Une équivalence sur les lambda-termes. *TCS*, 2(126):281–292, 1994.

46 José Espírito Santo, Luís Pinto, and Tarmo Uustalu. Modal embeddings and calling paradigms. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany.*, volume 131 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.18.

Generalized Universe Hierarchies and First-Class Universe Levels

András Kovács
Eötvös Loránd University, Budapest, Hungary

Abstract

In type theories, universe hierarchies are commonly used to increase the expressive power of the theory while avoiding inconsistencies arising from size issues. There are numerous ways to specify universe hierarchies, and theories may differ in details of cumulativity, choice of universe levels, specification of type formers and eliminators, and available internal operations on levels. In the current work, we aim to provide a framework which covers a large part of the design space. First, we develop syntax and semantics for cumulative universe hierarchies, where levels may come from any set equipped with a transitive well-founded ordering. In the semantics, we show that induction-recursion can be used to model transfinite hierarchies, and also support lifting operations on type codes which strictly preserve type formers. Then, we consider a setup where universe levels are first-class types and subject to arbitrary internal reasoning. This generalizes the bounded polymorphism features of Coq and at the same time the internal level computations in Agda.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases type theory, universes

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.28

Supplementary Material *Software (Source Code)*: `https://github.com/AndrasKovacs/universes/tree/master/agda`; archived at `swh:1:dir:b74a7da080ca804b662e1038e025e76ea202edf3`

Funding The author was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

1 Introduction

Users of type theories often view universe levels as a bureaucratic detail, a necessary annoyance in service of boosting expressive power while retaining logical consistency. However, universe hierarchies are not going away any time soon in practical implementations of type theory. In recent developments of systems, we are getting more universes and more adjacent features:

- Agda recently added a limited cumulativity as an optional feature for universes [9], and the upcoming 2.6.2 version will extend the $\omega + 1$ universe hierarchy to $\omega * 2$.
- Coq added support for cumulative inductive types [26] and a form of bounded universe polymorphism [30].

At this point, there is a veritable zoo of universe features in existing implementations. We have perhaps even more design choices when considering the formal metatheory of type theories. Do type formers stay in the same universe, or take the $\sqcup$ of universes of constituent types? Can eliminators target any universe, or do we instead use lifting operators to cross levels? What kind of universe polymorphism do we have, can we quantify over level bounds? Is there a type of levels, or are levels in a separate syntactic layer?

The aim of the current work is to develop semantics which covers as much as possible from the range of sensible universe features. This way, theorists and language implementors can grab a desired bag of features, and be able to show consistency of their system by a straightforward translation to one of the systems in this paper.

Contributions

1. In Section 3 we describe models of type theories where universe levels may come from any set with a well-founded transitive ordering relation. We specify models as categories equipped with level-indexed diagrams of families, as a variation on categories with families. Each morphism of levels is mapped to a lifting operation on terms and types. By varying the preservation properties of lifting operations, we can describe a range of stratification features, from two-level type theory to cumulative universes.
2. In Section 4 we use induction-recursion to model the mentioned theories. We model the strongest formulations for lifting and universes, namely cumulative universes with Russell-style type decoding.
3. In Section 5 we describe type theories with internal types for levels and level morphisms, and extend the previous inductive-recursive semantics to cover these as well. Here, we can additionally represent various universe polymorphism features and level computations.

We provide an Agda formalization of the contents of the paper at `https://github.com/AndrasKovacs/universes/tree/master/agda`. The formalization is not complete, as we skip proofs involving an excessive number of equality coercions (which are more suited to informal reasoning, using equality reflection), and instead focus on the key points.

2 Metatheory

We work in a Martin-Löf type theory which has the following features.

- Two universes named Set_0 and Set_1, where Set_0 supports inductive-recursive types (IR) as specified by Dybjer and Setzer [12]. We may omit the universe indices if they can be inferred or if we work over arbitrary indices.
- Function extensionality and uniqueness of identity proofs (UIP). Additionally, we assume equality reflection in this paper, thus working in extensional type theory, to avoid noise from equality transports.
- We write function types as $(x : A) \to B$ with $\lambda\, x.\, t$ inhabitants. We may group multiple arguments with the same type, as in $(x\, y : A) \to B$. We have Σ-types as $(x : A) \times B$, with pairing as $(t,\, u)$. We have $\top$ as the unit type with inhabitant tt, $\bot$ as the empty type, and Bool with true and false inhabitants. Propositional identity is written as $t = u$ (coinciding with definitional equality).
- We occasionally use $\{x : A\} \to B$ for an Agda-like notation for function types with implicit arguments. We usually omit implicit applications but may explicitly write them as $t\,\{u\}$. We may omit implicit function types altogether if it is clear where certain variables are quantified.

3 Generalized Universe Hierarchies

In this section, we first describe notions of models for type theories with generalized universes, and discuss several variations of universes and lifting operations. Then, we pick a concrete variant (the strongest, in a sense) and construct a model for it in the metatheory.

For the basic structure of typing contexts and substitutions, let us review categories with families.

3.1 Categories with Families

▶ **Definition 1.** A *category with family* (cwf) [11] consists of the following data:

- A category with a terminal object. We denote the set of objects as $\mathsf{Con} : \mathsf{Set}$ and use capital Greek letters starting from Γ to refer to objects. The set of morphisms is $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$, and we use σ, δ and so on to refer to morphisms. The terminal object is $\bullet$ with unique morphism $\epsilon : \mathsf{Sub}\,\Gamma\,\bullet$. In initial models (that is, syntaxes) of type theories, objects correspond to typing contexts, morphisms to parallel substitutions and the terminal object to the empty context; this informs the naming scheme.
- A *family structure*, containing $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}$ and $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$, where Ty is a presheaf over the category of contexts and Tm is a presheaf over the category of elements of Ty. This means that both types (Ty) and terms (Tm) can be substituted, and substitution has functorial action. We use A, B, C to refer to types and t, u, v to refer to terms, and use $A[\sigma]$ and $t[\sigma]$ for substituting types and terms. Additionally, a family structure has *context comprehension* which consists of a context extension operation $- \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$ together with an isomorphism $\mathsf{Sub}\,\Gamma\,(\Delta \triangleright A) \simeq ((\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \times \mathsf{Tm}\,\Gamma\,(A[\sigma]))$ which is natural in Γ.

From the comprehension structure, we recover the following notions:

- By going right-to-left along the isomorphism, we recover *substitution extension* $-, - : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma]) \to \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)$. This means that starting from ϵ or the identity substitution id, we can iterate $-, -$ to build substitutions as lists of terms.
- By going left-to-right, and starting from $\mathsf{id} : \mathsf{Sub}\,(\Gamma \triangleright A)\,(\Gamma \triangleright A)$, we recover the *weakening substitution* $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$ and the *zero variable* $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])$.
- By weakening q, we recover a notion of variables as De Bruijn indices. In general, the n-th De Bruijn index is defined as $\mathsf{q}[\mathsf{p}^n]$, where p^n denotes n-fold composition.

There are other ways for presenting the basic categorical structure of models, which are nonetheless equivalent to cwfs, including natural models [3] and categories with attributes [6]. We use the cwf presentation for its immediately algebraic character and closeness to conventional explicit substitutions. We consider the syntax of a type theory to be its initial model.

▶ Notation 1. As De Bruijn indices are hard to read, we will mostly use nameful notation for binders. For example, assuming $\mathsf{Nat} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Ty}\,\Gamma$ and $\mathsf{Id} : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$, we may write $\bullet \triangleright (n : \mathsf{Nat}) \triangleright (p : \mathsf{Id}\,\mathsf{Nat}\,n\,n)$ for a typing context, instead of using numbered variables or cwf combinators as in $\bullet \triangleright \mathsf{Nat} \triangleright \mathsf{Id}\,\mathsf{Nat}\,\mathsf{q}\,\mathsf{q}$.

▶ Notation 2. In the following, we will denote families by (Ty,Tm) pairs and overload context extension $- \triangleright -$ for different families.

A family structure may be closed under certain *type formers*. For example, we may close a family over function types by assuming $\Pi : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$ together with abstraction, application, $\beta\eta$-rules, and equations for the action of substitution on type and term formers.

In the following, whenever we introduce a type or term former, we always assume that it is natural with respect to substitution, i.e. all type and term formers have a corresponding substitution rule. This convention could be made precise by working in a framework for higher-order abstract syntax, where all specified structure is automatically stable under substitution [25, 27, 5]. While this can be effective at reducing formal clutter, this paper only presents models which are technically straightforward, so we choose not to use higher-order signatures, in order to make the presentation more direct.

3.2 Morphisms and Inclusions of Families

In the rest of the paper we make use of categories equipped with possibly multiple family structures, which serves as basis for specifying universe hierarchies. However, it is not very useful to simply have multiple copies of family structures together with their type formers. In that case, every constructor and eliminator of every type former stays in the same family, and there is no interaction between families, and the most we can do is to mix them together in typing contexts. In this subsection we describe several ways of crossing between families.

▶ **Definition 2.** A *family morphism* F between $(\mathsf{Ty}_0, \mathsf{Tm}_0)$ and $(\mathsf{Ty}_1, \mathsf{Tm}_1)$ families consists of natural transformations mapping types to types and terms to terms, which preserves context extensions up to context isomorphism, i.e. we have that $(\Gamma \triangleright F\, A) \simeq (\Gamma \triangleright A)$, where $\simeq$ denotes existence of an invertible context morphism.

Family morphisms are restrictions of so-called *weak morphisms* [4] (or *pseudomorphisms* [18]) of cwfs: a weak morphism which has the identity action on the base category is exactly a family morphism.

▶ **Lemma 1.** Every family morphism has invertible action on terms, i.e. there is an $F^{-1} : \mathsf{Tm}\,\Gamma\,(F\,A) \to \mathsf{Tm}\,\Gamma\,A$.

Proof. From the $\triangleright$-preservation isomorphism and the defining isomorphisms of comprehension, we get $\mathsf{q}' : \mathsf{Tm}\,(\Gamma \triangleright F\,A)\,(A[\mathsf{p}])$ such that $F\,\mathsf{q}' = \mathsf{q}$ and $\mathsf{q}'[\mathsf{p},\, F\,\mathsf{q}] = \mathsf{q}$. Now, for $t : \mathsf{Tm}\,\Gamma\,(F\,A)$, we define $F^{-1}\,t$ as $\mathsf{q}'[\mathsf{id},\, t] : \mathsf{Tm}\,\Gamma\,A$. We get the following:

$$F(F^{-1}\,t) = F(\mathsf{q}'[\mathsf{id},\, t]) = (F\,\mathsf{q}')[\mathsf{id},\, t] = \mathsf{q}[\mathsf{id},\, t] = t$$
$$F^{-1}(F\,t) = \mathsf{q}'[\mathsf{id},\, F\,t] = \mathsf{q}'[\mathsf{id},\, (F\,\mathsf{q})[\mathsf{id},\ t]] = \mathsf{q}'[\mathsf{p},\, F\,\mathsf{q}][\mathsf{id},\, t] = \mathsf{q}[\mathsf{id},\, t] = t$$

More concisely, F is invertible on the generic term q, which implies invertibility on any term. ◀

▶ Notation 3. In the following, we will write $\mathsf{Lift} : \mathsf{Ty}_0\,\Gamma \to \mathsf{Ty}_1\,\Gamma$ for the action of some morphism on types, $\uparrow : \mathsf{Tm}_0\,\Gamma\,A \to \mathsf{Tm}_1\,\Gamma\,(\mathsf{Lift}\,A)$ for the action on terms, and $\downarrow$ for the inverse action on terms. We will also call the action on types *type lifting* and the action on terms *term lifting*.

We may think about the relation between *modalities* and morphisms. The main difference is that morphisms impose no structural restrictions on variables and contexts. More concretely, every Lift is dependent right adjoint [4] to the identity functor on the base category, as we have $\mathsf{Tm}\,(\mathsf{Id}\,\Gamma)\,A \simeq \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,A)$. Hence, every morphism can be viewed as a degenerate modality.

Assume family structures $(\mathsf{Ty}_0, \mathsf{Tm}_0)$ and $(\mathsf{Ty}_1, \mathsf{Tm}_1)$ and a morphism between them. This corresponds to a basic version of *two-level type theory* [2]. This theory has an interpretation in presheaves over the category of contexts of some chosen model of a type theory, where $(\mathsf{Ty}_0, \mathsf{Tm}_0)$ is modeled using structure in the chosen model, and $(\mathsf{Ty}_1, \mathsf{Tm}_1)$ is modeled using presheaf constructions. More illustratively, this means interpreting $(\mathsf{Ty}_1, \mathsf{Tm}_1)$ as a metaprogramming layer which can generate object-level constructions in the $(\mathsf{Ty}_0, \mathsf{Tm}_0)$ layer. Lifted types correspond to types of object-level terms; for example, $\mathsf{Bool}_0 : \mathsf{Ty}_0\,\Gamma$ is the object-level type of Booleans, while $\mathsf{Lift}\,\mathsf{Bool}_0$ is the meta-level type of Bool_0-terms, and $\mathsf{Bool}_1 : \mathsf{Ty}_1\,\Gamma$ is the type of meta-level Booleans. It is possible to compute a Bool_0 from a Bool_1. Given $b : \mathsf{Tm}_1\,\Gamma\,\mathsf{Bool}_1$, we can construct $\downarrow(\mathsf{if}\,b\,\mathsf{then}\uparrow\mathsf{true}_0\,\mathsf{else}\uparrow\mathsf{false}_0) : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_0$. But there is no way to compute a Bool_1 from a Bool_0: we can try to lift the input, but there is no elimination rule for $\mathsf{Lift}\,\mathsf{Bool}_0$ in Ty_1.

Hence, plain family morphisms can model a metaprogramming hierarchy, but currently we are aiming for "sizing" hierarchies instead. This means that we want to eliminate from any family to any other family which is connected by a morphism.

▶ **Definition 3.** A *family inclusion* is a family morphism which preserves all type and term formers. This assumes that every type former which is contained in the source family, is also contained in the target family.

Some examples for preservation equations for type and term formers:

$$\begin{aligned}
\mathsf{Lift}\,(\Pi\,(x : A)B) &= \Pi\,(x : \mathsf{Lift}\,A)(\mathsf{Lift}\,(B[x \mapsto \downarrow x])) \\
\uparrow(\lambda\,(x : A).\,t) &= \lambda\,(x : \mathsf{Lift}\,A).\uparrow(t[x \mapsto \downarrow x]) \\
\mathsf{Lift}\,\mathsf{Bool}_0 &= \mathsf{Bool}_1 \\
\uparrow\mathsf{true}_0 &= \mathsf{true}_1
\end{aligned}$$

In general, we can skip specifying preservation for $\downarrow$, since it follows from $\uparrow$ preservation equations.

Assume an inclusion from $(\mathsf{Ty}_0,\ \mathsf{Tm}_0)$ to $(\mathsf{Ty}_1,\ \mathsf{Tm}_1)$. Now, we can eliminate from Bool_0 to Bool_1. If we have some $b : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_0$, we also have $\uparrow b : \mathsf{Tm}_1\,\Gamma\,(\mathsf{Lift}\,\mathsf{Bool}_0)$, hence $\uparrow b : \mathsf{Tm}_1\,\Gamma\,\mathsf{Bool}_1$. Then, we can use Bool_1 elimination, as in $\mathsf{if}\uparrow b\,\mathsf{then}\,\mathsf{true}_1\,\mathsf{else}\,\mathsf{false}_1 : \mathsf{Tm}_1\,\Gamma\,\mathsf{Bool}_1$. The $\uparrow$ computation ensures that the eliminator computes appropriately on canonical terms: if b is true_0, we get $\uparrow\mathsf{true}_0 = \mathsf{true}_1$ as the if-then-else scrutinee.

A family inclusion corresponds to a *cumulative hierarchy* consisting of two families: every type former of the smaller family is included in the larger family, with the same elimination rules.

▶ **Definition 4.** A *strict family inclusion* between $(\mathsf{Ty}_0,\ \mathsf{Tm}_0)$ and $(\mathsf{Ty}_1,\ \mathsf{Tm}_1)$ is a family inclusion $(\mathsf{Lift},\ \uparrow,\ \downarrow)$ for which the following equations hold:

$$(\Gamma \triangleright \mathsf{Lift}\,A) = (\Gamma \triangleright A) \tag{1}$$

$$\mathsf{Tm}_1\,\Gamma\,(\mathsf{Lift}\,A) = \mathsf{Tm}_0\,\Gamma\,A \tag{2}$$

$$\uparrow t = t \tag{3}$$

A strict inclusion corresponds to Sterling's *algebraic cumulativity* [24]. The additional equations are a matter of convenience: they allow us to omit term liftings in informal syntax[1]. Most of the time we can also omit level annotations on term formers. For example, we have $\mathsf{true}_0 : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_0$, but also $\mathsf{true}_0 : \mathsf{Tm}_0\,\Gamma\,(\mathsf{Lift}\,\mathsf{Bool}_0)$, hence $\mathsf{true}_0 : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_1$. Moreover, true_0 is definitionally equal to true_1, since $\mathsf{true}_0 = \uparrow\mathsf{true}_0 = \mathsf{true}_1$. Thus, using simply true is fine whenever the family is clear from context.

The definitional equality of true_0 and true_1 is important; without it canonicity would fail, since true_0, false_0, true_1 and false_1 would be four definitionally distinct inhabitants of Bool_1. See Luo [19] for a discussion of related issues with cumulativity. It is not sufficient to specify a strict inclusion just by equations 1 and 2 in Definition 4. We need $\uparrow$ together with equation 3 to identify term formers in different families. The other direction $\downarrow t = t$ is immediately derivable.

[1] In a proof assistant, often we would still have to explicitly transport along the strict inclusion equations.

3.3 Level Structures

We would like to describe a range of setups with multiple families and morphisms between them. In this subsection we describe the indexing structures for such family diagrams. First, we specify a notion of well-foundedness, which will be used to preclude size paradoxes in universe hierarchies.

▶ **Definition 5.** The *accessibility predicate* on relations is defined by the following inductive rules:

$$\mathsf{Acc} : \{A : \mathsf{Set}\} \to (R : A \to A \to \mathsf{Set}) \to A \to \mathsf{Set}$$
$$\mathsf{acc} : \{a : A\} \to ((a' : A) \to R\,a'\,a \to \mathsf{Acc}\,R\,a') \to \mathsf{Acc}\,R\,a$$

See [1] and [28, Section 10.3] for further exposition. An inhabitant of $\mathsf{Acc}\,R\,a$ proves that starting from $a : A$, all descending R-chains must be finite. This is ensured by the universal property of the inductive definition.

▶ **Lemma 2.** All inhabitants of $\mathsf{Acc}\,R\,a$ are equal [28, Lemma 10.3.4]. In other words, accessibility is proof-irrelevant.

▶ **Definition 6.** A relation $R : A \to A \to \mathsf{Set}$ is *well-founded* if $(a : A) \to \mathsf{Acc}\,R\,a$.

▶ **Definition 7.** A *level structure* consists of the following components:

$$\begin{aligned}
&\mathsf{Lvl} && : \mathsf{Set}_0\\
&- < - && : \mathsf{Lvl} \to \mathsf{Lvl} \to \mathsf{Set}_0\\
&< \mathsf{prop} && : (p\,q : i < j) \to p = q\\
&- \circ - && : j < k \to i < j \to i < k\\
&< \mathsf{wf} && : (i : \mathsf{Lvl}) \to \mathsf{Acc}\ <\ i
\end{aligned}$$

We overload Lvl to refer to a given level structure and also its underlying set. In short, a level structure is a set together with a transitive well-founded relation.

▶ **Definition 8.** A *family diagram* over Lvl maps each $i : \mathsf{Lvl}$ to a family structure $(\mathsf{Ty}_i, \mathsf{Tm}_i)$, and each $p : i < j$ to a family inclusion $(\mathsf{Lift}_i^j\,p, \uparrow_i^j p, \downarrow_i^j p)$ between $(\mathsf{Ty}_i, \mathsf{Tm}_i)$ and $(\mathsf{Ty}_j, \mathsf{Tm}_j)$. Moreover, the mapping is functorial, so $\mathsf{Lift}_i^k\,(p \circ q)\,A = \mathsf{Lift}_j^k\,p\,(\mathsf{Lift}_i^j\,q\,A)$, and similarly for $\uparrow_i^j p$ and $\downarrow_i^j p$. A *strict family diagram* is a family diagram where each inclusion is strict.

▶ Notation 4. Sometimes we omit some of the i, j, p annotations from type and term liftings, if they are clear from context.

Our choice of level structures and diagrams is motivated by the following. First, we do not need identity morphisms in levels, because they would be mapped to trivial liftings, which are not interesting in our setting. Second, we do not need proof-relevant level morphisms, since any parallel pair of morphisms gives rise to isomorphic types. Concretely, given $p : i < j$ and $q : i < j$ such that $p \neq q$, we have $\mathsf{Tm}_j\,\Gamma\,(\mathsf{Lift}\,p\,A) \simeq \mathsf{Tm}_i\,\Gamma\,A \simeq \mathsf{Tm}_j\,\Gamma\,(\mathsf{Lift}\,q\,A)$, and since $\mathsf{Lift}\,p\,A$ and $\mathsf{Lift}\,q\,A$ are in the same family, we can internally prove them isomorphic using function types and identity types. That said, every construction in this paper would still work with direct categories as level structures.

3.4 Universes

At this point, we can talk about family diagrams, but no previously seen type former depends on levels in an interesting way. For example, Bool_i has the same inhabitants as Bool_j, for any i and j. Universes introduce dependency on levels, by serving as classifiers for smaller families internally to larger families.

▶ **Definition 9.** A family diagram supports *universe formation* if it supports the following:

$$\mathsf{U} \quad : (i\,j : \mathsf{Lvl}) \to i < j \to \mathsf{Ty}_j\,\Gamma$$
$$\mathsf{LiftU} : \mathsf{Lift}^k_j\,p\,(\mathsf{U}\,i\,j\,q) = \mathsf{U}\,i\,k\,(p \circ q)$$

We also need a way to pin down universes as classifiers. We consider two variants.

▶ **Definition 10.** A family diagram has *Coquand universes* [8] if it has universe formation and additionally supports $\mathsf{El} : \mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p) \to \mathsf{Ty}_i\,\Gamma$, and its inverse $\mathsf{Code} : \mathsf{Ty}_i\,\Gamma \to \mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p)$.

▶ **Definition 11.** A family diagram has *Russell universes* if it has Coquand universes and additionally satisfies $\mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p) = \mathsf{Ty}_i\,\Gamma$ and $\mathsf{El}\,t = t$.

The move from Coquand to Russell universes is fairly similar to the move from inclusions to strict inclusions. The Russell variant makes it possible to informally omit El and Code. Likewise, the $\mathsf{El}\,t = t$ condition ensures appropriate naturality. If we only assumed $\mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p) = \mathsf{Ty}_i\,\Gamma$ but not Coquand universes, we would not be able to prove that a $t : \mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p)$ substituted *as a term* is the same thing as t substituted *as a type*. Both would be written as $t[\sigma]$ in our notation, but they involve different $-[-]$ operations.

Unlike every other type or term former, there is no lifting computation rule for El and Code. Intuitively, the issue is that we would need to relate type lifting and term lifting, but while term lifting is invertible, type lifting is not. Lift sends a $\mathsf{Ty}_i\,\Gamma$ to a $\mathsf{Ty}_j\,\Gamma$, and $\mathsf{Ty}_j\,\Gamma$ is not isomorphic to $\mathsf{Ty}_i\,\Gamma$, because it contains more universes. So, for example, lifting $\mathsf{Bool}_0 : \mathsf{Ty}_0\,\Gamma$ as a type to $\mathsf{Ty}_1\,\Gamma$ yields Bool_1, but lifting Bool_0 as a term yields Bool_0.

Assuming Coquand or Russell universes and $p : i < j$, we can recover polymorphic functions, for example, we may have $id : \Pi(A : \mathsf{U}\,i\,j\,p)(\mathsf{Lift}\,p\,(\mathsf{El}\,A) \to \mathsf{Lift}\,p\,(\mathsf{El}\,A))$ for the polymorphic identity function. Here, we quantify over terms of U, and since every type former stays on the same level (including Π), we have to Lift the types in the codomain to match the level of the domain. We can also recover large elimination, for example as in

$$(\lambda\,(b : \mathsf{Bool}_j).\,\mathsf{if}\,b\,\mathsf{then}\,\mathsf{Code}\,\top_i\,\mathsf{else}\,\mathsf{Code}\,\bot_i) : \mathsf{Tm}_j\,\Gamma\,(\mathsf{Bool}_j \to \mathsf{U}\,i\,j\,p).$$

4 Semantics

In this section we give a model for a type theory with generalized universes. Let us make the notion of model concrete first.

▶ **Definition 12** (Notion of model for a type theory with generalized universes (TTGU))**.** Fix a Lvl structure. A model for TTGU consists of

1. A base category $(\mathsf{Con}, \mathsf{Sub})$ with a terminal object $\bullet$.
2. A strict family diagram $(\mathsf{Ty}_i, \mathsf{Tm}_i)$ over Lvl, supporting Russell universes, and each family structure is closed under the same basic type formers.

The choice of available basic type formers is up to personal taste, and it will not significantly affect the following model construction.

Both in families and universes we choose the stricter formulation, since if we give a model which proves the strict syntax consistent, we immediately get a model which proves the weak syntax consistent[2].

4.1 Inductive-Recursive Codes

The task is to interpret the Lvl-many universes of TTGU using an assumed metatheoretic feature. For this, we need to define a Lvl-indexed type of type codes. Since Lvl and $- < -$ can be arbitrary, we effectively need to define transfinite hierarchies of codes. We use an inductive-recursive [12] definition for the following reasons.

First, induction-recursion is already supported in the Agda proof assistant, and it is very useful to be able to sketch out ideas in a machine-checked setting. It would be much harder to do the same when developing semantics in set theory.

Second, could we use type-theoretic features with simpler specifications than induction-recursion, such as super universes [22] or Mahlo universes [23]? These are sufficient to model transfinite hierarchies. However, using these it is not clear how to additionally support the strict type former preservation property of Lift[3].

Therefore, we give a custom definition using induction-recursion, which corresponds more directly to TTGU structure. Our definition is essentially the same as McBride's redundancy-free hierarchy in [21, Section 6.3.1], but we generalize levels from natural numbers to arbitrary level structures.

▶ **Definition 13** (Codes for the universe). Assume $i : \mathsf{Lvl}$ and $f : (j : \mathsf{Lvl}) \to j < i \to \mathsf{Set}_0$. We define U^{IR} and $\mathsf{El}^{\mathsf{IR}}$ by induction-recursion:

$$
\begin{array}{lll}
\mathsf{U}^{\mathsf{IR}} & : \mathsf{Set}_0 & \mathsf{El}^{\mathsf{IR}} : \mathsf{U}^{\mathsf{IR}} \to \mathsf{Set}_0 \\
\mathsf{U}' & : (j : \mathsf{Lvl}) \to j < i \to \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,(\mathsf{U}'\,j\,p) \;\; = f\,j\,p \\
\Pi' & : (A : \mathsf{U}^{\mathsf{IR}}) \to (\mathsf{El}^{\mathsf{IR}}\,A \to \mathsf{U}^{\mathsf{IR}}) \to \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,(\Pi'\,A\,B) = (a : \mathsf{El}^{\mathsf{IR}}\,A) \to \mathsf{El}^{\mathsf{IR}}\,(B\,a) \\
\bot' & : \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,\bot' \;\; = \bot \\
\mathsf{Bool}' & : \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,\mathsf{Bool}' \;\; = \mathsf{Bool}
\end{array}
$$

We use the prime accents ($'$) to disambiguate inductive-recursive codes from type formers in TTGU or the metatheory. For basic type formers, we only include codes for function types, the empty type, and Bool. Other type formers are straightforward to add (and we do have more in the Agda formalization).

▶ Notation 5. We may write $\mathsf{U}^{\mathsf{IR}}{}_{i\,f}$ and $\mathsf{El}^{\mathsf{IR}}{}_{i\,f}$ in order to make parameters explicit.

(U^{IR}, $\mathsf{El}^{\mathsf{IR}}$) can be viewed as a *universe operator*: given semantics for an initial segment of Lvl (given by i and f), we create a new universe which is closed under basic type formers, and also closed under all sets in f by the way of U'. Most importantly, this operation can be transfinitely iterated. We first define universes for initial segments of Lvl, by induction on the accessibility of levels:

$$
\begin{array}{l}
\mathsf{U}_< \; : (i : \mathsf{Lvl})\{p : \mathsf{Acc}\,(- < -)\,i\} \to (j : \mathsf{Lvl}) \to j < i \to \mathsf{Set}_0 \\
\mathsf{U}_<\,i\,\{\mathsf{acc}\,f\}\,j\,p = \mathsf{U}^{\mathsf{IR}}{}_{j\,(\mathsf{U}_<\,j\,\{f\,j\,p\})}
\end{array}
$$

---

[2] We always get *initial* and *terminal* models automatically, because of the algebraic character of the theories in this paper. We also get a *freely generated* strict model from a weak model, from the left adjoint of the functor which forgets the strictness equations. But none of these tricks can be used to automatically get a consistency proof.

[3] Palmgren calls this property as having *recursive sub-universes* [22].

▶ **Definition 14** (Semantic universe)**.** Since every level is accessible, we can define the full semantic hierarchy and its decoding function.

$$\mathsf{U} : \mathsf{Lvl} \to \mathsf{Set}_0 \qquad \mathsf{El} : \{i : \mathsf{Lvl}\} \to \mathsf{U}\, i \to \mathsf{Set}_0$$
$$\mathsf{U}\, i = \mathsf{U}^{\mathsf{IR}}{}_{i\,(\mathsf{U}_<\, i\, \{<\mathsf{wf}\, i\})} \qquad \mathsf{El}\, \{i\} = \mathsf{El}^{\mathsf{IR}}{}_{i\,(\mathsf{U}_<\, i\, \{<\mathsf{wf}\, i\})}$$

▶ **Lemma 3.** Assuming $p : i < j$, we have the computation rule $\mathsf{U}_<\, j\, p = \mathsf{U}\, j$. Proof: we may assume that any witness for $\mathsf{Acc}\, (- <-)\, i$ is of the form $\mathsf{acc}\, f$ for some f. Then the equation becomes $\mathsf{U}^{\mathsf{IR}}{}_{j\,(\mathsf{U}_<\, j\, \{f\, j\, p\})} = \mathsf{U}^{\mathsf{IR}}{}_{j\,(\mathsf{U}_<\, j\, \{<\mathsf{wf}\, j\})}$, but by Lemma 2 the $f\, j\, p$ and $<\mathsf{wf}\, j$ witnesses are equal. ◀

▶ **Definition 15** (Semantic Lift)**.** We define by induction on U^{IR} a function with type $(p : i < j) \to (A : \mathsf{U}\, i) \to (A' : \mathsf{U}\, j) \times (\mathsf{El}\, A' = \mathsf{El}\, A)$. However, for the sake of clarity, we present this here as two (mutual) functions:

$$\mathsf{Lift} \quad : (p : i < j) \to \mathsf{U}\, i \to \mathsf{U}\, j$$
$$\mathsf{ElLift} : (p : i < j) \to (A : \mathsf{U}\, i) \to \mathsf{El}\, (\mathsf{Lift}\, A) = \mathsf{El}\, A$$

Let us look at Lift first:

$$\begin{aligned}
\mathsf{Lift}\, p\, (\mathsf{U}'\, k\, q) &= \mathsf{U}'\, k\, (p \circ q)\\
\mathsf{Lift}\, p\, (\Pi'\, A\, B) &= \Pi'\, (\mathsf{Lift}\, p\, A)\, (\lambda\, a.\, \mathsf{Lift}\, p\, (B\, a))\\
\mathsf{Lift}\, p\, \bot' &= \bot'\\
\mathsf{Lift}\, p\, \mathsf{Bool}' &= \mathsf{Bool}'
\end{aligned}$$

Above, the Π' definition is well-typed by $\mathsf{ElLift}\, p\, A$. For the proof of ElLift, the only interesting case is U'. Here, we need to show $\mathsf{U}_<\, j\, k\, (p \circ q) = \mathsf{U}_<\, i\, k\, q$, but by Lemma 3 both sides are $\mathsf{U}\, k$.

▶ **Lemma 4.** Properties of Lift:

1. Lift preserves all basic type formers; this is immediate from the definition.
2. Lift is functorial, i.e. $\mathsf{Lift}\, (p \circ q)\, A = \mathsf{Lift}\, p\, (\mathsf{Lift}\, q\, A)$. This follows by induction on A, and we make use of the irrelevance of $- <-$ in the U' case. ◀

4.2 Inductive-Recursive Model of TTGU

We give a model of TTGU in this section.

▶ Notation 6. To avoid name clashing between components of the model and metatheoretic definitions, we use **bold** font to refer to TTGU components.

▶ **Definition 16** (Base category)**.** The base category is simply the category of sets and functions in Set_0, i.e. $\mathbf{Con} = \mathsf{Set}_0$, $\mathbf{Sub}\, \Gamma\, \Delta = \Gamma \to \Delta$, and the terminal object is $\top$.

▶ **Definition 17** (Family diagram)**.** We map $i : \mathsf{Lvl}$ to a family structure as follows.

$$\mathbf{Ty}_i\, \Gamma = \Gamma \to \mathsf{U}\, i \qquad \mathbf{Tm}_i\, \Gamma\, A = (\gamma : \Gamma) \to \mathsf{El}\, (A\, \gamma)$$

Type and term substitution are given by composition with some function $\sigma : \Gamma \to \Delta$. Comprehension structure is given by $\Gamma \triangleright A = (\gamma : \Gamma) \times \mathsf{El}\, (A\, \gamma)$. Type lifting along $p : i < j$ is as follows:

$$\mathbf{Lift}^j_i p : \mathbf{Ty}_i\, \Gamma \to \mathbf{Ty}_j\, \Gamma$$
$$\mathbf{Lift}^j_i p\, A = \lambda\, \gamma.\, \mathsf{Lift}^j_i p\, (A\, \gamma)$$

Now, two of the strict inclusion equations follow from ElLift, namely $(\Gamma \triangleright \mathbf{Lift}_i^j\, p\, A) = (\Gamma \triangleright A)$ and $\mathbf{Tm}_j\, \Gamma\, (\mathbf{Lift}_i^j\, p\, A) = \mathbf{Tm}_i\, \Gamma\, A$. Thus, we can just define term lifting as $\uparrow_i^j p\, t = t$ and $\downarrow_i^j p\, t = t$. Basic type formers are as follows.

$$\mathbf{\Pi}\, A\, B = \lambda\, \gamma.\, \Pi'\, (A\, \gamma)\, (\lambda\, \alpha.\, B\, (\gamma,\, \alpha)) \qquad \bot_i = \lambda\, \gamma.\, \bot' \qquad \mathbf{Bool}_i = \lambda\, \gamma.\, \mathsf{Bool}'$$

$\mathbf{Lift}_i^j p$ preserves type formers by Lemma 4. We define basic term formers and eliminators using metatheoretic features, e.g. $\mathbf{true}_i = \lambda\, \gamma.\, \mathsf{true}$ and $(\boldsymbol{\lambda}_i\, x.\, t) = \lambda\, \gamma\, \alpha.\, t\, (\gamma,\, \alpha)$. Note that since semantic term formers are just external constructors, they do not depend on levels, so e.g. $\mathbf{true}_i$ is the same at all i. This implies that $\uparrow_i^j p$ preserves term formers as well, so $(\mathbf{Lift}_i^j p,\, \uparrow_i^j p,\, \downarrow_i^j p)$ is a strict family inclusion.

We define universes as $\mathbf{U}\, i\, j\, p = \lambda\, \gamma.\, \mathsf{U}_i'\, j\, p$. With this, $\mathbf{Lift}_j^k\, p\, (\mathbf{U}\, i\, j\, q) = \mathbf{U}\, i\, k\, (p \circ q)$ follows by the definition of semantic Lift. The Russell universe equation $\mathbf{Tm}_j\, \Gamma\, (\mathbf{U}\, i\, j\, p) = \mathbf{Ty}_i\, \Gamma$ follows from Lemma 3, so we can define **El** and **Code** as identity functions.

▶ **Theorem 1** (Consistency of TTGU). *There is no closed syntactic term of $\bot_i$ for any i.*

Proof. Assuming a syntactic $t : \mathsf{Tm}_i \bullet \bot_i$, we can interpret it in the previously given model, which yields an inhabitant of the metatheoretic $\bot$, hence a contradiction. ◀

5 First-Class Universe Levels

In the following, we specify and model type theories where levels and their morphisms are represented by internal types.

However, it would be awkward to pick a particular structure for levels, and specify a type theory which internalizes that structure; for example internalizing levels as natural numbers. We do not want to repeat the specification and semantics for each choice of level structure; instead, we aim to have a more generic solution.

1. We first give a specification of *type theory with dependent levels*, or TTDL, where levels and level morphisms may depend on typing contexts. Here, liftings, universes and type formers are specified, but the internal structure of levels is not yet pinned down.
2. We show that we can extend TTDL with *level reflection* rules, which identify levels with particular internal types, thereby getting *type theories with first-class levels*, or TTFL.

This decreases the amount of work that we have to do, in order to get semantics for different level setups. We only need to pick an external level structure such that it can be also represented using TTDL type formers.

▶ **Definition 18.** A model of TTDL consists of the following.

1. A base category $(\mathsf{Con},\, \mathsf{Sub})$ with terminal object $\bullet$.
2. A "dependent" level structure on the base category:

$$\begin{aligned}
&\mathsf{Lvl} && : \mathsf{Con} \to \mathsf{Set}\\
&- < - && : \{\Gamma : \mathsf{Con}\} \to \mathsf{Lvl}\, \Gamma \to \mathsf{Lvl}\, \Gamma \to \mathsf{Set}\\
&< \mathsf{prop} && : (p\, q : i < j) \to p = q\\
&- \circ - && : j < k \to i < j \to i < k
\end{aligned}$$

Additionally, Lvl and $- < -$ are natural in the base category, so they support substitution operations. *Remark:* at this point, we do not require well-foundedness for $- < -$, as it has no bearing on basic lifting and universe rules, and well-foundedness will be usually internally provable when we add level reflection rules.

3. A "bootstrapping" assumption on levels. This can be any non-empty collection of levels and morphisms. It will be used shortly in Section 5.1, where we specify first-class levels using the syntax (i.e. the initial model) of TTDL. Without bootstrapping, the syntax is trivial and has no closed types. Of course, models of TTDL in general make sense without the bootstrapping assumption.
   We pick the assumption that $l_0, l_1 : \mathsf{Lvl}\,\Gamma$ exist together with $l_{01} : l_0 < l_1$. This allows large eliminations on type formers, so it provides a fair amount of power for specifying internal levels.
4. A family structure:

$$\begin{aligned}
&\mathsf{Ty} &&: (\Gamma : \mathsf{Con}) \to \mathsf{Lvl}\,\Gamma \to \mathsf{Set}\\
&\mathsf{Tm} &&: (\Gamma : \mathsf{Con})\{i : \mathsf{Lvl}\,\Gamma\} \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Set}\\
&- \triangleright - &&: (\Gamma : \mathsf{Con})\{i : \mathsf{Lvl}\,\Gamma\} \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Con}
\end{aligned}$$

   We have type and term substitution, which depends on level substitution. For instance, we have:

$$-[-] : \mathsf{Ty}\,\Delta\,i \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Ty}\,\Gamma\,(i[\sigma])$$

   We also have a comprehension isomorphism $\mathsf{Sub}\,\Gamma\,(\Delta \triangleright A) \simeq (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \times \mathsf{Tm}\,\Gamma\,(A[\sigma])$, which is natural in Γ.
5. A lifting structure with

$$\begin{aligned}
&\mathsf{Lift} : \{\Gamma : \mathsf{Con}\}\{i\,j : \mathsf{Lvl}\,\Gamma\} \to i < j \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,\Gamma\,j\\
&\uparrow \quad : \{\Gamma : \mathsf{Con}\}\{i\,j : \mathsf{Lvl}\,\Gamma\}(p : i < j) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,p\,A)
\end{aligned}$$

   Such that
   a. Lift preserves all basic type formers and has functorial action on $p \circ q$.
   b. $\uparrow$ has an inverse $\downarrow$, preserves all basic term formers and has functorial action on $p \circ q$.
   c. $(\Gamma \triangleright A) = (\Gamma \triangleright \mathsf{Lift}\,p\,A)$, and $\mathsf{Tm}\,\Gamma\,A = \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,p\,A)$ and $\uparrow t = t$.
   Above we mention basic type formers, although we have not yet specified those. The way this should be understood, is that any basic type former introduced from now on should come equipped with preservation equations for lifting. This is similar to how we mandate that any introduced type former must be natural with respect to substitution.
6. A universe structure

$$\mathsf{U} : \{\Gamma : \mathsf{Con}\}(i\,j : \mathsf{Lvl}\,\Gamma) \to i < j \to \mathsf{Ty}\,\Gamma\,j \qquad \mathsf{El} : \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i\,j\,p) \to \mathsf{Ty}\,\Gamma\,i$$

   such that $\mathsf{Lift}\,p\,(\mathsf{U}\,i\,j\,q) = \mathsf{U}\,i\,k\,(p \circ q)$, El has inverse Code, $\mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i\,j\,p) = \mathsf{Ty}\,\Gamma\,i$ and $\mathsf{El}\,t = t$.
7. Basic type formers.

▶ **Definition 19** (Inductive-recursive model of TTDL)**.** Assume an external Lvl structure that supports $l_0, l_1 : \mathsf{Lvl}$ and $l_{01} : l_0 < l_1$ (the bootstrapping assumption). We again use the universe constructions from Section 4.1, instantiated to the assumed Lvl structure. We describe components of the model in order. Again, we write components of the model in **bold** font.

1. The base category remains unchanged from the TTGU model.
2. For the level structure, we define $\mathbf{Lvl}\,\Gamma = \Gamma \to \mathsf{Lvl}$ and $i \lessdot j = (\gamma : \Gamma) \to i\,\gamma < j\,\gamma$. Subsitution for internal levels and morphisms is given by function composition with $\sigma : \Gamma \to \Delta$. Internal composition and $\lessdot$**prop** follow from the external counterparts.

3. The internal bootstrapping assumption is modeled with the external counterpart.
4. We define $\mathbf{Ty}\,\Gamma\,i = (\gamma : \Gamma) \to \mathsf{U}\,(i\,\gamma)$ and $\mathbf{Tm}\,\Gamma\,A = (\gamma : \Gamma) \to \mathsf{El}\,(A\,\gamma)$. Substitution is again function composition, and we have $\Gamma \triangleright A = (\gamma : \Gamma) \times \mathsf{El}\,(A\,\gamma)$.
5. Type lifting is given by $\mathbf{Lift}\,p\,A = \lambda\,\gamma.\,\mathsf{Lift}\,(p\,\gamma)\,(A\,\gamma)$. Similarly as in the TTGU model, $\mathbf{Tm}\,\Gamma\,A = \mathbf{Tm}\,\Gamma\,(\mathbf{Lift}\,p\,A)$ and $\Gamma \triangleright A = \Gamma \triangleright (\mathbf{Lift}\,p\,A)$ follow from the ElLift equality, and term lifting is the identity function.
6. We define $\mathbf{U}\,i\,j\,p = \lambda\,\gamma.\,\mathsf{U}'\,(i\,\gamma)\,(p\,\gamma)$. Again, we have $\mathbf{Tm}\,\Gamma\,(\mathbf{U}\,i\,j\,p) = \mathbf{Ty}\,\Gamma\,i$ by Lemma 3, and $\mathbf{El}$ and $\mathbf{Code}$ are identity functions.
7. Basic type formers are interpreted using U^{IR} codes. Preservation of type and term formers by lifting follows by the definition of Lift and El.

To summarize, the only interesting change compared to the TTGU model is that levels and level morphisms gain potential dependency on contexts. However, in the inductive-recursive model this is simply the addition of an extra semantic function parameter.

5.1 Level Reflection

▶ **Definition 20** (Level reflection rules)**.** Assume that we have definitions for internal levels in the syntax of TTDL, i.e. all of the following are defined:

$$
\begin{aligned}
&\mathsf{Lvl}^I && : \mathsf{Ty}\,\Gamma\,l_0\\
&l_0^I,\,l_1^I && : \mathsf{Tm}\,\Gamma\,\mathsf{Lvl}^I\\
&- <^I - && : \mathsf{Tm}\,\Gamma\,\mathsf{Lvl}^I \to \mathsf{Tm}\,\Gamma\,\mathsf{Lvl}^I \to \mathsf{Ty}\,\Gamma\,l_0\\
&l_{01}^I && : \mathsf{Tm}\,\Gamma\,(l_0^I <^I l_1^I)
\end{aligned}
$$

A *reflection rule* for the above consists of

1. $\mathsf{mk}_{\mathsf{Lvl}} : \mathsf{Tm}\,\Gamma\,\mathsf{Lvl}^I \to \mathsf{Lvl}\,\Gamma$ with its inverse $\mathsf{un}_{\mathsf{Lvl}}$, such that $\mathsf{mk}_{\mathsf{Lvl}}\,l_0^I = l_0$ and $\mathsf{mk}_{\mathsf{Lvl}}\,l_1^I = l_1$.
2. $\mathsf{mk}_{<} : \mathsf{Tm}\,\Gamma\,(i <^I j) \to \mathsf{mk}_{\mathsf{Lvl}}\,i < \mathsf{mk}_{\mathsf{Lvl}}\,i$ with its inverse $\mathsf{un}_{<}$.

For any definition of internal levels, we may extend the specification of TTDL with the corresponding reflection rule, thereby getting an algebraic signature for a type theory with first-class levels (TTFL). We can easily get a TTFL with an inductive-recursive model in the following way. First, we pick an external Lvl structure which a) satisfies the bootstrapping assumption b) has sets of levels and morphisms which can be represented with syntactic TTDL types.

For example, if $\mathsf{Lvl} = (\mathsf{Nat},\, - < -)$, with $l_0 = 0$ and $l_1 = 1$, and TTDL supports natural numbers, then we can define Lvl^I as the internal Nat_{l_0}, and define $- <^I -$ as the usual ordering of numbers, using TTDL type formers and large elimination (which is available from $l_0 < l_1$). Then it follows that the model in Definition 19, instantiated to the current level structure, satisfies level reflection. The model even supports the stricter $\mathsf{Tm}\,\Gamma\,\mathsf{Nat}_{l_0} = \mathsf{Lvl}\,\Gamma$ equation, but in general it is easier to set up models if only an isomorphism is required.

5.2 Universe Features in TTFL

We describe some of the features expressible in TTFL.

Bounded universe polymorphism is realized by quantifying over levels and morphisms with the usual Π types. For example, if levels strictly correspond to internal natural numbers, we may have

$$\mathsf{idUpTo3} : \Pi(l : \mathsf{Nat}_3)(p : \mathsf{Lift}^3_0(l <^I 3))(A : \mathsf{U}\, l\, 3\, (\mathsf{mk}_< p)) \to \mathsf{Lift}\, (\mathsf{mk}_< p)\, A \to \mathsf{Lift}\, (\mathsf{mk}_< p)\, A$$
$$\mathsf{idUpTo3} = \lambda\, l\, p\, A\, a.\, a$$

Here, we make sure that all types are on the same level, by appropriate lifting. We assume that internal levels are in Nat_0, but we can bind an $l : \mathsf{Nat}_3$, because by cumulativity l is also a term of Nat_0. Likewise, the p variable is a term of $\mathsf{Lift}^3_0(l <^I 3)$ and $l <^I 3$ as well.

Transfinite hierarchies are naturally supported. For example, Lvl can be identified with $\mathsf{Maybe}\, \mathsf{Nat}_0$, where $\mathsf{Nothing}$ defines ω and $\mathsf{Just}\, n$ is a finite level. Then, by the definition of morphisms, we have $<\omega : \Pi(n : \mathsf{Nat}_0) \to \mathsf{Just}\, n <^I \omega$. We can use this to quantify over finite levels, as in the following type:

$$\Pi(n : \mathsf{Nat}_\omega)(A : \mathsf{U}\, n\, \omega\, (\mathsf{mk}_< (<\omega\, n))) \to \mathsf{Lift}\, (\mathsf{mk}_< (<\omega\, n))\, A \to \mathsf{Lift}\, (\mathsf{mk}_< (<\omega\, n))\, A$$

This type is in $\mathsf{Ty}\, \Gamma\, \omega$, but it is not in any universe, since ω is the greatest level.

Induction on levels and level morphisms. In Agda 2.6.1, there is an internal type of finite levels, and while construction rules and some built-in operations on levels are exposed, there is no general elimination rule on levels. Thus, there is a $\mathsf{Nat} \to \mathsf{Lvl}$ conversion function but it has no inverse. In contrast, TTFL supports arbitrary elimination on levels and morphisms.

Type formers returning in least upper bounds of levels. It is common in type theories to allow type formers to have parameter types in different universe levels, say i and j, and return in level $i \sqcup j$. In TTFL, whenever levels are *trichotomous*, meaning that the ordering and equality of levels is internally decidable, $i \sqcup j$ can be defined as the greater of i and j, and the "heterogeneous" type formers are derivable[4].

Coercive cumulative subtyping. TTFL as specified does not directly support cumulative subtyping. However, it is compatible with coercive subtyping. Consider the following rules:

$$\begin{aligned}
&- \leq - &&: \mathsf{Ty}\, \Gamma\, i \to \mathsf{Ty}\, \Gamma\, j \to \mathsf{Set}\\
&\mathsf{coerce} &&: A \leq B \to \mathsf{Tm}\, \Gamma\, A \to \mathsf{Tm}\, \Gamma\, B\\
&\leq\mathsf{refl} &&: A \leq A\\
&\mathsf{U}\leq &&: i < i' \to \mathsf{U}\, i\, j\, p \leq \mathsf{U}\, i'\, k\, q\\
&\Pi\leq &&: (p : A' \leq A) \to ((a' : \mathsf{Tm}\, \Gamma\, A') \to B[x \mapsto \mathsf{coerce}\, p\, a'] \leq B'[x \mapsto a'])\\
& &&\quad \to \Pi(x : A)B \leq \Pi(x : A')B'
\end{aligned}$$

Any model of TTFL can support the above rules: we can define $- \leq -$ and coerce by indexed induction-recursion [13], where we define coercion along $\mathsf{U}\leq$ by type lifting, and coercion along $\Pi\leq$ by backwards-forwards coercion. It is possible to extend the subtyping relation with rules for other basic type formers.

Note that Π is contravariant in the domain. This is easily supported with our inductive-recursive semantics, unlike in the set-theoretic model of cumulativity for Coq [26], where function domains are invariant.

[4] A level structure which is trichotonomous and supports *extensionality*, i.e. $(\forall i.\, (i < j) \iff (i < k)) \to j = k$, is a *type-theoretic ordinal*. Assuming excluded middle, type-theoretic ordinals are equivalent to classical ordinals [28, Section 10.3].

5.3 Effects of Choice of Level Structure

TTFL features clearly vary depending on level structures. We make some basic observations.

- We did not mandate that the level of Lvl^I is the least level, i.e. that $l_0 < i$ for every $i \neq l_0$. If this holds, then it is possible to have level polymorphism at every level: at l_0 we can just bind a Lvl^I, and at every other level, we can lift Lvl^I to that level. However, levels are not necessarily totally ordered, and l_0 does not have to be the least. This means that universe polymorphism is prohibited in levels which are not connected to l_0.
- If levels are given by a limit ordinal, then every TTFL type is contained in a universe. If levels form a successor ordinal, then this is not the case. For example, Agda 2.6.1 has $\omega + 1$ levels (externally), where Set_ω is the topmost universe, but Set_ω is not in any universe.
- While it is possible to quantify over all levels (using plain Π types), it is not possible to have level polymorphism over all levels. We may try to type an identity function for all levels, as $\Pi(i : \mathsf{Lift}\,?\,\mathsf{Lvl}^I)(A : \mathsf{U}\,(\mathsf{mk}_{\mathsf{Lvl}}\,i)\,?\,?) \to \mathsf{Lift}\,?\,A \to \mathsf{Lift}\,?\,A$. The issue is in $\mathsf{U}\,(\mathsf{mk}_{\mathsf{Lvl}}\,i)\,?\,?$, where we would have to find a level which is larger than *every* level. The solution to this issue is to simply add more levels. For example, for polymorphism over finite levels, we may pick $\omega + \omega$ as the first limit ordinal which can internalize finite level polymorphism; this is what Agda 2.6.2 does.

6 Related Work

Predicative hierarchies originate from Russell's ramified type theories [29]. In the more modern formulations of type theory, Martin-Löf proposed a countable predicative hierarchy [20], as a way to remedy the inconsistency of the previous version of the theory (which assumed type-in-type). Harper and Pollack described universe inference with level assignments and also a form of level polymorphism [17]. Sterling [24] gave an algebraic specification much like ours for a type theory with countable cumulative universes, and proved canonicity for it.

There have been proposals for strengthening universes with various closure principles and universe operators. Palmgren's super universes and higher-order universes [22] and Setzer's Mahlo universes [23] are examples for this. These are sufficient to model transfinite hierarchies, but as we noted in Section 4.1, we do not know how to model strict inclusions with them. Variants of induction-recursion [12, 13, 14] are particularly flexible and powerful extensions to universes. McBride gave an inductive-recursion definition of cumulative universes that we adapted in this work [21].

It is worth to summarize here the universe features in the current type theory implementations.

Agda 2.6.1 has $\omega + 1$-many non-cumulative predicative universes as Set_i, with optional cumulative subtyping only for universes [9]. It also has an internal type $\mathsf{Level} : \mathsf{Set}_0$ for finite levels (hence, exluding ω), which supports constructors and some built-in operations, but no general elimination rule. There is also a countable parallel hierarchy Prop_i for strict propositions [15]. Agda 2.6.2 will extend the Set_i hierarchy to $\omega * 2$.

Coq 8.13 has ω-many cumulative predicative universes with cumulative subtyping for all type formers [26]. It supports bounded universe polymorphism, but it has no internal type for levels, and universe polymorphic definitions are not internally typeable. It also has an impredicative Prop universe and optionally impredicative bottom Set universe. Version 8.13 added experimental support for a parallel countable cumulative hierarchy for strict propositions.

Lean 3.3 has countable non-cumulative predicative Type_i universes with universe polymorphism, and no internal type of levels [10]. It also has strict impredicative Prop.

Idris 1 has countable cumulative predicative universes with cumulative subtyping only for universes, typical-ambiguity-style level inference and no universe polymorphism [7].

Of the above features, what TTFL does not support is a) impredicativity b) the interaction of Prop and Type universes, i.e. the restrictions on Prop elimination.

7 Conclusion and Future Work

In the current work, we developed a framework for modeling a variety of universe features in type theories. At this point, we may ask the question: if induction-recursion is sufficient to model every feature, why not simply support it in a practical implementation, and drop the menagerie of universe features?

The answer is that induction-recursion provides a *deep embedding* of universe features, which is usually less convenient to use than *native* features. For example, both Coq and Agda have powerful automatic solving for filling out implicit universe levels. We also do not have to invoke El or the $\mathsf{U}_<$ computation rule explicitly, and in Coq we can use implicit syntax for subtyping instead of explicit coercions.

This trade-off between convenience and formal minimalism is similar to the situation with inductive types. Formally, W-types and identity types are easier to handle than general inductive families, but the latter are far more convenient to actually use. Ideally, we would like to justify complicated convenience features by reduction to minimal features. With the current paper, we hope to have made progress in this manner.

Future Work

Several related topics are not discussed in this paper and could be subject to future work.

First, besides consistency, we are often interested in *canonicity*, *normalization* or other metatheoretical properties. The current work focuses on consistency and leaves other properties to future work. We did keep canonicity in mind when specifying the systems in this paper. Hopefully the usual proof method of gluing (in other words, proof-relevant logical predicates) [8, 18, 24] can be adapted to the theories in this paper.

Second, we only focus on using universes as size-based classifiers for types. Stratification features are also present in two-level type theory [2], modal type theories [16] or as h-levels in homotopy type theory [28]. It would be interesting to port universe features in this paper to two-level type theory, as they would hopefully model a form of stage polymorphism in multi-stage compilation. We could try representing Prop universes in TTFL as well. This is closely related to h-level based stratification.

Third, we do not discuss implementation strategies and ergonomics of universe features. Which universe hierarchies support good proof automation? What kind of impact do first-class levels have on elaboration algorithms? Hopefully the current work can aid answering these questions, by at least giving a way to quickly check if some features are logically consistent.

Lastly, we do not handle impredicative universes. The main reason for this is that we do not know the consistency of having induction-recursion and impredicative function space together in the same universe, and modeling impredicativity seems to require this assumption in the metatheory. This could be investigated as well in future work.

References

1 Peter Aczel. An introduction to inductive definitions. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 739–782. Elsevier, 1977.

2 Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *ArXiv e-prints*, May 2019. arXiv:1705.03307.

3 Steve Awodey. Natural models of homotopy type theory. *Math. Struct. Comput. Sci.*, 28(2):241–286, 2018. doi:10.1017/S0960129516000268.

4 Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Math. Struct. Comput. Sci.*, 30(2):118–138, 2020. doi:10.1017/S0960129519000197.

5 Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. Relative induction principles for type theories. *arXiv preprint*, 2021. arXiv:2102.11649.

6 John Cartmell. *Generalised algebraic theories and contextual categories*. PhD thesis, Oxford University, 1978.

7 The Idris Community. Documentation for the idris language, 2021. URL: http://docs.idris-lang.org/en/latest/index.html.

8 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019. doi:10.1016/j.tcs.2019.01.015.

9 Agda developers. Agda documentation, 2021. URL: https://agda.readthedocs.io/en/v2.6.1.3/.

10 Lean developers. Lean reference manual, version 3.3, 2021. URL: https://leanprover.github.io/reference/lean_reference.pdf.

11 Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1995. doi:10.1007/3-540-61780-9_66.

12 Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 1999. doi:10.1007/3-540-48959-2_11.

13 Peter Dybjer and Anton Setzer. Indexed induction-recursion. *J. Log. Algebraic Methods Program.*, 66(1):1–49, 2006. doi:10.1016/j.jlap.2005.07.001.

14 Neil Ghani, Lorenzo Malatesta, and Fredrik Nordvall Forsberg. Positive inductive-recursive definitions. *Log. Methods Comput. Sci.*, 11(1), 2015. doi:10.2168/LMCS-11(1:13)2015.

15 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019. doi:10.1145/3290316.

16 Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 492–506. ACM, 2020. doi:10.1145/3373718.3394736.

17 Robert Harper and Robert Pollack. Type checking with universes. *Theor. Comput. Sci.*, 89(1):107–136, 1991. doi:10.1016/0304-3975(90)90108-T.

18 Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for type theory. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.25.

19 Zhaohui Luo. Notes on universes in type theory. *Lecture notes for a talk at Institute for Advanced Study, Princeton (URL: http://www. cs. rhul. ac. uk/home/zhaohui/universes. pdf)*, page 16, 2012.

20 Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

21 Conor McBride. Datatypes of datatypes, 2015. URL: `http://staff.mmcs.sfedu.ru/~ulysses/Edu/SSGEP/conor/conor.pdf`.

22 Erik Palmgren. On universes in type theory. In *Twenty-five years of constructive type theory*, volume 36 of *Oxford Logic Guides*, pages 191–204. Oxford University Press, 1998.

23 Anton Setzer. Extending martin-löf type theory by one mahlo-universe. *Arch. Math. Log.*, 39(3):155–181, 2000. `doi:10.1007/s001530050140`.

24 Jonathan Sterling. Algebraic type theory and universe hierarchies. *CoRR*, abs/1902.08848, 2019. `arXiv:1902.08848`.

25 Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–15. IEEE, 2021. `doi:10.1109/LICS52264.2021.9470719`.

26 Amin Timany and Matthieu Sozeau. Cumulative inductive types in coq. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.FSCD.2018.29`.

27 Taichi Uemura. A general framework for the semantics of type theory. *CoRR*, abs/1904.04097, 2019. `arXiv:1904.04097`.

28 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

29 A. N. Whitehead and B. Russell. Principia mathematica. *Revue de Métaphysique et de Morale*, 19(2):19–19, 1911.

30 Beta Ziliani and Matthieu Sozeau. A unification algorithm for coq featuring universe polymorphism and overloading. In Kathleen Fisher and John H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 179–191. ACM, 2015. `doi:10.1145/2784731.2784751`.

Succinct Graph Representations of μ-Calculus Formulas

Clemens Kupke
University of Strathclyde, UK

Johannes Marti
University of Amsterdam, The Netherlands

Yde Venema
University of Amsterdam, The Netherlands

Abstract

Many algorithmic results on the modal mu-calculus use representations of formulas such as alternating tree automata or hierarchical equation systems. At closer inspection, these results are not always optimal, since the exact relation between the formula and its representation is not clearly understood. In particular, there has been confusion about the definition of the fundamental notion of the size of a mu-calculus formula.

We propose the notion of a parity formula as a natural way of representing a mu-calculus formula, and as a yardstick for measuring its complexity. We discuss the close connection of this concept with alternating tree automata, hierarchical equation systems and parity games. We show that well-known size measures for mu-calculus formulas correspond to a parity formula representation of the formula using its syntax tree, subformula graph or closure graph, respectively. Building on work by Bruse, Friedmann & Lange we argue that for optimal complexity results one needs to work with the closure graph, and thus define the size of a formula in terms of its Fischer-Ladner closure. As a new observation, we show that the common assumption of a formula being clean, that is, with every variable bound in at most one subformula, incurs an exponential blow-up of the size of the closure.

To realise the optimal upper complexity bound of model checking for all formulas, our main result is to provide a construction of a parity formula that (a) is based on the closure graph of a given formula, (b) preserves the alternation-depth but (c) does not assume the input formula to be clean.

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

Keywords and phrases modal mu-calculus, model checking, alternating tree automata, hierachical equation systems

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.29

Related Version This paper is largely based on our technical report:
Full Version: `https://arxiv.org/abs/2010.14430` [15]

Funding *Clemens Kupke*: Supported by Leverhulme Trust Research Project Grant RPG-2020-232.
Johannes Marti: The research of this author has been made possible by a grant from the Dutch Research Council NWO, project nr. 617.001.857.

1 Introduction

The modal μ-calculus, introduced by Kozen [14] and surveyed in for instance [2, 12, 4, 9], is a logic for describing properties of processes that are modelled by labelled transition systems. It extends the expressive power of propositional modal logic by means of least and greatest fixpoint operators. This addition permits the expression of all bisimulation-invariant monadic second order properties of such processes [13]. As a *logic*, μML has many desirable

properties, such as a natural complete axiomatisation [14, 19], uniform interpolation and other interesting model-theoretical properties [8, 11], and a complete cut-free proof system [1]. Here we will be interested in some of its computational properties.

The μ-calculus is generally regarded as a universal specification language for reactive systems, since it embeds most other logics that are used for this purpose, such as LTL, CTL, CTL* and PDL. Given this status, the computational complexity of its model checking and satisfiability problems is of central importance. While the satisfiability problem has been shown to be EXPTIME-complete [10] already thirty years ago, the precise complexity of its model checking problem turned out to be a challenging problem. A breakthrough was obtained by Calude et alii [7] who gave a *quasi-polynomial* algorithm for deciding parity games; since model checking for the modal μ-calculus can be determined by such games, this indicates a quasi-polynomial upper bound of the complexity of the model checking problem.

Generally, to determine the complexity of a proposed algorithm operating on μ-calculus formulas, one needs sensible measures of the complexity of the formula that is (part of) the input to the algorithm; the most important of these concern *size* and *alternation depth*. Different notions of size have been used, depending on how precisely formulas are represented in the input. Standard size measures include: (1) length, corresponding to a representation of the formula as a string or syntax tree; (2) subformula size, corresponding to a representation of the formula as the directed acyclic graph of its subformulas; and (3) closure size, corresponding to a similar representation of a formula via its (Fischer-Ladner) closure.

The choice between these representations is non-trivial because the subformula size of a formula may be exponentially smaller than its length, and, as was shown by Bruse, Friedmann & Lange [6], its closure size may be exponentially smaller than its subformula size. Consequently, complexity results about the μ-calculus may be suboptimal when expressed in terms of subformula size, in the sense that a stronger version of the result holds when formulated in terms of closure size. In other words, it is desirable to design algorithms that operate on a representation of a formula that is based on its closure.

At closer inspection it turns out that generally, the literature on algorithmic aspects of the μ-calculus is crystal clear in terms of the structures on which the algorithms operate, but less so on the precise way in which these structures represent formulas. As a consequence, when formulated in terms of the actual formulas, complexity results as given may be suboptimal or somewhat fuzzy. Our long-term goal is to study the representation of μ-calculus formulas in more detail, and to develop a framework in which various approaches can easily be compared, and in which complexity results can be formulated and proved optimally and unambiguously.

As a starting point, we note that in the literature different frameworks are used to represent μ-calculus formulas. The parity games that feature in model checking algorithms are usually based on an arena which is some kind of Cartesian product of a graph that represents the formula with the model where this formula is evaluated. Other prominent ways to represent formulas are (alternating) tree automata and (hierarchical) equation systems; as we shall see further on, in these cases we can think of the structures that represent formulas in graph-theoretic terms as well. In all cases then, the mathematically fundamental structure representing a formula is a graph, whose nodes are labelled with logical connectives or atomic formulas, and with priorities that are used to determine some winning or acceptance condition. The graph itself can be based on the syntax tree, the subformula dag or the closure graph of the formula that it represents.

We make this fundamental labelled graph structure explicit and call the resulting concept a *parity formula*.[1] Intuitively, parity formulas generalise standard formulas by dropping the requirement that the underlying graph structure of the formula is a tree with back edges, and adding an explicit parity acceptance condition. A good way to think about a parity formula is as the formula component of a model checking game. As we shall see below, parity formulas are closely related to alternating tree automata and hierarchical equation systems. Compared to these however, parity formulas have a very simple mathematical structure, which allows for a straightforward and unambiguous definition of its size and its index (alternation depth).

The explicit introduction of this notion is not a goal in itself. We intend to *use* it as a tool to analyse some underexposed sides of the theory of the modal μ-calculus. In this paper we discuss some key constructions turning standard formulas into parity formulas and vice versa. Along the way we make two observations that we consider the key contributions of this paper:

1) A common assumption in the literature on the μ-calculus is that one may assume, without loss of generality, that formulas are clean or well-named, in the sense that bound variables are disjoint from free variables, and each bound variable determines a unique subformula. In Proposition 10 we show that this assumption may lead to an exponential blow-up in terms of closure-size. This means that, if one is interested in optimal complexity results, one should not assume the input formula to be clean.

2) To the best of our knowledge, all representations of μ-calculus formulas known from the literature, are suboptimal in one way or another: they are based on the subformula dag, they presuppose cleanness, or they use a priority function which yields an unnecessarily big index. The main result of our paper, Theorem 12, concerns a construction that provides, for every μ-calculus formula, an equivalent parity formula that is based on its closure graph, and has an index that matches its alternation depth. The fact that we do *not* assume the input formula to be clean makes our proof non-trivial.[2]

Because of Proposition 10, Theorem 12 has an impact on the quasi-polynomial time complexity of the model checking problem for the modal μ-calculus. If one wants to formulate an optimal version of this complexity result, by the observations of Bruse, Friedmann & Lange [6] one needs to measure the formula in terms of closure-size; but then Theorem 12 is needed to ensure that the result applies to all formulas, not just to the ones that are clean.

2 Preliminaries

In this section we briefly review the syntax and semantics of the modal μ-calculus.

Syntax. It will be convenient to assume that μ-calculus formulas are in negation normal form. That is, the formulas of the modal μ-calculus $\mu\mathtt{ML}$ are given by the following grammar:

$$\mu\mathtt{ML} \ni \varphi ::= p \mid \overline{p} \mid \bot \mid \top \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid \Diamond\varphi \mid \Box\varphi \mid \mu x\, \varphi \mid \nu x\, \varphi,$$

where p, x are variables, and the formation of the formulas $\mu x\, \varphi$ and $\nu x\, \varphi$ is subject to the constraint that φ is *positive* in x, i.e., there are no occurrences of $\overline{x}$ in φ. Elements of $\mu\mathtt{ML}$ will be called *μ-calculus formulas* or *standard formulas*. Formulas of the form $\mu x.\varphi$ or $\nu x.\varphi$

[1] Parity formulas are almost the same structures as the alternating binary tree automata of Emerson & Jutla [10] and as the version of Wilke's alternating tree automata where the transition conditions are basic formulas, i.e., contain at most one logical connective [20, 12].

[2] Proof details, which we could not include here for lack of space, can be found in the technical report [15].

will be called *fixpoint formulas*. We define $\mathtt{Lit}(\mathsf{Q}) := \{p, \overline{p} \mid p \in \mathsf{Q}\}$ as the set of *literals* over Q, and $\mathtt{At}(\mathsf{Q}) := \{\bot, \top\} \cup \mathtt{Lit}(\mathsf{Q})$ as the set of *atomic formulas* over Q. We will associate μ and ν with the odd and even numbers, respectively, and for $\eta \in \{\mu, \nu\}$ define $\overline{\eta}$ by putting $\overline{\mu} := \nu$ and $\overline{\nu} := \mu$. The notion of *subformula* is defined as usual; we write $\varphi \trianglelefteq \psi$ if φ is a subformula of ψ, and define $\mathit{Sfor}(\psi)$ as the set of subformulas of ψ.

We use standard terminology related to the binding of variables. We write $BV(\xi)$ and $FV(\xi)$ for, respectively, the set of *bound* and *free variables* of a formula ξ. A formula ξ is *tidy*[3] if $FV(\xi) \cap BV(\xi) = \varnothing$. We fix a set Q of proposition letters and let $\mu\mathtt{ML}(\mathsf{Q})$ denote the set of formulas ξ with $FV(\xi) \subseteq \mathsf{Q}$. We let $\varphi[\psi/x]$ denote the formula φ, with every free occurrence of x replaced by the formula ψ; we will make sure that we only apply this substitution operation if ψ is free for x in φ (meaning that no free variable of ψ gets bound after substituting). This saves us from involving alphabetical variants when substituting. The *unfolding* of a formula $\eta x.\chi$ is the formula $\chi[\eta x.\chi/x]$; this formula is tidy if χ is so.

Semantics. The modal μ-calculus is interpreted over Kripke structures. A *(Kripke) model* is a triple $\mathbb{S} = (S, R, V)$ where S is the set of *states* or *points* of $\mathbb{S}$, $R \subseteq S \times S$ is its *accessibility* relation, and $V : \mathsf{Q} \to \mathcal{P}(S)$ its *valuation*. A *pointed model* is a pair $(\mathbb{S}, s)$ where s is a designated state of $\mathbb{S}$. Inductively we define the *meaning* $[\![\varphi]\!]^{\mathbb{S}} \subseteq S$ of a formula $\varphi \in \mu\mathtt{ML}(\mathsf{Q})$ in a model $\mathbb{S}$ as follows:

$$
\begin{array}{lcllcl}
[\![p]\!]^{\mathbb{S}} & := & V(p) & [\![\overline{p}]\!]^{\mathbb{S}} & := & S \setminus V(p) \\
[\![\bot]\!]^{\mathbb{S}} & := & \varnothing & [\![\top]\!]^{\mathbb{S}} & := & S \\
[\![\varphi \vee \psi]\!]^{\mathbb{S}} & := & [\![\varphi]\!]^{\mathbb{S}} \cup [\![\psi]\!]^{\mathbb{S}} & [\![\varphi \wedge \psi]\!]^{\mathbb{S}} & := & [\![\varphi]\!]^{\mathbb{S}} \cap [\![\psi]\!]^{\mathbb{S}} \\
[\![\Diamond\varphi]\!]^{\mathbb{S}} & := & \{s \in S \mid R[s] \cap [\![\varphi]\!]^{\mathbb{S}} \neq \varnothing\} & [\![\Box\varphi]\!]^{\mathbb{S}} & := & \{s \in S \mid R[s] \subseteq [\![\varphi]\!]^{\mathbb{S}}\} \\
[\![\mu x.\varphi]\!]^{\mathbb{S}} & := & \bigcap\{U \subseteq S \mid [\![\varphi]\!]^{\mathbb{S}[x \mapsto U]} \subseteq U\} & [\![\nu x.\varphi]\!]^{\mathbb{S}} & := & \bigcup\{U \subseteq S \mid [\![\varphi]\!]^{\mathbb{S}[x \mapsto U]} \supseteq U\}.
\end{array}
$$

Here $\mathbb{S}[x \mapsto U] := (S, R, V[x \mapsto U]$ where $V[x \mapsto U]$ is the $\mathsf{Q} \cup \{x\}$-valuation mapping x to U and any $p \neq x$ to $V(p)$. If a state $s \in S$ belongs to the set $[\![\varphi]\!]^{\mathbb{S}}$, we write $\mathbb{S}, s \Vdash \varphi$, and say that s *satisfies* φ.

Complexity measures. The size of a formula $\xi \in \mu\mathtt{ML}$ can be measured in at least three different ways. First, its *length* $|\xi|^{\ell}$ is defined as the number of symbols that occur in ξ. Second, we define its *subformula size* $|\xi|^{s} := |\mathit{Sfor}(\xi)|$ as the number of distinct subformulas of ξ.

Third, we can measure the size of ξ by counting the number of formulas in its (Fischer-Ladner) closure. We need some notation and terminology here, where we assume that ξ is tidy. The set $\mathit{Clos}_0(\xi)$ is defined by the following case distinction:

$$
\begin{array}{lcll}
\mathit{Clos}_0(\varphi) & := & \varnothing & \text{if } \varphi \in \mathtt{At}(\mathsf{Q}) \\
\mathit{Clos}_0(\varphi_0 \odot \varphi_1) & := & \{\varphi_0, \varphi_1\} & \text{where } \odot \in \{\wedge, \vee\} \\
\mathit{Clos}_0(\heartsuit\varphi) & := & \{\varphi\} & \text{where } \heartsuit \in \{\Diamond, \Box\} \\
\mathit{Clos}_0(\eta x.\varphi) & := & \{\varphi[\eta x.\varphi/x]\} & \text{where } \eta \in \{\mu, \nu\}.
\end{array}
$$

We write $\xi \to_C \varphi$ if $\varphi \in \mathit{Clos}_0(\xi)$ and call $\to_C$ the *trace* relation on $\mu\mathtt{ML}$. We let $\twoheadrightarrow_C$ denote the reflexive and transitive closure of $\to_C$, and define the *closure* of ξ as the set $\mathit{Clos}(\xi) := \{\varphi \mid \xi \twoheadrightarrow_C \varphi\}$. The *closure graph* of ξ is the directed graph $(\mathit{Clos}(\xi), \to_C)$. The *closure size* $|\xi|^{c}$ of ξ is given as $|\xi|^{c} := |\mathit{Clos}(\xi)|$.

[3] In the literature, some authors make a distinction between proposition letters (which can only occur freely in a formula), and propositional variables, which can be bound. Our tidy formulas correspond to *sentences* in this approach, that is, formulas without free variables.

Next to its size, the most important complexity measure of a μ-calculus formula is its *alternation depth*. We shall work with the definition originating with Niwiński [16]. By natural induction we first define classes $\Theta_n^\mu, \Theta_n^\nu \subseteq \mu\mathtt{ML}$ (corresponding to, respectively, the sets Π_{n+1} and Σ_{n+1} in [16]). Intuitively, Θ_n^η consists of those μ-calculus formulas where n bounds the length of any alternating nesting of fixpoint operators of which the most significant formula is an η-formula. For the definition, we set, for $\eta, \lambda \in \{\mu, \nu\}$:

1. all atomic formulas belong to Θ_0^η;
2. if $\varphi_0, \varphi_1 \in \Theta_n^\eta$, then $\varphi_0 \vee \varphi_1, \varphi_0 \wedge \varphi_1, \Diamond\varphi_0, \Box\varphi_0 \in \Theta_n^\eta$;
3. if $\varphi \in \Theta_n^\eta$ then $\overline{\eta}x.\varphi \in \Theta_n^\eta$ (where we recall that $\overline{\mu} = \nu$ and $\overline{\nu} = \mu$);
4. if $\varphi(x), \psi \in \Theta_n^\eta$, then $\varphi[\psi/x] \in \Theta_n^\eta$, provided that ψ is free for x in φ;
5. all formulas in Θ_n^λ belong to Θ_{n+1}^η.

The *alternation depth* $ad(\xi)$ of a formula ξ is the least n such that $\xi \in \Theta_n^\mu \cap \Theta_n^\nu$. It measures the maximal number of alternations between least and greatest fixpoint operators in ξ.

3 Representations of μ-calculus formulas

In this section we discuss two of the most widely used representations for formulas of the modal μ-calculus that one may find in the literature: alternating tree automata (ATAs) and hierarchical equation systems (HESs). Both of these come in many different shapes, and in some of these shapes the two notions are actually very similar to one another. For lack of space we cannot give a proper survey here, and so we focus on a perspective, in which these similarities come out most clearly.[4] In this perspective, both kinds of representation can be defined using the syntactic notion of a *transition condition*. Recall that we have fixed a set Q of proposition letters; in addition to this we need a set A of objects that we shall call *states* in the setting of ATAs and *variables* in that of HESs. Now consider the following definitions of, respectively, the sets of *basic*, *standard* and *extended* transition conditions over Q and A.

$$\begin{array}{rcl}
\mathrm{BTC}(\mathsf{Q}, A) \ni \beta & ::= & \bot \mid \top \mid p \mid \overline{p} \mid a \mid \Diamond a \mid \Box a \mid a \wedge a \mid a \vee a, \\
\mathrm{STC}(\mathsf{Q}, A) \ni \beta & ::= & \bot \mid \top \mid p \mid \overline{p} \mid a \mid \Diamond a \mid \Box a \mid \beta \wedge \beta \mid \beta \vee \beta, \\
\mathrm{ETC}(\mathsf{Q}, A) \ni \beta & ::= & \bot \mid \top \mid p \mid \overline{p} \mid a \mid \Diamond \beta \mid \Box \beta \mid \beta \wedge \beta \mid \beta \vee \beta,
\end{array}$$

where $p \in \mathsf{Q}$ and $a \in A$.

▶ **Definition 1.** *An* alternating tree automaton *or* ATA *is a quadruple* $\mathbb{A} = (A, \Delta, \Omega, a_I)$ *where A is a non-empty finite set of* states*, of which $a_I \in A$ is the* initial *state, $\Omega : A \to \omega$ is the* priority map*, and $\Delta : A \to \mathrm{STC}(\mathsf{Q}, A)$ is the* transition map*. An* ATA *will be called* basic *if the range of its transition map consists of basic transition conditions.*

Before we move on to the definition of the semantics of ATAs, we make two comments. First and foremost, the ATAs that were introduced by Wilke [20] are in fact what we call *basic*; as we shall see in the next section, these are the ones that are in close correspondence with our parity formulas. In the subsequent literature however, it seems to have become quite common to allow for the more complex conditions that we here call "standard", and that may feature nesting of boolean connectives in transition conditions, (possibly restricted to disjunctive normal form).

Second, if we think of the powerset $\mathcal{P}(\mathsf{Q})$ as an alphabet, then tree-based Kripke models correspond to $\mathcal{P}(\mathsf{Q})$-labelled trees. In such a setting it is common to consider tree automata with a transition map of the form $\Delta : A \times \mathcal{P}(\mathsf{Q}) \to \mathrm{TC}(\emptyset, A)$ for some set of transition

[4] This means in particular that we only consider *amorphous* tree automata here, i.e., we disregard automata operating on trees where the children of a node are given by a bounded number of functions.

conditions in which the proposition letters in Q may not occur. That is, the proposition letters in Q move from the co-domain of the transition map to its domain. It is in fact quite easy to transform automata of the one kind into devices of the other kind, but for lack of space we cannot go into detail here.

The semantics of alternating tree automata is usually given in terms of run trees, but we may also use parity games [12, ch. 9]. A simple version is the acceptance game $\mathcal{A}(\mathbb{A}, \mathbb{S})$ for an ATA $\mathbb{A}$ and a model $\mathbb{S} = (S, R, V)$; it takes positions in the set $V_{\mathbb{A}} \times S$, where $V_{\mathbb{A}}$ is given as

$$V_{\mathbb{A}} := \{a_I\} \cup \bigcup_{a \in A} \mathit{Sfor}(\Delta(a)).$$

For each of these positions Table 1 below lists the set of possible moves and the player that is to move. (We need not assign a player to positions that admit a single move only.) As usual in parity games finite matches are lost by the player who gets stuck (i.e., needs to pick an element from the empty set) and infinite matches are won by $\exists$ iff the maximal priority $\Omega(a)$ of all positions $(a, s) \in A \times S$ that occur infinitely often in the match is even. The starting position is (a_I, s), with $(\mathbb{S}, s)$ the pointed model for which we want to check acceptance.

Table 1 The acceptance game $\mathcal{A}(\mathbb{A}, \mathbb{S})$.

Position	Player	Admissible moves
$(\bot, s)$	$\exists$	$\varnothing$
$(\top, s)$	$\forall$	$\varnothing$
(p, s) for $s \in V(p)$	$\forall$	$\varnothing$
(p, s) for $s \notin V(p)$	$\exists$	$\varnothing$
$(\overline{p}, s)$ for $s \in V(p)$	$\exists$	$\varnothing$
$(\overline{p}, s)$ for $s \notin V(p)$	$\forall$	$\varnothing$
(a, s) for $a \in A$	-	$\{(\Delta(a), s)\}$
$(\alpha_0 \vee \alpha_1, s)$	$\exists$	$\{(\alpha_0, s), (\alpha_1, s)\}$
$(\alpha_0 \wedge \alpha_1, s)$	$\forall$	$\{(\alpha_0, s), (\alpha_1, s)\}$
$(\Diamond a, s)$	$\exists$	$\{(a, t) \mid sRt\}$
$(\Box a, s)$	$\forall$	$\{(a, t) \mid sRt\}$

As a second way of representing μ-calculus formulas we now discuss *hierarchical equation systems* [18, 6]. As with alternating tree automata there are multiple definitions of hierarchical equation systems in the literature. Here we recall the definition from [9] (where they are called *modal* equation systems).

▶ **Definition 2.** *A* hierarchical equation system *or* HES *consists of a finite set of variables* $A = \{X_1, \ldots, X_n\}$*, together with a set*

$$\mathcal{E} = \{X_1 =_{p_1} \beta_1, \ldots, X_n =_{p_n} \beta_n\}.$$

of prioritised modal equations. That is, for each i*, the number* $p_i \in \omega$ *denotes the* priority *of the* i*-th equation, and* β_i *is an expression in the set* $\mathrm{ETC}(\mathsf{Q}, A)$.

By convention the first variable X_1 is the entry point of the equation system, which functions similarly to the initial state of an ATA. In [18, 6] the semantics of hierarchical equation systems is defined on the basis of the Knaster-Tarski fixpoint theorem, as in the compositional semantics of standard formulas defined in Section 2. It is however also possible to give a semantics in terms of parity games, completely analogous to the game semantics for ATAs mentioned above. We leave the details to the reader.

It is clear that there is a close correspondence between hierarchical equation systems and alternating tree automata. In fact one might view an HES as a generalised version of an ATA in which modalities can be nested inside of the transition conditions – such a generalised notion of ATA has been used for example in [5]. With this in mind, in the sequel we will take this generalised perspective on ATAs, so that we include HESs when we refer to ATAs.

It is not entirely obvious what is the right measure for the size of an alternating tree automaton $\mathbb{A} = (A, \Delta, \Omega, a_I)$. One might simply consider the number of states in $\mathbb{A}$, but since any actual representation of the automaton needs to encode the arbitrarily large transition conditions a more adequate measure of the size of $\mathbb{A}$ should take these into account as well. Moreover, since the acceptance game $\mathcal{A}(\mathbb{A}, \mathbb{S})$ is based on the set $V_{\mathbb{A}} \times S$, it makes sense to define $|\mathbb{A}| := |V_{\mathbb{A}}|$, but also, to consider a representation of $\mathbb{A}$ that is more directly based on this set $V_{\mathbb{A}}$. This is what we will do in the next section.

4 Parity formulas

As the backbone of our framework we introduce the notion of a parity formula. These are like ordinary (modal) formulas, with the difference that (i) the underlying structure of a parity formula is a directed graph, possibly with cycles, rather than a tree; and (ii) one adds a priority labelling to this syntax graph, to ensure a well-defined game-theoretical semantics in terms of parity games.

▶ **Definition 3.** *A* parity formula over Q *is a quintuple* $\mathbb{G} = (V, E, L, \Omega, v_I)$*, where*

- (V, E) *is a finite, directed graph, with* $|E[v]| \leq 2$ *for every vertex* v*;*
- $L : V \to \mathtt{At}(\mathsf{Q}) \cup \{\wedge, \vee, \Diamond, \Box, \epsilon\}$ *is a labelling function;*
- $\Omega : V \stackrel{\circ}{\to} \omega$ *is a partial map, the* priority *map of* $\mathbb{G}$*; and*
- v_I *is a vertex in* V*, referred to as the* initial *node of* $\mathbb{G}$*;*

such that (with $E[v] := \{u \in V \mid Evu\}$*):*

1. $|E[v]| = 0$ *if* $L(v) \in \mathtt{At}(\mathsf{Q})$*, and* $|E[v]| = 1$ *if* $L(v) \in \{\Diamond, \Box\} \cup \{\epsilon\}$*;*
2. *every cycle of* (V, E) *contains at least one node in* $\mathsf{Dom}(\Omega)$*.*

A node $v \in V$ *is called* silent *if* $L(v) = \epsilon$*,* constant *if* $L(v) \in \{\top, \bot\}$*,* literal *if* $L(v) \in \mathtt{Lit}(\mathsf{Q})$*,* atomic *if it is either constant or literal,* boolean *if* $L(v) \in \{\wedge, \vee\}$*, and* modal *if* $L(v) \in \{\Diamond, \Box\}$*. The elements of* $\mathsf{Dom}(\Omega)$ *will be called* states*.*

The semantics of parity formulas is given in terms of a *model checking game*, which is defined as the following parity game between $\exists$ and $\forall$.

▶ **Definition 4.** *Let* $\mathbb{S} = (S, R, V)$ *be a model, and let* $\mathbb{G} = (V, E, L, \Omega, v_I)$ *be a parity formula. We define the* model checking game $\mathcal{E}(\mathbb{G}, \mathbb{S})$ *as the parity game* (G, E, Ω') *of which the board (or arena) consists of the set* $V \times S$*, the priority map* $\Omega' : V \times S \to \omega$ *is given by putting* $\Omega'(v, s) := \Omega(v)$ *if* $v \in \mathsf{Dom}(\Omega)$ *and* $\Omega'(v, s) := 0$ *otherwise. and the game graph is given in Table 2.* $\mathbb{G}$ holds at *or* is satisfied by *the pointed model* $(\mathbb{S}, s)$*, notation:* $\mathbb{S}, s \Vdash \mathbb{G}$*, if the pair* (v_I, s) *is a winning position for* $\exists$ *in* $\mathcal{E}(\mathbb{G}, \mathbb{S})$*.*

Equivalence of parity formulas, and between parity formulas and standard formulas (or ATAs or HESs), is defined in the obvious way.

▶ **Example 5.** Figure 1 to the right displays an example of a parity formula that is based on the standard μ-calculus formula $\xi = \mu x.(\overline{p} \vee \Diamond x) \vee \nu y.(q \wedge \Box(x \vee y))$, by adding *back edges* to the subformula dag of ξ. Nodes in the domain of the priority map are indicated by the notation $\cdot|n$, where n is the priority. The initial node is $\epsilon|1$.

Table 2 The model checking game $\mathcal{E}(\mathbb{G}, \mathbb{S})$.

Position		Player	Admissible moves
(v, s)	with $L(v) = p$ and $s \in V(p)$	$\forall$	$\varnothing$
(v, s)	with $L(v) = p$ and $s \notin V(p)$	$\exists$	$\varnothing$
(v, s)	with $L(v) = \overline{p}$ and $s \in V(p)$	$\exists$	$\varnothing$
(v, s)	with $L(v) = \overline{p}$ and $s \notin V(p)$	$\forall$	$\varnothing$
(v, s)	with $L(v) = \epsilon$	-	$E[v] \times \{s\}$
(v, s)	with $L(v) = \vee$	$\exists$	$E[v] \times \{s\}$
(v, s)	with $L(v) = \wedge$	$\forall$	$E[v] \times \{s\}$
(v, s)	with $L(v) = \Diamond$	$\exists$	$E[v] \times R[s]$
(v, s)	with $L(v) = \Box$	$\forall$	$E[v] \times R[s]$

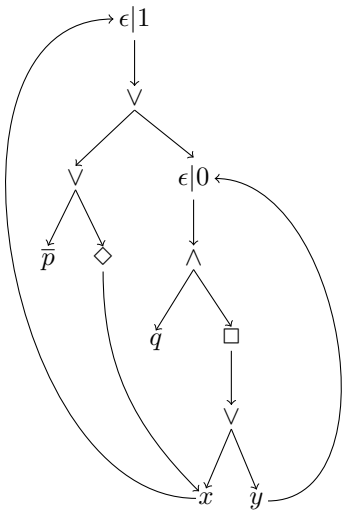


Figure 1 Example of a parity formula.

▶ **Example 6.** One can also build a parity formula from the closure graph of some standard μ-calculus formula. As an example we consider the formula ξ_2 from our proof of Proposition 10 in Section 5:

$$\xi_2 := \mu x_0.\gamma_2 \wedge (\gamma_1 \wedge x_0),$$

where

$$\gamma_1 := \mu x_1.x_1 \wedge (\mu x_0.\gamma_2 \wedge x_1 \wedge x_0), \text{ and}$$
$$\gamma_2 := \mu x_2.x_2 \wedge \big((\mu x_1.x_1 \wedge (\mu x_0.x_2 \wedge x_1 \wedge x_0)) \wedge (\mu x_0.x_2 \wedge (\mu x_1.x_1 \wedge (\mu x_0.x_2 \wedge x_1 \wedge x_0)) \wedge x_0)\big).$$

A picture of the closure graph $(\mathit{Clos}(\xi_2), \rightarrow_C)$ of ξ_2 is on the left in Figure 2 below (where γ_2 is represented by γ_0). This closure graph gives rise to a parity formula whose vertices are the elements of $\mathit{Clos}(\xi_2)$ and edges are given by the trace relation $\rightarrow_C$. The labelling is obvious and the initial node is the node $\xi_2 = \gamma_0$. The priority map Ω can be defined such that $\Omega(\gamma_0) = \Omega(\gamma_1) = \Omega(\gamma_2) = 1$ and Ω is undefined on all other vertices.

We impose a bound on the outdegree of vertices in parity formulas, so that the size of any reasonable encoding of a parity formula is linear in the number of vertices. This facilitates the following simple definition of size:

▶ **Definition 7.** *The* size *of a parity formula* $\mathbb{G} = (V, E, L, \Omega, v_I)$ *is defined as its number of nodes:* $|\mathbb{G}| := |V|$.

The second fundamental complexity measure for a parity formula is its index, which corresponds to the alternation depth of standard formulas. The most straightforward definition of this notion would be to take the size of the range of the priority map; a slightly more sophisticated approach[5] involves the notions of an *alternating* Ω*-chain* and of a *cluster* (or maximal strongly connected component) of $\mathbb{G}$

▶ **Definition 8.** *Let* $\mathbb{G} = (V, E, L, \Omega, v_I)$ *be a parity formula.*

A set $C \subseteq V$ *is a* cluster *in* $\mathbb{G}$ *if* C *is a maximal set such that* E^*uv *and* E^*vu *for all* $u, v \in C$*. Clusters are partially ordered by placing one cluster* C higher *than another cluster* C' *if* E^*uu' *for all* $u \in C$ *and* $u' \in C'$*. A cluster* C *in* $\mathbb{G}$ *is* degenerate *if* $C = \{v\}$ *is a singleton and we do not have* Evv*; otherwise,* C *is called* nondegenerate*.*

An alternating Ω-chain *of* length k *in* $\mathbb{G}$ *is a finite sequence* $v_1 \cdots v_k$ *of states that all belong to the same cluster, and satisfy, for all* $i < k$*, that* $\Omega(v_i) < \Omega(v_{i+1})$ *while* v_i *and* v_{i+1} *have different parity. Such a chain is called an* μ*-chain (*ν*-chain) if* $\Omega(v_k)$ *is odd (even, respectively). Given a cluster* C *of* $\mathbb{G}$ *and* $\eta \in \{\mu, \nu\}$ *we define* $ind_\eta(C)$*, the* η*-index of* C*, as the maximal length of an alternating* η*-chain in* C*, and the* index *of* C *as* $ind_{\mathbb{G}}(C) := \max\big(ind_\mu(C), ind_\nu(C)\big)$*. Finally, we define*

$$ind(\mathbb{G}) := \max\{ind_{\mathbb{G}}(C) \mid C \in Clus(\mathbb{G})\}.$$

Note that if $\mathbb{G}$ has cycles then $\mathsf{Dom}(\Omega) \neq \varnothing$, so that $\mathbb{G}$ has alternating chains. If $\mathbb{G}$ is cycle-free then we can assume that $\mathsf{Dom}(\Omega)$ is empty, in which case $ind(\mathbb{G}) = 0$.

Parity formulas, alternating tree automata and hierarchical equation systems

It should be clear from the definitions that parity formulas are *very* similar to both alternating tree automata and hierarchical equation systems. To transform a given ATA $\mathbb{A} = (A, \Delta, \Omega, a_I)$ into an equivalent parity formula $\mathbb{G}_{\mathbb{A}} = (V, E, L, \Omega', v_I)$, one just builds a graph on the set $V_{\mathbb{A}}$ in the obvious way, and defines $\Omega' := \Omega$ (with the understanding that Ω' is now a *partial* map on V), and $v_I := a_I$. Finally, one defines $L(a) := \epsilon$ if $a \in A$, whereas $L(\alpha)$ for $\alpha \in \mathrm{STC}(\mathsf{Q}, A) \setminus A$ is given as $L(\alpha) := \alpha$ in case α is atomic, and $L(\alpha)$ is the main connective of α otherwise. It is then straightforward to show that $\mathbb{A} \equiv \mathbb{G}_{\mathbb{A}}$, whereas $\mathbb{G}_{\mathbb{A}}$ obviously has the same size as $\mathbb{A}$. In the opposite direction, it is as straightforward to define, for an arbitrary parity formula $\mathbb{G}$, an equivalent basic ATA $\mathbb{A}$ of the same size and index.

Parity formulas, then, can be seen as a definitional variation of ATAs or HESs. We prefer the graph-based format of parity formulas, since this shows more clearly how to generalise standard formulas, and allows for very perspicuous definitions of complexity measures. What matters most, however, is that the results that we prove in the next two sections apply to ATAs and HESs, in the same way as to parity formulas, see for instance Remark 11 where we make this point explicit.

5 Size issues

It follows from our observations in the previous paragraphs that we may solve the model checking problem for the modal μ-calculus by transforming an arbitrary formula $\xi \in \mu\mathtt{ML}$ into an equivalent parity formula $\mathbb{G}$, and then use the model checking game for parity formulas,

[5] Note that these two definitions almost coincide, since we may shift the priorities of any cluster to either $0, \ldots, d$ or $1, \ldots, d+1$.

together with an algorithm for solving parity games.[6] While the complexity of solving parity games is still not exactly understood, there is no doubt that the key parameters that determine this complexity are the size and the index of the game. Thus, given the definition of the model checking game for parity formulas, it is of crucial importance to find, for an arbitrary μ-calculus formula ξ, an equivalent parity formula $\mathbb{G}$ of minimal size and index. While Kozen [14] already showed that the closure set $\mathit{Clos}(\xi)$ of a clean μ-calculus formula ξ never exceeds the number of subformulas of ξ, Bruse, Friedmann & Lange [6] revealed that $\mathit{Clos}(\xi)$ can in fact be exponentially smaller than $\mathit{Sfor}(\xi)$ of its subformulas. This difference in size indicates that for optimal complexity results, rather than building a parity formula for ξ on the set $\mathit{Sfor}(\xi)$, one should work with the closure graph of ξ.

In the next section we will give a concrete definition of such a parity formula. Here we point out a complication in this definition that seems to have gone unnoticed until now; it concerns the notion of a formula being *clean* or *well-named*.

▶ **Definition 9.** *A tidy μ-calculus formula ξ is* clean *or* well-named *if we may associate with each $x \in BV(\xi)$ a unique subformula of the form $\eta x.\delta$. This unique subformula will be denoted as $\eta_x x.\delta_x$, and we call x a μ-variable if $\eta_x = \mu$, and a ν-variable if $\eta_x = \nu$.*

It is generally very convenient to work with clean formulas, since the bound variables of a clean formula are in 1-1 correspondence with its fixpoint subformulas.[7] For this reason one often sees in the literature that authors assume that the formulas they work with are clean. It is easy to rewrite a μ-calculus formula into an equivalent clean variant, by a suitable renaming of bound variables. The problem, however, is that such a renaming comes at a high cost, as is stated by the following proposition.

▶ **Proposition 10.** *There exists a family $\xi_1, \xi_2, \ldots$ of formulas in $\mu\mathtt{ML}$ such that $|\xi_n|^c \leq 2n+2$, but $|\psi_n|^c \geq 2^n$ for every clean alphabetic variant ψ_n of ξ_n.*

Proof. Fix a number n. The formula ξ_n is defined in terms of simpler families of formulas β_i, γ_i for all $i \in \{0, \ldots, n\}$ and $\alpha_{i,j}$ for all $i, j \in \{0, \ldots, n\}$ with $j \leq i$. First we define β_i by an induction on $i \leq n$:

$\beta_0 := \mu x_0.x_n \wedge \cdots \wedge x_0$

$\beta_i := \mu x_i.\alpha_{i,i} \wedge \cdots \wedge \alpha_{i,0}$,

where $\alpha_{i,j}$ for $j \leq i$ is defined by an inner downwards induction such that $\alpha_{i,i} := x_i$ and for all j with $0 \leq j < i$ we set

$\alpha_{i,j} := \beta_j[\alpha_{i,i}/x_i] \cdots [\alpha_{i,j+1}/x_{j+1}]$.

Note that $FV(\beta_i) \subseteq \{x_n, \ldots, x_{i+1}\}$ and $FV(\alpha_{i,j}) \subseteq \{x_n, \ldots, x_i\}$ for all $j \leq i$. In the definition of β_i and the remainder of this section we assume that conjunction associates to the right. We then define γ_i with a downwards induction on i such that

$$\gamma_i := \beta_i[\gamma_n/x_n] \cdots [\gamma_{i+1}/x_{i+1}].$$

Finally, we set $\xi_n := \gamma_0$. Figure 2 depicts the closure graphs for ξ_2 and ξ_3. The formula ξ_2 is given in Example 6. The formula ξ_3 is already too large to be written out.

---

[6] Because the correspondence between parity formulas and ATAs and HESs, this is the standard way of approaching model checking for $\mu\mathtt{ML}$.

[7] In some situations it is even necessary to work with clean formulas. Suppose, for instance, that for a formula $\xi \in \mu\mathtt{ML}$ one wants to base an equivalent ATA $\mathbb{A}_\xi$ on the set of *subformulas* of ξ. If we cannot associate a unique subformula of ξ with some bound variable x of ξ, then there is no sensible way to define the value of the transition map for this x.

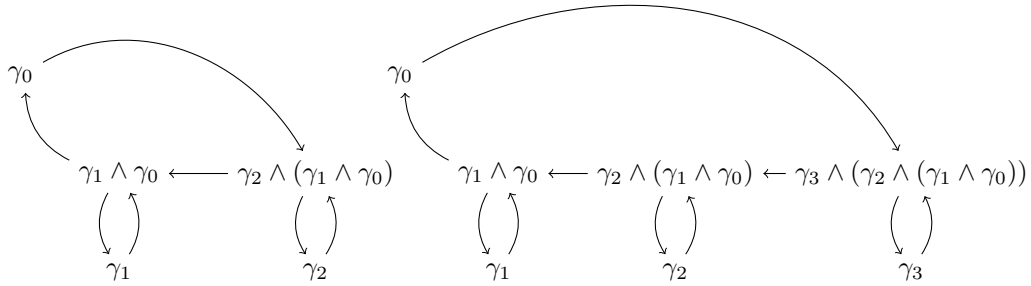


Figure 2 Structure of the closure graphs for ξ_2 (represented by γ_0 in the left graph) and for ξ_3 (represented by γ_0 in the right graph).

To show that $|\xi_n|^c \leq 2n+2$ one needs to verify that

$$Clos(\xi_n) = \{\gamma_0, \ldots, \gamma_n, \gamma_1 \wedge \gamma_0, \gamma_2 \wedge (\gamma_1 \wedge \gamma_0), \ldots, \gamma_n \wedge \cdots \wedge \gamma_0\}.$$

The crucial observation behind this result is that for all $j \leq i$ it holds that

$$\alpha_{i,j}[\gamma_n/x_n] \cdots [\gamma_{i+1}/x_{i+1}][\gamma_i/x_i] = \gamma_j.$$

This equation can be proved by a downward induction over $j \in \{i, \ldots, 0\}$ for every fixed i.

To prove the result on the closure size of clean renamings of ξ_n we use the notion of *fixpoint depth*. Inductively we define $\mathsf{fd}(\varphi) := 0$ if φ is atomic, $\mathsf{fd}(\varphi_0 \odot \varphi_1) := \max(\mathsf{fd}(\varphi_0), \mathsf{fd}(\varphi_1))$, $\mathsf{fd}(\heartsuit\varphi) := \mathsf{fd}(\varphi)$, and $\mathsf{fd}(\eta x.\varphi) := 1 + \mathsf{fd}(\varphi)$. As we sketch below one can then show that

$$\mathsf{fd}(\xi_n) \geq 2^n. \tag{1}$$

To see how the claim about clean alphabetic variants follows from (1) let ψ_n be some clean alphabetical variant of ξ_n; it is not hard to see that we have $\mathsf{fd}(\psi_n) \geq 2^n$ as well. The claim then follows by the observation that

$$\text{every clean } \mu\text{-calculus formula } \chi \text{ satisfies } |\chi|^c \geq \mathsf{fd}(\chi). \tag{2}$$

For a proof of this statement, first observe that for any subformula $\eta x.\varphi \trianglelefteq \chi$, the closure of χ contains a formula of the form $\eta x.\varphi'$. This implies that $|\chi|^c = |Clos(\chi)| \geq |BV(\chi)|$. But if χ is a formula of fixpoint depth k, then there is a chain of subformulas $\eta_1 x_1.\varphi_1 \trianglelefteq \eta_2 x_2.\varphi_2 \trianglelefteq \cdots \trianglelefteq \eta_k x_k.\varphi_k$, and if χ is *clean*, then all these variables x_i must be distinct. This shows that $|BV(\chi)| \geq \mathsf{fd}(\chi)$. Combining these observations, we see that $|\chi|^c \geq \mathsf{fd}(\chi)$ indeed.

To prove (1) we need the auxiliary notion of the fixpoint depth of a variable in a formula. Given a formula φ and variable x, we let $\mathsf{fd}(x, \varphi)$, the *fixpoint depth of x in φ*, denote the maximum number of fixpoint operators that one may meet on a path from the root of the syntax tree of φ to a free occurrence of x in φ, with $\mathsf{fd}(x, \varphi) = -\infty$ if no such occurrence exists. Formally, we set $\mathsf{fd}(x, x) := 0$, $\mathsf{fd}(x, y) := -\infty$ if $x \neq y$, $\mathsf{fd}(x, \varphi_0 \odot \varphi_1) := \max\big(\mathsf{fd}(x, \varphi_0), \mathsf{fd}(x, \varphi_1)\big)$, $\mathsf{fd}(x, \heartsuit\varphi) := \mathsf{fd}(x, \varphi)$, $\mathsf{fd}(x, \eta x.\varphi) := -\infty$, and $\mathsf{fd}(x, \eta y.\varphi) = 1 + \mathsf{fd}(x, \varphi)$ if $x \neq y$. Without proof we mention that, provided $x \neq y$ and ψ is free for y in φ:

$$\mathsf{fd}(x, \varphi[\psi/y]) = \max\big(\mathsf{fd}(x, \varphi), \mathsf{fd}(y, \varphi) + \mathsf{fd}(x, \psi)\big).$$

From this we immediately infer that

$$\mathsf{fd}(x, \varphi[\psi/y]) \geq \mathsf{fd}(y, \varphi) + \mathsf{fd}(x, \psi), \tag{3}$$

which shows that every substitution doubles the fixpoint depth of a variable and leads to the exponential bound in (1). More concretely one can show that for all k and i such that $k > i$ it holds that

$$\mathsf{fd}(x_k, \beta_i) \geq 2^i \tag{4}$$

From this (1) follows because β_n is a subformula of ξ_n. The statement (4) is shown by an induction over i, where in the inductive step one proves with an inner induction over $j \in \{i-1, \ldots, 0\}$ that $\mathsf{fd}(x_k, \alpha_{i,j}) \geq 2^{i-1} + \cdots + 2^j$. We leave the details to the reader. ◀

6 Standard formulas and parity formulas

In this section we show how to move back and forth between standard μ-calculus formulas and parity formulas, in such a way that the closure-size of the standard formula corresponds linearly to the size of the parity formula and the alternation depth is preserved.

From standard formulas to parity formulas

Our main theorem states that for an arbitrary tidy formula, we can find an equivalent parity formula that is based on the formula's closure graph, and has an index which is bounded by the alternation depth of the formula.

▶ Remark 11. To stress our point that our results apply to ATAs and HESs too, suppose that we want to base an ATA $\mathbb{A}_\xi$ on the closure set of a formula ξ, or, for the sake of a perspicuous definition, on the set $A := \{\widehat{\varphi} \mid \varphi \in \mathit{Clos}(\xi)\}$. It is clear how to define the transition map Δ: we simply put $\Delta(\widehat{\varphi}) := \varphi$ if φ is atomic, $\Delta(\widehat{\varphi \odot \psi}) := \widehat{\varphi} \odot \widehat{\psi}$ (for $\odot \in \{\wedge, \vee\}$), $\Delta(\widehat{\heartsuit\varphi}) := \heartsuit\widehat{\varphi}$ (for $\heartsuit \in \{\Diamond, \Box\}$), and $\Delta(\widehat{\eta x.\varphi}) := \varphi[\widehat{\eta x.\varphi}/x]$ (for $\eta \in \{\mu, \nu\}$). What is *not* obvious, however, is how to define the priority map on the set A (unless ξ is clean); this is exactly the issue we address here.

▶ **Theorem 12.** *There is a construction transforming an arbitrary tidy formula $\xi \in \mu\mathtt{ML}$ into an equivalent parity formula $\mathbb{G}_\xi$, which is based on the closure graph of ξ, so that $|\mathbb{G}_\xi| = |\xi|^c$ and $\mathit{ind}(\mathbb{G}_\xi) \leq \mathit{ad}(\xi)$.*

The formula $\mathbb{G}_\xi = (V, E, L, \Omega, v_I)$ is defined such that (V, E) is the closure graph of ξ, $v_I = \xi$ and L is the labelling that maps a literal to itself, a boolean or modal formula to its main connective and a fixpoint formula to ϵ. Clearly this guarantees $|\mathbb{G}_\xi| = |\xi|^c$. The main difficulty is in defining the priority map Ω on $\mathit{Clos}(\xi)$ such that $\mathbb{G}_\xi$ is equivalent to ξ and $\mathit{ind}(\mathbb{G}_\xi) \leq \mathit{ad}(\xi)$, *without assuming that ξ is clean.*

The definition of Ω is such that it assigns priorities to the fixpoint formulas in the closure of ξ. Because every cycle in the trace relation needs to pass over at least one fixpoint formula this makes sure that condition 2) of Definition 3 is satisfied by $\mathbb{G}_\xi$. In fact we can take Ω to be the restriction of a global priority map Ω_g, which uniformly assigns a priority to every tidy fixpoint formula in $\mu\mathtt{ML}$. The function Ω_g itself is defined cluster-wise from a strict partial ordering $\sqsubset_C$ over the set of all tidy fixpoint formulas. To define $\sqsubset_C$ we make use of the following notion of a *free* subformula.

▶ **Definition 13.** *Let φ and ψ be μ-calculus formulas. We say that φ is a* free *subformula of ψ, notation: $\varphi \trianglelefteq_f \psi$, if $\psi = \psi'[\varphi/x]$ for some formula ψ' such that $x \in FV(\psi')$ and φ is free for x in ψ'.*

The following is a useful characterisation of the free subformula relation (see [15] for a proof):

$\varphi \trianglelefteq_f \psi$ iff $\varphi \in \mathit{Sfor}(\psi) \cap \mathit{Clos}(\psi)$.

▶ **Definition 14.** *We let $\equiv_C$ denote the equivalence relation generated by the relation $\to_C$, in the sense that: $\varphi \equiv_C \psi$ if $\varphi \twoheadrightarrow_C \psi$ and $\psi \twoheadrightarrow_C \varphi$. We will refer to the equivalence classes of $\equiv_C$ as* (closure) clusters*, and denote the cluster of a formula φ as $C(\varphi)$.*

We define the closure priority relation $\sqsubseteq_C$ on fixpoint formulas by putting $\varphi \sqsubseteq_C \psi$ precisely if $\psi \twoheadrightarrow_C^{\psi} \varphi$, where $\twoheadrightarrow_C^{\psi}$ is the relation given by $\rho \twoheadrightarrow_C^{\psi} \sigma$ if there is a trace $\rho = \chi_0 \to_C \chi_1 \to_C \cdots \to_C \chi_n = \sigma$ such that $\psi \trianglelefteq_f \chi_i$, for every $i \in [0,..,n]$. We write $\varphi \sqsubset_C \psi$ if $\varphi \sqsubseteq_C \psi$ and $\psi \not\sqsubseteq_C \varphi$.

Using $\sqsubset_C$ we can define the priority of a fixpoint formula as follows:

▶ **Definition 15.** *An* alternating $\sqsubset_C$-chain *of length n is a sequence $(\eta_i x_i.\chi_i)_{i\in[1,..,n]}$ of tidy fixpoint formulas such that $\eta_i x_i.\chi_i \sqsubset_C \eta_{i+1}x_{i+1}.\chi_{i+1}$ and $\eta_{i+1} = \overline{\eta_i}$ for all $i \in [0,..,n-1]$. We say that such a chain* starts at $\eta_1 x_1.\chi_1$ *and* leads up to $\eta_n x_n.\chi_n$.

Given a tidy fixpoint formula ξ, we let $h^{\uparrow}(\xi)$ and $h^{\downarrow}(\xi)$ denote the maximal length of any alternating $\sqsubset_C$-chain starting at, respectively leading up to, ξ. Given a closure cluster D, we let $cd(D)$ denote the maximal length of an alternating $\sqsubset_C$-chain in D.

The global priority function $\Omega_g : \mu\mathtt{ML}^t \to \omega$ *is defined cluster-wise, as follows. Take an arbitrary tidy fixpoint formula $\eta y.\varphi$, and define*

$$\Omega_g(\eta y.\varphi) := \begin{cases} cd(C(\psi)) - h^{\uparrow}(\psi)) & \text{if } cd(C(\psi) - h^{\uparrow}(\psi)) \text{ has parity } \eta \\ \big(cd(C(\psi)) - h^{\uparrow}(\psi)\big) + 1 & \text{if } cd(C(\psi)) - h^{\uparrow}(\psi)) \text{ has parity } \overline{\eta}, \end{cases}$$

where we recall that we associate μ and ν with odd and even parity, respectively. If ψ is not of the form $\eta y.\varphi$, we leave $\Omega_g(\psi)$ undefined.

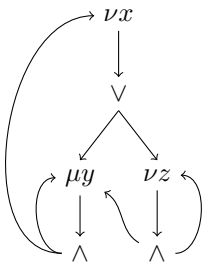
Finally we define the priority function Ω of the parity formula $\mathbb{G}_\xi$ to be $\Omega := \Omega_g \restriction_{Clos(\xi)}$.

▶ Remark 16. The definition of the priority map Ω_g and of the priority order $\sqsubset_C$ on which it is based, may look overly complicated. In fact, simpler definitions would suffice if we are only after the equivalence of ξ with $\mathbb{G}_\xi$ and we do not need an exact match of index and alternation depth.

In particular, we could have introduced an alternative priority order $\sqsubset'_C$ by putting $\varphi \sqsubset'_C \psi$ if $\varphi \equiv_C \psi$ and $\psi \lhd_f \varphi$. This definition of $\sqsubset'_C$ is similar to the definition of a valid thread in [3]. If we would base a priority map Ω'_g on $\sqsubset'_C$ instead of on $\sqsubset_C$, then we could prove the equivalence of any tidy formula ξ with the associated parity formula $\mathbb{G}'_\xi$ that is just like $\mathbb{G}$ but uses Ω'_g as its priority map. However, we would not be able to prove that the index of $\mathbb{G}'_\xi$ is bounded by the alternation depth of ξ.
To see this, consider the following formula:

$$\alpha_x := \nu x.\big((\mu y.x \wedge y) \vee \nu z.(z \wedge \mu y.x \wedge y)\big).$$

We leave it for the reader to verify that this formula has alternation depth two, and that its closure graph looks as in the picture to the right (where we only indicate the main connective of the formulas):



Let α_y and α_z be the other two fixpoint formulas in the cluster of α_x, that is, let $\alpha_y := \mu y.\alpha_x \wedge y$ and $\alpha_z := \nu z.z \wedge \alpha_y$. These formulas correspond to the nodes in the graph that are labelled μy and νz, respectively. Now observe that we have $\alpha_x \lhd_f \alpha_y \lhd_f \alpha_z$, so that this cluster has an alternating $\sqsubset'_C$-chain of length *three*: $\alpha_z \sqsubset'_C \alpha_y \sqsubset'_C \alpha_x$. Note however, that any trace from α_y to α_z must pass through α_x, the $\sqsubset_C$-maximal element of the cluster. In particular, we do *not* have $\alpha_z \sqsubset_C \alpha_y$, so that there is *no* $\sqsubset_C$-chain of length three in the cluster.

A different kind of simplification of the global priority map would be to define

$$\Omega''_g(\psi) := \begin{cases} h^{\downarrow}(\psi) & \text{if } h^{\downarrow}(\psi) \text{ has parity } \eta \\ h^{\downarrow}(\psi) - 1 & \text{if } h^{\downarrow}(\psi) \text{ has parity } \overline{\eta}. \end{cases} \tag{5}$$

Using this definition for a priority map Ω''_g, we would again obtain the equivalence of ξ and the resulting parity formula $\mathbb{G}''_\xi := (\mathbb{C}_\xi, \Omega''_g \restriction_{Clos(\xi)})$. In addition, we would achieve that the index of the parity formula $\mathbb{G}''_\xi$ satisfies $ind(\mathbb{G}''_\xi) \leq ad(\xi) + 1$. However, the above formula α_x would be an example of a formula ξ where $ind(\mathbb{G}''_\xi)$ exceeds $ad(\xi)$: We leave it for the reader to verify that we would get $\Omega''_g(\alpha_z) = 0$, $\Omega''_g(\alpha_y) = 1$ and $\Omega''_g(\alpha_x) = 2$, implying that $ind(\mathbb{G}''_\xi) = 3$.

With our definition of the priority map Ω_g, we find the same values for α_y and α_x as with Ω''_g, but we obtain $\Omega_g(\alpha_z) = 2$, implying that $ind(\mathbb{G}_x) = 2 = ad(\xi)$ as required.

In our technical report [15] we prove in detail that $\mathbb{G}_\xi$ is in fact equivalent to ξ and that $ind(\mathbb{G}_\xi) \leq ad(\xi)$. The proof of the equivalence proceeds by induction on the length of ξ, where we use the strengthened inductive hypothesis that each formula $\varphi \in Clos(\xi)$ is equivalent to $\mathbb{G}_\xi\langle\varphi\rangle$ (that is, the version of $\mathbb{G}$ where we take φ as the initial state). In the crucial case of the inductive step we have $\xi = \eta x.\chi$ and because of our strengthened inductive hypothesis we can assume that $\xi \notin Clos(\chi)$. We then apply the inductive hypothesis to the tidy variant $\chi[x'/x]$ of χ. The claim follows from a comparison of the evaluation games for $\mathbb{G}_\xi$ with the evaluation games for $\mathbb{G}_{\chi[x'/x]}$. For this we need the following proposition:

▶ **Proposition 17.** *Let $\xi = \eta x.\chi$ be a tidy fixpoint formula such that $x \in FV(\chi)$ and $\xi \notin Clos(\chi)$. Let $\chi' := \chi[x'/x]$ for some fresh variable x'. Then χ' is tidy and we have:*

1. *the substitution ξ/x' is a bijection between $Clos(\chi')$ and $Clos(\xi)$.*

Let $\varphi, \psi \in Clos(\chi')$. Then we have

2. *if $\varphi \neq x'$, then $\varphi \to_C \psi$ iff $\varphi[\xi/x'] \to_C \psi[\xi/x']$ and $L_C(\varphi) = L_C(\varphi[\xi/x'])$;*
3. *if $x' \in FV(\varphi)$ then $\varphi \trianglelefteq_f \psi$ iff $\varphi[\xi/x'] \trianglelefteq_f \psi[\xi/x']$;*
4. *if φ and ψ are fixpoint formulas then $\psi \sqsubseteq_C \varphi$ iff $\psi[\xi/x'] \sqsubseteq_C \varphi[\xi/x']$;*
5. *if $(\varphi_n)_{n\in\omega}$ is an infinite trace through $Clos(\chi')$, then $(\varphi_n)_{n\in\omega}$ has the same winner as $(\varphi_n[\xi/x'])_{n\in\omega}$.*

The crucial step in proving that $ind(\mathbb{G}_\xi) \leq ad(\xi)$ is to establish a link between the alternation depth of ξ and the length of alternating $\sqsubset_C$-chains in the closure graph of ξ. This is done by the following proposition, which can be seen as giving an alternative characterisation of the alternation depth of a formula. With $\eta \in \{\mu, \nu\}$, we let $cd_\eta(\xi)$ denote the maximal length of an alternating $\sqsubset_C$-chain in $Clos(\xi)$ that leads up to an η-formula.

▶ **Proposition 18.** *For any tidy formula ξ and $\eta \in \{\mu, \nu\}$, we have*

$$cd_\eta(\xi) \leq n \text{ iff } \xi \in \Theta_n^\eta. \tag{6}$$

Hence the alternation depth of ξ is equal to the length of its longest alternating $\sqsubset_C$-chain.

The main challenge in proving Proposition 18 is the direction from right to left, and more specifically the case of the definition of alternation depth that concerns the closure of Θ_n^η under substitutions. Here we carefully analyse how the alternating $\sqsubset_C$-chains in $C(\psi[\xi/x])$ relate to the ones in $C(\psi)$. For the details, which are fairly complex, we refer to our technical report [15]. Here we just state the crucial proposition that establishes this relation.

▶ **Proposition 19.** *Let ξ and χ be formulas such that ξ is free for x in χ, $\xi \not\trianglelefteq_f \chi$, and $x \notin FV(\xi)$. Furthermore, let $\psi \in Clos(\chi)$ be such that $\psi[\xi/x] \notin Clos(\chi) \cup Clos(\xi)$. Then*

1. *the substitution $\xi/x : C(\psi) \to C(\psi[\xi/x])$ is a bijection between $C(\psi)$ and $C(\psi[\xi/x])$.*

Let $\varphi_0, \varphi_1 \in C(\psi)$. Then we have

2. *$\varphi_0 \to_C \varphi_1$ iff $\varphi_0[\xi/x] \to_C \varphi_1[\xi/x]$ and $L_C(\varphi_0) = L_C(\varphi_0[\xi/x])$;*
3. *$\varphi_0 \trianglelefteq_f \varphi_1$ iff $\varphi_0[\xi/x] \trianglelefteq_f \varphi_1[\xi/x]$;*
4. *$h^\downarrow(\varphi_0) = h^\downarrow(\varphi_0[\xi/x])$, if φ_0 is a fixpoint formula.*

From parity formulas to standard formulas

The construction of an equivalent μ-calculus formula from a parity formula is well known, see for instance [17, 20]. The following theorem provides an analysis on how it behaves in terms of closure size and alternation depth. Given a parity formula $\mathbb{G}$, we let $\mathbb{G}\langle v\rangle$ denote its variant that takes v as its initial state.

▶ **Theorem 20.** *For any parity formula $\mathbb{G} = (V, E, L, \Omega, v_I)$ there is a map $\mathtt{tr}_\mathbb{G} : V \to \mu\mathtt{ML}$ such that, for every $v \in V$:*

1. *$\mathbb{G}\langle v\rangle \equiv \mathtt{tr}_\mathbb{G}(v)$;*
2. *$|\mathtt{tr}_\mathbb{G}(v)|^c \leq 2 \cdot |\mathbb{G}|$;*
3. *$ad(\mathtt{tr}_\mathbb{G}(v)) \leq ind(\mathbb{G})$.*

The details of the definition of $\mathtt{tr}_\mathbb{G}$ and the proofs of items 1–3 can be found in our technical report [15]. Here, we illustrate the basic idea behind the construction by considering the simplified case where the priority map Ω is injective.[8] The definition of $\mathtt{tr}_\mathbb{G}$ proceeds by an induction on the lexicographic order over the pairs of numbers $(|\mathsf{Dom}(\Omega)|, |\mathbb{G}|)$, and we allow ourselves to be sloppy in considering structures consisting of parity formulas without initial vertex. Let T be a top cluster of $\mathbb{G}$, that is, the states in T are not reachable from any state outside T. We make the following case distinction:

Case 1: T is degenerate. In this case we have $T = \{v\}$ for some $v \notin \mathsf{Ran}(E)$. Let $\mathbb{G}'$ be the structure we obtain from $\mathbb{G}$ by removing v from V. We may apply the induction hypothesis to $\mathbb{G}'$ because it is strictly smaller than $\mathbb{G}$, while having no more elements in the domain of the priority map. We define $\mathtt{tr}_\mathbb{G}(u) := \mathtt{tr}_{\mathbb{G}\langle u\rangle}(u)$ for $u \neq v$, while for v we set define $\mathtt{tr}_\mathbb{G}(v)$ by connecting the formulas $\mathtt{tr}_{\mathbb{G}\langle u\rangle}(u)$ for $u \in E(v)$ with $L(v)$ in the obvious way.

Case 2: T is non-degenerate. In this case we have $T \cap \mathsf{Dom}(\Omega) \neq \varnothing$; let $m \in T$ be the state in T of maximal priority, which is unique because of our assumption that Ω is injective.

[8] In fact, it is not hard to see that by shifting priorities we can reduce the general case to this.

For the induction we then consider a fresh propositional variable p_m and define $\mathbb{G}^- = (V^-, E^-, L^-, \Omega^-, v_I)$ as the parity formula over $\mathsf{Q} \cup \{p_m\}$, given by

$$\begin{array}{lcl} V^- & := & V \cup \{m^*\} \\ E^- & := & \{(v,x) \mid (v,x) \in E, x \neq m\} \cup \{(v,m^*) \mid (v,m) \in E\} \\ \Omega^- & := & \Omega\restriction_{V \setminus \{m\}}, \end{array}$$

while its labelling L^- is defined by putting

$$L^-(v) := \begin{cases} L(v) & \text{if } v \in V \\ p_m & \text{if } v = m^*. \end{cases}$$

Since $|\mathsf{Dom}(\Omega^-)| < |\mathsf{Dom}(\Omega)|$, inductively we have a map $\mathtt{tr}_{\mathbb{G}^-} : V^- \to \mu\mathtt{ML}(\mathsf{Q} \cup \{p_m\})$. Let η be the parity of m and define $\mathtt{tr}_{\mathbb{G}}$ as

$$\begin{array}{lcl} \mathtt{tr}_{\mathbb{G}}(m) & := & \eta p_m.\mathtt{tr}_{\mathbb{G}^-}(m) \\ \mathtt{tr}_{\mathbb{G}}(v) & := & \mathtt{tr}_{\mathbb{G}^-}(v)[\mathtt{tr}_{\mathbb{G}}(m)/p_m] \quad \text{for } v \in V. \end{array}$$

The key claim that entails item 2 of Theorem 20 is that

$$|\mathit{Clos}(\mathbb{G})| \leq |\mathbb{G}| + |\mathsf{Dom}(\Omega)|,$$

where $\mathit{Clos}(\mathbb{G}) := \bigcup \{\mathit{Clos}(\mathtt{tr}_{\mathbb{G}}(v)) \mid v \in V\}$. This claim can be proved by the same induction as is used in the definition of $\mathtt{tr}_{\mathbb{G}}$: The point is to treat the closures of all the translations for vertices in $\mathbb{G}$ *in parallel*. The inductive case for non-degenerate clusters then follows with the observation that $\mathit{Clos}(\mathbb{G}) \subseteq \{\varphi[\mathtt{tr}_{\mathbb{G}}(m)/p_m] \mid \varphi \in \mathit{Clos}(\mathbb{G}^-)\}$.

7 Conclusion

This paper contributes to the theory of the modal μ-calculus by studying in detail some representations that are commonly used in order to prove complexity-theoretic results on problems such as model checking or satisfiability. We introduced the notion of a parity formula as a natural graph-based structure for representing formulas, and, building on work by Bruse, Friedmann & Lange [6] we focused on defining succinct parity formula representation on the closure graph of a standard formula. We showed in Proposition 10 that the renaming of bound variables can cause an exponential blow-up if the target formula is required to be clean. To realise the optimal upper complexity bound of model checking for all μ-calculus formulas, as our main contribution, Theorem 12 provides a construction of a parity formula that is based on the closure graph of a given formula, preserves its alternation-depth but does *not* assume the input formula to be clean.

There is a lot more to say about parity formulas as graph-based representations of μ-calculus formulas, but here we confine ourselves to the following.

Our example in Section 5 shows that closure size is not invariant under alphabetical equivalence. This matter could be investigated more thoroughly – here are some pertinent questions. Can we compute alphabetical variants of *minimal* closure size? If we make the reasonable assumption that alphabetical variants should be identified, then we should define the size of a formula as the size of its closure, up to alpha-equivalence; but can we base a parity formula on the quotient of the closure set under α-equivalence? Some answers to these questions can be found in our technical report [15].

Second, we used parity formulas here as a means to understand complexity-theoretic results pertaining to the modal μ-calculus, but it could be interesting to study these structures in their own right. A natural first question is to find a good notion of a morphism or an

equivalence between parity formulas. One might then for instance investigate whether Kozen's expansion map [14] is a morphism from the parity formula based on the subformula dag to the parity formula on the closure. Furthermore, because parity formulas are representations of μ-calculus formulas one might also take a more logical perspective, and develop, for instance, their model theory or proof theory.

References

1. B. Afshari and G. Leigh. Cut-free completeness for modal mu-calculus. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic In Computer Science (LICS'17)*, pages 1–12. IEEE Computer Society, 2017.
2. A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 2001.
3. D. Baelde, A. Doumane, and A. Saurin. Infinitary proof theory: the multiplicative additive case. In J.-M. Talbot and L. Regnier, editors, *Proceedings of the 25th EACSL Annual Conference on Computer Science Logic, CSL 2016*, volume 62 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
4. J. Bradfield and C. Stirling. Modal μ-calculi. In J. van Benthem, P. Blackburn, and F. Wolter, editors, *Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.
5. J.C. Bradfield and I. Walukiewicz. The mu-calculus and model checking. In E. M. Clarke, Th.A. Henzinger, H. Veith, and R. Bloem, editors, *Handbook of Model Checking*, pages 871–919. Springer, 2018. doi:10.1007/978-3-319-10575-8_26.
6. F. Bruse, O. Friedmann, and M. Lange. On guarded transformation in the modal μ-calculus. *Logic Journal of the IGPL*, 23(2):194–216, 2015.
7. C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In H. Hatami, P. McKenzie, and V. King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, (STOC 2017)*, pages 252–263, 2017.
8. G. D'Agostino and M. Hollenberg. Logical questions concerning the μ-calculus. *Journal of Symbolic Logic*, 65:310–332, 2000.
9. S. Demri, V. Goranko, and M. Lange. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.
10. E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proceedings of the 32nd Symposium on the Foundations of Computer Science*, pages 368–377. IEEE Computer Society Press, 1991.
11. G. Fontaine and Y. Venema. Some model theory for the modal mu-calculus: syntactic characterizations of semantic properties. *Logical Methods in Computer Science*, 14(1), 2018.
12. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logic, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.
13. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus w.r.t. monadic second-order logic. In *Proceedings of the Seventh International Conference on Concurrency Theory, CONCUR '96*, volume 1119 of *LNCS*, pages 263–277, 1996.
14. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
15. C. Kupke, J. Marti, and Y. Venema. Size matters in the modal μ-calculus, 2020. arXiv:2010.14430.
16. D. Niwiński. On fixed point clones. In L. Kott, editor, *Proceedings of the 13th International Colloquium on Automata, Languages and Programming (ICALP 13)*, volume 226 of *LNCS*, pages 464–473, 1986.
17. D. Niwiński. Fixed point characterization of infinite behavior of finite-state systems. *Theoretical Computer Science*, 189:1–69, 1997.

18 H. Seidl and A. Neumann. On guarding nested fixpoints. In *Proceedings of the 13th EACSL Annual Conference on Computer Science Logic, CSL '99*, pages 484–498, 1999.

19 I. Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ-calculus. *Information and Computation*, 157:142–182, 2000.

20 T. Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bulletin of the Belgian Mathematical Society*, 8:359–391, 2001.

Spatial Existential Positive Logics for Hyperedge Replacement Grammars

Yoshiki Nakamura
Tokyo Institute of Technology, Japan

Abstract

We study a (first-order) spatial logic based on graphs of conjunctive queries for expressing (hyper-)graph languages. In this logic, each primitive positive (resp. existential positive) formula plays a role of an expression of a graph (resp. a finite language of graphs) modulo graph isomorphism. First, this paper presents a sound- and complete axiomatization for the equational theory of primitive/existential positive formulas under this spatial semantics. Second, we show Kleene theorems between this logic and hyperedge replacement grammars (HRGs), namely that over graphs, the class of existential positive first-order (resp. least fixpoint, transitive closure) formulas has the same expressive power as that of non-recursive (resp. all, linear) HRGs.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Existential positive logic, spatial logic, Kleene theorem

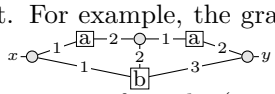
Digital Object Identifier 10.4230/LIPIcs.CSL.2022.30

Related Version *Full Version*: `https://yoshikinakamura.bitbucket.io/papers/ep_for_graph`

Funding This work was supported by JSPS KAKENHI Grant Number JP21K13828.

Acknowledgements We would like to thank the anonymous reviewers for their useful comments.

1 Introduction

Existential positive (EP) *formulas* are first-order formulas that are built up from atomic predicates, equality ($=$), top (tt), bottom (ff), conjunction ($\wedge$), disjunction ($\vee$), and existential quantifier ($\exists$). In particular, *primitive positive* (PP) *formulas* are EP formulas without ff nor $\vee$. PP formulas are semantically equivalent to *conjunctive queries* [1], which are at the core of query languages in database theory. In this paper, we focus on the *(hyper-)graphs* of conjunctive queries (a.k.a. *natural models* of conjunctive queries) [11][12, Fig. 1], which were introduced to characterize the semantical equivalence of conjunctive queries [11, Lemma 13][28] as follows: two PP formulas are semantically equivalent if and only if their graphs are *homomorphically* equivalent. For example, the graph of the PP formula $\exists z.\mathrm{a}(x,z) \wedge \mathrm{a}(z,y) \wedge \mathrm{b}(x,z,y)$ is the following: 

. This characterization can be generalized to EP formulas by using finite sets of graphs (see, e.g., [40, Sect. 2.6]).

In this paper, turning our attention to the correspondence between primitive positive logics and (hyper-)graphs, we study PP/EP formulas as *graph/graph-language expressions*. To this end, we introduce a *spatial* semantics (like that of graph logic [10] or separation logic [35, 38]), which is based on graphs of conjunctive queries, called *GI-semantics*. The semantics enables us to study graphs and graph languages through logical formulas in a natural way. The remarkable difference from classical semantics is the following (cf. the above): two PP formulas are equivalent under GI-semantics if and only if their graphs are *(graph-)isomorphically* equivalent. While the equational theory of PP/EP formulas under GI-semantics is subclassical, some formula transformations under classical semantics, in logic and database theory, still work under GI-semantics.

Our first contribution is to present a sound- and complete axiomatization of the equational theory of PP/EP formulas under GI-semantics. Furthermore, we extend EP with the least-fixpoint operator and the transitive closure operator (see, e.g., [20, Sect. 8]), denoted by EP(LFP) and EP(TC), respectively. They can express possibly infinite graph languages. Our second contribution is to show that each of the logics above has the same expressive power as some class of *hyperedge replacement grammar* (HRG) [25, 36] (see also [19]), which is a generalization of context-free word grammar from words to graphs, as follows.

▶ **Theorem 1.** *Under GI-semantics, for every graph language $\mathcal{G}$ (closed under isomorphism):*

(1) *Some* EP *formula recognizes $\mathcal{G}$ iff some non-recursive HRG recognizes $\mathcal{G}$ (i.e., $\mathcal{G}$ is finite up to isomorphism). In particular, some* PP *formula recognizes $\mathcal{G}$ iff some deterministic and non-recursive HRG recognizes $\mathcal{G}$ (i.e., $\mathcal{G}$ is a singleton up to isomorphism).*

(2) *Some* EP(LFP) *formula recognizes $\mathcal{G}$ iff some HRG recognizes $\mathcal{G}$.*

(3) *Some* EP(TC) *formula recognizes $\mathcal{G}$ iff some linear HRG recognizes $\mathcal{G}$.*

This theorem is an analogy of *Kleene theorem* [27], that over words, for every language L: some regular word grammar (or equivalently, non-deterministic finite automaton) recognizes L if and only if some regular expression recognizes L. Such an equivalence between expressions and grammars/automata like Kleene theorem has also been widely studied for many other language classes (e.g., context-free word languages [29], ω-regular word languages [31], regular tree languages [13, Theorem 2.2.8], language classes over some specific graph classes [30, 6, 5]). To our knowledge, the Kleene theorem for HRGs and linear HRGs (namely, some syntax having the same expressive power) has not yet been investigated, whereas logical or algebraic characterizations are known, e.g., [3, 15].

Related work. This paper uses PP formulas as graph expressions and uses EP(LFP) formulas as graph language expressions. There also are some expressions for (bounded treewidth) graphs (or relational structures), e.g., HR-algebra [3, 16], SP-terms [34], 2p-algebra [14, 18], graphical (string diagrammatic) conjunctive queries [4]. As for the completeness result of PP (Theorem 19), Bauderon and Courcelle [3] have already presented a syntax and a complete axiomatization for graphs modulo isomorphism. However, our completeness proof (essentially [3] also) would have a sufficiently simple strategy relying on the transformation for obtaining conjunctive-queries from primitive positive formulas (under classical semantics); this is a reason that our expressions are based on logical formulas.

As for characterizing language classes by classical logics, it dates back to Büchi-Elgot-Trakhtenbrot Theorem [8, 9, 21, 43] (see also [23]), which states that over words, monadic second-order logic has the same expressive power as the class of regular expressions. See [16, Theorem 7.51][15] for a logical characterization of HRGs, by using monadic second-order logic as a graph transducer. However, the characterization presented in this paper uses logical formulas as *graph-language expressions*.

Also, the number of variables in formulas has a deep connection with the *treewidth* [39, 26] of (hyper-)graphs (or relational structures), which is a parameter indicating how much a graph is similar to a tree. It was mentioned in [28, Remark 5.3] that under the classical semantics, for every relational structure of treewidth k, its conjunctive query is semantically equivalent to an $\mathsf{PP}^{(k+1)(0)}$ formula. Here, $\mathsf{PP}^{k(l)}$ denotes the set of PP formulas using at most k variables and at most l free variables. In particular, it is shown in [32] that under the classical semantics, $\mathsf{PP}^{3(2)}$ has the same expressive power as the primitive positive calculus of relations, which is a fragment of Tarski's calculus of relations [41]. In [14, 18], a sound- and complete axiomatization is presented for 2p-algebra, which is intuitively the primitive positive calculus of relations under GI-semantics. In connection with them, it would be interesting to present a sound- and complete axiomatization of the equational theory of $\mathsf{PP}^{k(l)}$ formulas under GI-semantics, but it still remains open.

Outline. Section 2 presents preliminaries. Section 3 introduces GI-semantics. Section 4 presents an axiomatization of the equational theory under GI-semantics for PP/EP formulas. Section 5 (and 3) shows Kleene theorems between spatial existential positive logic and HRGs (Theorem 1(1)–(3)). Section 6 concludes this paper.

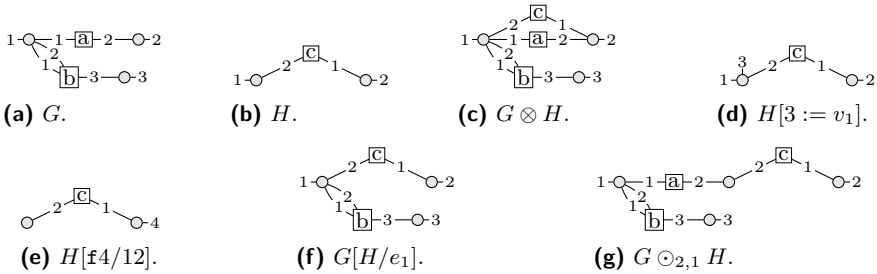
2 Preliminaries

We write $\mathbb{N}$ (resp. $\mathbb{N}_+$) for the set of all non-negative (resp. positive) integers. For $l, r \in \mathbb{N}$, we write $[l, r]$ for the set $\{i \in \mathbb{N} \mid l \leq i \leq r\}$. In particular, we write $[n]$ for $[1, n]$. The *cardinality* of a set A is denoted by $\#(A)$. For an equivalence relation $\sim$ on a set X, the quotient set of X by $\sim$ is denoted by $X/\sim$ and the equivalence class of an element x w.r.t. $\sim$ is denoted by $[x]_\sim$. For sets X_1 and X_2, the *disjoint union* $X_1 \uplus X_2$ is defined by $\{\langle i, a\rangle \mid i \in [2], a \in X_i\}$. We denote by $\vec{a} = \langle a_1, \dots, a_n\rangle$ (also denoted by $a_1 \dots a_n$ or $\langle a_i\rangle_{i=1}^n$) a *finite sequence*. The *length* $|\vec{a}|$ of $\vec{a}$ is n. We denote by $\mathrm{Occ}(\vec{a})$ the set $\{a_1, \dots, a_n\}$. We say that a sequence $\vec{a}$ is a *permutation* of a set A if $\mathrm{Occ}(\vec{a}) = A$ and the elements of $\vec{a}$ are pairwise distinct. We denote by $\mathrm{Perm}(A)$ the set of all permutations of a set A. We denote by A^* (resp. A^k) the set of all finite sequences (resp. sequences of length k) over a set A. Also, we denote by ι_n (or just by ι if n is obvious) the sequence $\langle 1, 2, \dots, n\rangle$. An *alphabet* A is a possibly infinite set. A *(finite-set-)typed alphabet* A is an alphabet with a function ty^A (or written ty for simplicity) from A to finite sets. In particular we say that a symbol a in A is *ordinal-typed* if $\mathrm{ty}^A(a) = [k]$ for some $k \in \mathbb{N}$. The *arity* of a in A is k, denoted by $\mathrm{ar}^A(a)$ (or just by $\mathrm{ar}(a)$).

Graphs. In the following, we define graphs (with ports) and graph languages.

▶ **Definition 2** (graph). *Given a typed alphabet A and a finite set τ, an A-*labelled graph *G of type τ is a tuple $\langle V^G, E^G, \mathrm{lab}^G, \mathrm{vert}^G, \mathrm{port}^G\rangle$, where V^G is a finite set of* vertices*, E^G is a finite set of* (hyper-)edges*, $\mathrm{lab}^G \colon E^G \to A$ is a function denoting the* label *of each edge, $\mathrm{vert}^G(e) \colon \mathrm{ty}^G(e) \to V^G$ is a function denoting the* vertices *of each edge, and $\mathrm{port}^G \colon \mathrm{ty}(G) \to V^G$ is a function denoting the* ports *of G. Here, $\mathrm{ty}(G) \triangleq \tau$ and $\mathrm{ty}^G \triangleq \mathrm{ty}^A \circ \mathrm{lab}^G$.*

▶ **Example 3.** Let $A = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ with type $\mathrm{ty}^A = \{\mathrm{a} \mapsto [2], \mathrm{b} \mapsto [3], \mathrm{c} \mapsto [2]\}$. Let $G = \langle\{v_1, v_2, v_3\}, \{e_1, e_2\}, \{e_1 \mapsto \mathrm{a}, e_2 \mapsto \mathrm{b}\}, \{e_1 \mapsto \lambda i \in [2].v_i, e_2 \mapsto \{1 \mapsto v_1, 2 \mapsto v_1, 3 \mapsto v_3\}\}, \lambda i \in [3].v_i\rangle$ and let $H = \langle\{v_1, v_2\}, \{e\}, \{e \mapsto \mathrm{c}\}, \{e \mapsto \{1 \mapsto v_2, 2 \mapsto v_1\}\}, \lambda i \in [2].v_i\rangle$ be A-labelled graphs (of type $[3]$ and of type $[2]$, respectively), where v_1, v_2, v_3, e_1, e_2 are pairwise distinct. Their graphical representations are in Figure 1a and 1b, respectively.



(a) G. **(b)** H. **(c)** $G \otimes H$. **(d)** $H[3 := v_1]$.

(e) $H[\mathfrak{f}4/12]$. **(f)** $G[H/e_1]$. **(g)** $G \odot_{2,1} H$.

Figure 1 Examples of graphs and operations on graphs.

Later (e.g., in Example 12), for binary edges and ports, we often use ○-[a]-▸○ to denote ○-1-[a]-2-○ for symbols a of the type $[2]$ and use →○ ○→ to denote 1-○ ○-2. Also, for unlabelled non-hyper graphs, let $A_{\mathrm{E}} \triangleq \{\mathrm{E}\}$ with $\mathrm{ty}^{A_{\mathrm{E}}} = \{\mathrm{E} \mapsto [2]\}$ and we use ○⟶○ to denote ○-[E]-▸○.

We denote by GR_A^τ the set of all A-labelled graphs of type τ. An *(A-labelled) graph language* $\mathcal{G}$ (of type τ) is a subset of GR_A^τ. Given a system $\mathfrak{S}$ (e.g., HRGs, EP formulas, ...) over A (that defines a graph language $\mathcal{G}(E)$ for every E in $\mathfrak{S}$), we say that $\mathcal{G}$ is *recognized* by $\mathfrak{S}$ if there exists some element E in $\mathfrak{S}$ such that $\mathcal{G} = \mathcal{G}(E)$.

▶ **Definition 4** (homomorphism, isomorphism). *Let $G, H \in \mathsf{GR}_A^\tau$ be graphs. A pair $h = \langle h^{\mathrm{V}}, h^{\mathrm{E}} \rangle$ of $h^{\mathrm{V}} \colon V^G \to V^H$ and $h^{\mathrm{E}} \colon E^G \to E^H$ is a* homomorphism *from G to H if (1) $\mathrm{lab}^G = \mathrm{lab}^H \circ h^{\mathrm{E}}$, (2) $\mathrm{vert}^H(h^{\mathrm{E}}(e))(x) = h^{\mathrm{V}}(\mathrm{vert}^G(e)(x))$, and (3) $\mathrm{port}^H = h^{\mathrm{V}} \circ \mathrm{port}^G$. In particular, h is an* isomorphism *if both h^{V} and h^{E} are bijective. We say that G and H are* isomorphic*, written $G \cong H$ if there exists an isomorphism between G and H.*

In this paper, we will only focus on $\cong$-*closed* (i.e., if $G \in \mathcal{G}$ and $G \cong H$, then $H \in \mathcal{G}$) graph languages. We denote by $\mathcal{G}^{\cong}$ the minimal $\cong$-closed graph language including $\mathcal{G}$.

Some operations on graphs. In the following, we present some primitive operations on graphs. See Figure 1c-1g for graphical examples of Definition 5-8. In GI-semantics, $*$ uses glueing, $\exists$ uses forgetting, LFP uses hyperedge replacing, TC uses concatenating.

▶ **Definition 5** (glueing). *Let $G_1 \in \mathsf{GR}_A^\tau$ and $G_2 \in \mathsf{GR}_A^\upsilon$. Let $G_1 \otimes G_2 \in \mathsf{GR}_A^{\tau \cup \upsilon}$ be the graph such that $V^{G_1 \otimes G_2} = (V^{G_1} \uplus V^{G_2})/\simeq$, $E^{G_1 \otimes G_2} = E^{G_1} \uplus E^{G_2}$, $\mathrm{lab}^{G_1 \otimes G_2}(\langle k, e \rangle) = \mathrm{lab}^{G_k}(e)$, $\mathrm{vert}^{G_1 \otimes G_2}(\langle k, e \rangle)(x) = [\mathrm{vert}^{G_k}(e)(x)]_\simeq$, and $\mathrm{port}^{G_1 \otimes G_2}(x) = [\mathrm{port}^{G_k}(x)]_\simeq$. Here, $\simeq$ is the minimal equivalence relation such that for every $x \in \tau \cap \upsilon$, $\langle 1, \mathrm{port}^{G_1}(x) \rangle \simeq \langle 2, \mathrm{port}^{G_2}(x) \rangle$.*

▶ **Definition 6** (labelling/forgetting/renaming). *Let $G \in \mathsf{GR}_A^\tau$. For a vertex $v \in V^G$, a variable $z \notin \tau$, and a variable $x \in \tau$, we define the graphs $G[z := v] \in \mathsf{GR}_A^{\tau \cup \{z\}}$, $G[\mathtt{f}/x] \in \mathsf{GR}_A^{\tau \setminus \{x\}}$, $G[z/x] \in \mathsf{GR}_A^{(\tau \setminus \{x\}) \cup \{z\}}$ by $G[z := v] \triangleq \langle V^G, E^G, \mathrm{lab}^G, \mathrm{vert}^G, \mathrm{port}^G \cup \{z \mapsto v\} \rangle$, $G[\mathtt{f}/x] \triangleq \langle V^G, E^G, \mathrm{lab}^G, \mathrm{vert}^G, \mathrm{port}^G \setminus \{x \mapsto \mathrm{port}^G(x)\} \rangle$, $G[z/x] \triangleq G[\mathtt{f}/x][z := \mathrm{port}^G(x)]$.*

We write $G[y_1 \dots y_n / x_1 \dots x_n]$ for $G[z_1/x_1] \dots [z_n/x_n][y_1/z_1] \dots [y_n/z_n]$, where $z_1 \dots z_n$ is a sequence of fresh variables. For a sequence $z_1 \dots z_n$ of pairwise distinct variables, we write $G[z_1 \dots z_n := v_1 \dots v_n]$ for $G[z_1 := v_1] \dots [z_n := v_n]$.

▶ **Definition 7** (hyperedge replacing). *Let $G \in \mathsf{GR}_A^\tau$. For an edge $e \in E^G$ and a graph $H \in \mathsf{GR}_A^{\mathrm{ty}^G(e)}$, let $G[H/e] \in \mathsf{GR}_A^\tau$ be the graph $((G \setminus e)[\vec{z} := \mathrm{vert}^G(e)(x_1) \dots \mathrm{vert}^G(e)(x_n)] \otimes H[\vec{z}/x_1 \dots x_n])[\mathtt{f} \dots \mathtt{f}/\vec{z}]$, where $G \setminus e$ denotes the graph G in which the edge e has been removed. Here, $x_1 \dots x_n \in \mathrm{Perm}(\mathrm{ty}(H))$, and $\vec{z}$ is a sequence of fresh variables.*

We write $G[H_1 \dots H_n / e_1 \dots e_n]$ for $G[H_1/e_1][H_2 \dots H_n / \langle 1, e_2 \rangle \dots \langle 1, e_n \rangle]$ if $n \geq 1$, and G if $n = 0$.

▶ **Definition 8** (concatenating). *Let $G \in \mathsf{GR}_A^\tau$ and $H \in \mathsf{GR}_A^\upsilon$. Let $\vec{x} \in \mathrm{ty}(G)^k$ and $\vec{y} \in \mathrm{ty}(H)^k$ be sequences of pairwise distinct elements, where $k \geq 1$. Then, let $G \odot_{\vec{x}\vec{y}} H \in \mathsf{GR}_A^{(\tau \setminus \mathrm{Occ}(\vec{x})) \cup (\upsilon \setminus \mathrm{Occ}(\vec{y}))}$ be the graph $(G[\vec{z}/\vec{x}] \otimes H[\vec{z}/\vec{y}])[\mathtt{f} \dots \mathtt{f}/\vec{z}]$, where $\vec{z}$ is a sequence of fresh variables.*

Finally, we list some basic equations in the following.

▶ **Proposition 9.** *(1) $G_1 \otimes (G_2 \otimes G_3) \cong (G_1 \otimes G_2) \otimes G_3$; (2) $G \otimes H \cong H \otimes G$; (3) $(H_1 \otimes H_2)[G/\langle 1, e \rangle] \cong H_1[G/e] \otimes H_2$; (4) $G[z/x][H/e] \cong G[H/e][z/x]$; (5) $G[z/x] \otimes H \cong (G \otimes H)[z/x]$ if $x \notin \mathrm{ty}(H)$.*

Hyperedge Replacement Grammars. In the following, we present the definition of hyperedge replacement grammars (HRGs).

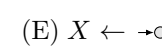
▶ **Definition 10** (e.g., [19]). *A* hyperedge replacement grammar *(HRG)* $\mathscr{H}$ *over a typed alphabet* A *is a tuple* $\langle \mathcal{X}^{\mathscr{H}}, \mathcal{R}^{\mathscr{H}}, \mathtt{S}^{\mathscr{H}} \rangle$*, where* $\mathcal{X}^{\mathscr{H}}$ *is a finite typed alphabet disjoint with* A *for* (non-terminal) labels, $\mathcal{R}^{\mathscr{H}}$ *is a finite set of pairs* $r = \langle X, G \rangle$ *(written* $X \leftarrow G$*) of* $X \in \mathcal{X}^{\mathscr{H}}$ *and* $G \in \mathsf{GR}_{A \cup \mathcal{X}^{\mathscr{H}}}^{\mathrm{ty}(X)}$ *for* rewriting rules*, and* $\mathtt{S}^{\mathscr{H}} \in \mathcal{X}^{\mathscr{H}}$ *denotes the* source label.

We also define the graph languages of HRGs as follows.

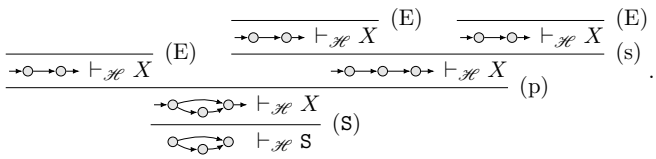
▶ **Definition 11** (cf. [19, Sect. 2.3.2]). *For an HRG* $\mathscr{H} = \langle \mathcal{X}, \mathcal{R}, \mathtt{S} \rangle$ *over a typed alphabet* A*, the binary relation* $\vdash_{\mathscr{H}} \subseteq \bigcup_{X \in \mathcal{X}} \mathsf{GR}_A^{\mathrm{ty}(X)} \times \{X\}$ *is defined as the least* $\cong$*-closed (i.e., if* $G \cong H$ *and* $G \vdash_{\mathscr{H}} X$*, then* $H \vdash_{\mathscr{H}} X$*) relation closed under the following rule: If* $X \leftarrow G \in \mathcal{R}$*, then* $\dfrac{H_1 \vdash_{\mathscr{H}} \mathrm{lab}^G(e_1) \ \dots \ H_n \vdash_{\mathscr{H}} \mathrm{lab}^G(e_n)}{G[H_1 \dots H_n / e_1 \dots e_n] \vdash_{\mathscr{H}} X}$*. The* graph language *is defined by:* $\mathcal{G}(\mathscr{H}) \triangleq \{G \in \mathsf{GR}_A^{\mathrm{ty}(\mathtt{S})} \mid G \vdash_{\mathscr{H}} \mathtt{S}\}$.

For an HRG $\mathscr{H}$, we say that $\mathscr{H}$ is *linear* [36, Definition 3] if for every rule $X \leftarrow G \in \mathcal{R}^{\mathscr{H}}$, the number of non-terminal labels occurring in G is at most one. We say that $\mathscr{H}$ is *(n-)recursive* if there exist rules $X_0 \leftarrow G_0, \dots, X_n \leftarrow G_n \in \mathcal{R}^{\mathscr{H}}$ such that X_i occurs in G_{i-1} for $i \in [0, n]$ where $n \in \mathbb{N}$ and G_{-1} denotes G_n.

▶ **Example 12.** Let $\mathscr{H}$ be the HRG over A_{E}, defined by $\mathrm{ty}^{\mathcal{X}^{\mathscr{H}}} = \{\mathtt{S} \mapsto [0], X \mapsto [2]\}$, $\mathcal{R}^{\mathscr{H}} = \{(\mathtt{S}), (\mathrm{E}), (\mathrm{s}), (\mathrm{p})\}$, and $\mathtt{S}^{\mathscr{H}} = \mathtt{S}$, where each rule in $\mathcal{R}^{\mathscr{H}}$ is as follows:

(S) $\mathtt{S} \leftarrow$  (E) $X \leftarrow$ (s) $X \leftarrow$  (p) $X \leftarrow$

Then, $\mathcal{G}(\mathscr{H})$ is the set of all (directed) series-parallel graphs [24], e.g.,  $\in \mathcal{G}(\mathscr{H})$ is shown by:



3 Existential Positive Logics under GI-Semantics

In this section, we introduce the syntax and a *spatial semantics* of our existential positive logics. Let A be an ordinal-typed alphabet, $\mathscr{V}_1$ be a countably infinite set of *first-order variables*, and $\mathscr{V}_2$ be an ordinal-typed set of *second-order variables*, where for every $k \in \mathbb{N}_+$, the number of second-order variables of arity k is countably infinite. Here, A, $\mathscr{V}_1$, and $\mathscr{V}_2$ are disjoint. For $\tau \subseteq \mathscr{V}_1$ and $\mathcal{X} \subseteq A \cup \mathscr{V}_2$, we define $\mathsf{Fml}_{\mathcal{X}}^{\tau}$ as the least set closed under the rules as follows.[1]

$$\frac{}{\top \in \mathsf{Fml}_{\mathcal{X}}^{\emptyset}} \quad \frac{}{x = y \in \mathsf{Fml}_{\mathcal{X}}^{\{x,y\}}} \quad \frac{}{X\vec{x} \in \mathsf{Fml}_{\mathcal{X}}^{\mathrm{Occ}(\vec{x})}}\dagger_1 \quad \frac{\varphi \in \mathsf{Fml}_{\mathcal{X}}^{\tau} \qquad \psi \in \mathsf{Fml}_{\mathcal{X}}^{\upsilon}}{\varphi * \psi \in \mathsf{Fml}_{\mathcal{X}}^{\tau \cup \upsilon}} \quad \frac{\varphi \in \mathsf{Fml}_{\mathcal{X}}^{\tau \cup \{x\}}}{\exists x.\varphi \in \mathsf{Fml}_{\mathcal{X}}^{\tau}}\dagger_2$$

$$\frac{}{\mathsf{ff} \in \mathsf{Fml}_{\mathcal{X}}^{\tau}} \quad \frac{\varphi \in \mathsf{Fml}_{\mathcal{X}}^{\tau} \qquad \psi \in \mathsf{Fml}_{\mathcal{X}}^{\tau}}{\varphi \vee \psi \in \mathsf{Fml}_{\mathcal{X}}^{\tau}} \quad \frac{\varphi \in \mathsf{Fml}_{\mathcal{X} \cup \{X\}}^{\mathrm{Occ}(\vec{x})}}{[\mathsf{LFP}_{\vec{x},X}\varphi]\vec{y} \in \mathsf{Fml}_{\mathcal{X}}^{\mathrm{Occ}(\vec{y})}}\dagger_3 \quad \frac{\varphi \in \mathsf{Fml}_{\mathcal{X}}^{\mathrm{Occ}(\vec{x}\vec{y})}}{[\varphi]^{+}_{\vec{x}\vec{y}}\vec{u}\vec{w} \in \mathsf{Fml}_{\mathcal{X}}^{\mathrm{Occ}(\vec{u}\vec{w})}}\dagger_4$$

$\dagger_1$: $X \in \mathcal{X}$ and $\mathrm{ar}^{\mathcal{X}}(X) = |\vec{x}|$. $\dagger_2$: $x \notin \tau$. $\dagger_3$: $\mathrm{ar}(X) = |\vec{x}| = |\vec{y}| \geq 1$. $\vec{x}$ and $\vec{y}$ are sequences of pairwise distinct variables. $\dagger_4$: $|\vec{x}| = |\vec{y}| = |\vec{u}| = |\vec{w}| \geq 1$. $\vec{x}\vec{y}$ and $\vec{u}\vec{w}$ are sequences of pairwise distinct variables.

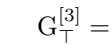
[1] We adopt the spatial conjunction symbol $*$ instead of $\wedge$.

We often use parentheses in ambiguous situations. We say that φ is a *formula* over A of *type* τ if $\varphi \in \mathsf{Fml}^{\tau}_{A}$. Note that, for a technical reason, ff has any type τ. We use $\mathsf{FV}_1(\varphi)/\mathsf{FV}_2(\varphi)$ (resp. $\mathsf{BV}_1(\varphi)/\mathsf{BV}_2(\varphi)$) to denote the set of first-/second-order free (resp. bound) variables of φ, and use $\mathsf{V}_l(\varphi)$ to denote the set $\mathsf{FV}_l(\varphi) \cup \mathsf{BV}_l(\varphi)$ for $l = 1, 2$. The set PP^{τ}_{A} *(resp.* EP^{τ}_{A}, $\mathsf{EP}(\mathsf{LFP})^{\tau}_{A}$, $\mathsf{EP}(\mathsf{TC})^{\tau}_{A}$*)* is defined as the set of all $\varphi \in \mathsf{Fml}^{\tau}_{A}$ such that φ is generated from the rules for $\top$, $=$, $X\vec{x}$, $*$, and $\exists$. (resp. the rules for PP with ff and $\vee$, the rules for EP with LFP, the rules for EP with TC). Note that some syntax restrictions exist, e.g., $\top \vee Xx \notin \mathsf{Fml}^{\tau}_{\mathcal{X}}$ for any $\mathcal{X}$ and τ. They are for simplifying the definition of GI-semantics.

For notational simplicity, we denote by $\bigstar^{n}_{i=1} \varphi_i$ (similarly for $\bigvee^{n}_{i=1} \varphi_i$) the formula $(\bigstar^{n-1}_{i=1} \varphi_i) * \varphi_n$ if $n \geq 1$ and the formula $\top$ if $n = 0$, by $x_1 \dots x_n = y_1 \dots y_n$ the formula $\bigstar^{n}_{i=1} x_i = y_i$, by $\exists x_1 \dots x_n.\varphi$ the formula $\exists x_1.\exists x_2. \dots \exists x_n.\varphi$, and by $\varphi[y_1 \dots y_n/x_1 \dots x_n]$ the formula φ in which each free variable x_i occurring in φ has been replaced with y_i where $i \in [n]$. A formula φ is *atomic* if φ forms $\top$, $x = y$, or $X\vec{x}$. Explicitly, we may use $\tilde{\varphi}$ to denote an atomic formula. We use atomic formulas to denote atomic graphs as follows.

▶ **Definition 13.** *For a finite set* τ*, let* $\mathrm{G}^{\tau}_{\top} \triangleq \langle \tau, \emptyset, \emptyset, \emptyset, \lambda x \in \tau.x \rangle$*. For an atomic formula* $\tilde{\varphi}$*, we define the graph* $\mathrm{G}_{\tilde{\varphi}}$ *by:* $\mathrm{G}_{\top} \triangleq \mathrm{G}^{\emptyset}_{\top}$*,* $\mathrm{G}_{x=y} \triangleq \langle \{\mathrm{v}\}, \emptyset, \emptyset, \emptyset, \lambda z \in \{x, y\}.\mathrm{v} \rangle$*, and* $\mathrm{G}_{X\vec{x}} \triangleq \langle \mathrm{Occ}(\vec{x}), \{\mathrm{e}\}, \{\mathrm{e} \mapsto X\}, \{\mathrm{e} \mapsto \lambda i \in \mathrm{ty}(X).x_i\}, \lambda y \in \mathrm{Occ}(\vec{x}).y \rangle$*.*

For example, $\mathrm{G}^{[3]}_{\top}$, $\mathrm{G}_{x=y}$, and G_{Xxxy} are as follows, where $x \neq y$:

$\mathrm{G}^{[3]}_{\top} =$ $\mathrm{G}_{x=y} =$  $\mathrm{G}_{Xxxy} =$ 

In the following, we define a spatial semantics for graph languages, called *GI-semantics.*[2] Note that for every φ, if $G \models^{\mathrm{GI}} \varphi$, then $\mathrm{ty}(G)$ is determined to $\mathsf{FV}_1(\varphi)$.

▶ **Definition 14** (GI-semantics)**.** *The binary relation* $\models^{\mathrm{GI}} \subseteq \bigcup_{\tau \subseteq \mathscr{V}_1;\, \mathcal{X} \subseteq A \cup \mathscr{V}_2} \mathsf{GR}^{\tau}_{\mathcal{X}} \times \mathsf{Fml}^{\tau}_{\mathcal{X}}$ *is defined as the least* $\cong$*-closed relation closed under the rules in Figure 2.*

$$\frac{}{\mathrm{G}_{\tilde{\varphi}} \models^{\mathrm{GI}} \tilde{\varphi}}\,(\mathrm{At}) \qquad \frac{G \models^{\mathrm{GI}} \varphi \qquad H \models^{\mathrm{GI}} \psi}{G \otimes H \models^{\mathrm{GI}} \varphi * \psi}\,(*) \qquad \frac{\langle G_i \models^{\mathrm{GI}} \varphi \rangle^{n}_{i=1}}{(G_1 \odot_{\vec{y}\vec{x}} \cdots \odot_{\vec{y}\vec{x}} G_n)[\vec{u}\vec{w}/\vec{x}\vec{y}] \models^{\mathrm{GI}} [\varphi]^{+}_{\vec{x}\vec{y}}\vec{u}\vec{w}}\,(\mathsf{TC})\ \dagger_1$$

$$\frac{G \models^{\mathrm{GI}} \varphi}{G[\mathtt{f}/x] \models^{\mathrm{GI}} \exists x.\varphi}\,(\exists) \qquad \frac{G \models^{\mathrm{GI}} \varphi_i}{G \models^{\mathrm{GI}} \varphi_1 \vee \varphi_2}\,(\vee)\ \dagger_2 \qquad \frac{H \models^{\mathrm{GI}} \varphi \qquad \langle G_i \models^{\mathrm{GI}} [\mathsf{LFP}_{\vec{x},X}\varphi]\iota \rangle^{n}_{i=1}}{H[G_1 \dots G_n/\vec{e}^{H}_{X}][\vec{y}/\vec{x}] \models^{\mathrm{GI}} [\mathsf{LFP}_{\vec{x},X}\varphi]\vec{y}}\,(\mathsf{LFP})\ \dagger_3$$

$\dagger_1$: $n \in \mathbb{N}_+$. $\dagger_2$: $i \in [2]$. $\dagger_3$: $n \in \mathbb{N}$ and $\vec{e}^{H}_{X}$ denotes a permutation of all the X-labelled edges in H.

Figure 2 Definition of GI-semantics.

The *graph language* of φ is defined by $\mathcal{G}(\varphi) \triangleq \{G \mid G \models^{\mathrm{GI}} \varphi\}$. We say that φ and ψ are *graph-isomorphically equivalent* (GI-equivalent), written $\varphi \cong^{\mathrm{GI}} \psi$ if $\mathcal{G}(\varphi) = \mathcal{G}(\psi)$.

▶ **Example 15.** Let $G \triangleq$  and $\varphi \triangleq x = y * \exists z.\mathrm{E}xz * \mathrm{E}zy$. Then, $G \models^{\mathrm{GI}} \varphi$ is shown by:

$$\dfrac{\dfrac{}{\text{  } \models^{\mathrm{GI}} \mathrm{E}xz}\,(\mathrm{At}) \qquad \dfrac{}{\text{  } \models^{\mathrm{GI}} \mathrm{E}zy}\,(\mathrm{At})}{\text{} \models^{\mathrm{GI}} \mathrm{E}xz * \mathrm{E}zy}\,(*) \qquad \dfrac{}{\text{} \models^{\mathrm{GI}} x = y}\,(\mathrm{At})}{\text{} \models^{\mathrm{GI}} \exists z.\mathrm{E}xz * \mathrm{E}zy}\,(\exists)$$

$$\dfrac{\text{} \models^{\mathrm{GI}} x = y \qquad \text{} \models^{\mathrm{GI}} \exists z.\mathrm{E}xz * \mathrm{E}zy}{\text{  } \models^{\mathrm{GI}} x = y * \exists z.\mathrm{E}xz * \mathrm{E}zy}\,(*)$$

We will generalize this example in Definition 16, for expressing any graphs by PP formulas.

[2] See [33, Appendix A] for an alternative definition. Here, we adopt this style for extending to Definition 27.

3.1 PP/EP formulas as graph/finite-graph-language expressions

In this subsection, we show that PP (resp. EP) formulas under GI-semantics play a role as graph expressions (resp. finite graph language expressions).

▶ **Definition 16.** *Let G be a graph, $\vec{x} = x_1 \dots x_k \in \mathrm{Perm}(\mathrm{ty}(G))$, $\vec{v} = v_1 \dots v_n \in \mathrm{Perm}(V^G)$, and $\vec{e} = e_1 \dots e_m \in \mathrm{Perm}(E^G)$. Let $\varphi_G^{\vec{x},\vec{v},\vec{e}}$ (or written φ_G if they are not important) be the following* PP *formula, where $z_{v_1}, \dots, z_{v_n}$ are fresh variables:*

$$\exists z_{v_1} \dots z_{v_n}.(\mathop{\ast}_{i=1}^{k} z_{\mathrm{port}^G(x_i)} = x_i) * (\mathop{\ast}_{i=1}^{m} \mathrm{lab}^G(e_i)\ z_{\mathrm{vert}^G(e_i)(1)} \dots z_{\mathrm{vert}^G(e_i)(\mathrm{ar}^G(e_i))}).$$

Also, for a finite sequence $\vec{G} = G_1 \dots G_n$ of graphs, let $\varphi_{\vec{G}}$ be the EP *formula $\bigvee_{i=1}^{n} \varphi_{G_i}$.*

Then, $\mathcal{G}(\varphi_G) = \{G\}^{\cong}$ and $\mathcal{G}(\varphi_{\vec{G}}) = \mathrm{Occ}(\vec{G})^{\cong}$. By using them, the following holds.

▶ **Proposition 17** (Theorem 1(1))**.** *For every graph language $\mathcal{G}$ closed under isomorphism: (1): $\mathcal{G}$ is singleton up to isomorphism iff some* PP *formula recognizes $\mathcal{G}$. (2): $\mathcal{G}$ is finite up to isomorphism iff some* EP *formula recognizes $\mathcal{G}$.*

Proof. (1)(2)(⇒): By using φ_G and $\varphi_{\vec{G}}$, respectively. (1)(2)(⇐): By a straightforward induction on the structure of PP (resp. EP) formulas. ◀

▶ Remark 18. Indeed, GI-semantics characterizes the *graphs of* PP *formulas* [11] (see also [12, Figure 1]), namely, for every PP formula φ, $G \models^{\mathrm{GI}} \varphi$ iff G is isomorphic to the graph of φ. Thus, two PP formulas are GI-equivalent iff their graphs are isomorphically equivalent.

4 An Axiomatization of the Equational Theory of PP/EP

This section presents an axiomatization of the equational theory under GI-semantics (i.e., the binary relation $\cong^{\mathrm{GI}}$) of PP/EP formulas. Given an ordinal-typed alphabet A, we define the binary relation $\simeq \subseteq \bigcup_{\tau \subseteq \mathcal{V}_1} \mathsf{EP}_A^\tau \times \mathsf{EP}_A^\tau$ as the minimal relation closed under the rules in Figure 3.[3] Inference rules consist of the rules for equivalence relation and the rules for "α-equivalence" (see, e.g., [37, Sect. 4.1.] for λ-calculus).

Inference rules:

$$\frac{}{\varphi \simeq \varphi} \quad \frac{\varphi \simeq \psi}{\psi \simeq \varphi} \quad \frac{\varphi \simeq \psi \ \ \psi \simeq \rho}{\varphi \simeq \rho} \quad \frac{\varphi \simeq \varphi' \ \ \psi \simeq \psi'}{\varphi * \psi \simeq \varphi' * \psi'} \quad \frac{\varphi[z/x] \simeq \psi[z/y]}{\exists x.\varphi \simeq \exists y.\psi}\,\dagger_1 \quad \frac{\varphi \simeq \varphi' \ \ \psi \simeq \psi'}{\varphi \vee \psi \simeq \varphi' \vee \psi'}$$

Axioms:

(=1) $x = y \simeq y = x$ (=2) $x = x * \varphi \simeq \varphi$ (=3) $x = y * \varphi[x/z] \simeq x = y * \varphi[y/z]$ (=4) $\exists x.x = y \simeq y = y$
(∗1) $\varphi * (\psi * \rho) \simeq (\varphi * \psi) * \rho$ (∗2) $\varphi * \psi \simeq \psi * \varphi$ (∗3) $\varphi * \top \simeq \varphi$ (∃1) $\exists x.\exists y.\varphi \simeq \exists y.\exists x.\varphi$
(∃2) $(\exists x.\varphi) * \psi \simeq \exists x.\varphi * \psi$ (∨1) $\varphi \vee (\psi \vee \rho) \simeq (\varphi \vee \psi) \vee \rho$ (∨2) $\varphi \vee \psi \simeq \psi \vee \varphi$ (∨3) $\varphi \vee \mathsf{ff} \simeq \varphi$
(∨4) $\varphi \vee \varphi \simeq \varphi$ (∨5) $\exists x.\varphi \vee \psi \simeq (\exists x.\varphi) \vee (\exists x.\psi)$ (∨6) $\varphi * (\psi \vee \rho) \simeq (\varphi * \psi) \vee (\varphi * \rho)$ (ff) $\mathsf{ff} * \varphi \simeq \mathsf{ff}$
$\dagger_1$: z is a fresh variable.

Figure 3 An axiomatization of the equational theory under GI-semantics of PP/EP formulas.

[3] We assume that the left- and right-hand side formulas have an identical type. This restriction implicitly implies the following: when their graph languages are not empty, $x \notin \mathsf{FV}_1(\psi)$ in (∃2), $x \in \mathsf{FV}_1(\varphi)$ in (=2), and $y \neq x$ in (=4), respectively. Also, note that we can use (ff) even if $\mathrm{ty}(\varphi) \neq \emptyset$, because ff has any type.

▶ **Theorem 19.** *The system in Figure 3 is sound and complete for the equational theory under GI-semantics of* PP/EP *formulas, that is, for every* $\varphi, \psi \in \mathsf{EP}^{\tau}_{A}$, $\varphi \simeq \psi$ *iff* $\varphi \cong^{\mathrm{GI}} \psi$.

In the next subsection, we prove this theorem. The following is a proof sketch.

Proof Sketch of Theorem 19. The soundness is straightforward. For completeness, we show by using the rules in Figure 3 that we can transform each formula into a normal form in two steps: (1) transform each EP formula into a disjunctive normal form of PP formulas; (2) transform each PP formula into a formula of the form φ_G in Definition 16.

4.1 Proof of Theorem 19

▶ **Proposition 20.** (1)*:* $\varphi_G^{\vec{x}_1,\vec{v}_1,\vec{e}_1} \simeq \varphi_G^{\vec{x}_2,\vec{v}_2,\vec{e}_2}$. (2)*: If there is an isomorphism* h *from* G *to* H*, then* $\varphi_G^{x_1\dots x_k,v_1\dots v_n,e_1\dots e_m} \simeq \varphi_H^{x_1\dots x_k,h^{\mathrm{V}}(v_1)\dots h^{\mathrm{V}}(v_n),h^{\mathrm{E}}(e_1)\dots h^{\mathrm{E}}(e_m)}$. (3)*: If* $G \cong H$*, then* $\varphi_G^{\vec{x}_1,\vec{v}_1,\vec{e}_1} \simeq \varphi_H^{\vec{x}_2,\vec{v}_2,\vec{e}_2}$.

Proof. (1): By permutating names using $(*1)(*2)$ for $\vec{x}_1$ and $\vec{x}_2$, $(\exists 1)$ for $\vec{v}_1$ and $\vec{v}_2$, $(*1)(*2)$ for $\vec{e}_1$ and $\vec{e}_2$, respectively. (2): Since they are the same up to variable names. (3): By (2)(1). ◀

Hereafter in this section, relying on this proposition, we write $\varphi_G^{\vec{x},\vec{v},\vec{e}}$ as φ_G, for simplicity.

▶ **Lemma 21.** *For every* PP *formula* φ*: (1): Let* $x \in \mathsf{FV}_1(\varphi)$ *and* $y \neq x$*. Then,* $\exists x.x = y * \varphi \simeq \varphi[y/x]$*. (2): Let* $z_1 \dots z_n \in \mathrm{Perm}(\mathsf{FV}_1(\varphi))$*,* $k \in \mathbb{N}$*, and* $f, g\colon [k] \to [n]$ *be maps. Let* $\sim$ *be the minimal equivalence relation on* $[n]$ *such that for every* $i \in [k]$*,* $f(i) \sim g(i)$ *and let* $I_1 \dots I_m$ *be a permutation of all the quotient classes of* $[n]$ *w.r.t.* $\sim$*. Then,* $\exists z_1 \dots z_n.(\mathop{\ast}_{i=1}^{k} z_{f(i)} = z_{g(i)}) * \varphi \simeq \exists z_{I_1} \dots z_{I_m}.\varphi[z_{[1]_\sim} \dots z_{[n]_\sim}/z_1 \dots z_n]$*. Here,* $z_{I_1}, \dots, z_{I_m}$ *are pairwise distinct variables.*

Proof. (1): $\exists x.x = y * \varphi \simeq_{(=3)} \exists x.x = y * \varphi[y/x] \simeq_{(\exists 2)} (\exists x.x = y) * \varphi[y/x] \simeq_{(\exists 5)} y = y * \varphi[y/x] \simeq_{(=2)} \varphi[y/x]$. (2): By induction on k. Case $k = 0$. $\exists z_1 \dots z_n.\top * \varphi \simeq_{(*2)(*3)} \exists z_1 \dots z_n.\varphi \simeq \exists z_{\{1\}} \dots z_{\{n\}}.\varphi[z_{\{1\}} \dots z_{\{n\}}/z_1 \dots z_n]$. Case $k \geq 1$. Then,

$$\exists z_1 \dots z_n.(\mathop{\ast}_{i=1}^{k} z_{f(i)} = z_{g(i)}) * \varphi \simeq_{(*1)} \exists z_1 \dots z_n.(\mathop{\ast}_{i=1}^{k-1} z_{f(i)} = z_{g(i)}) * (z_{f(k)} = z_{g(k)} * \varphi)$$

$$\simeq \exists z_{I'_1} \dots z_{I'_{m'}}.z_{[f(k)]_{\sim'}} = z_{[g(k)]_{\sim'}} * \varphi[z_{[1]_{\sim'}} \dots z_{[n]_{\sim'}}/z_1 \dots z_n]$$

($\sim'$ and $I'_1 \dots I'_{m'}$ are the ones obtained by I.H. w.r.t. $k-1$.)

(Here, we assume without loss of generality by $(\exists 1)$ that $z_{I'_{m'}} = z_{[f(k)]_{\sim'}}$.)

$$\simeq \exists z_{I'_1} \dots z_{I'_m}.\varphi[z_{[1]_{\sim'}} \dots z_{[n]_{\sim'}}/z_1 \dots z_n][z_{[g(k)]_{\sim'}}/z_{[f(k)]_{\sim'}}]$$

(Apply (=2) if $[f(k)]_{\sim'} = [g(k)]_{\sim'}$ and (1) if $[f(k)]_{\sim'} \neq [g(k)]_{\sim'}$.)

(Here, $m = m'$ for (=2) and $m = m' - 1$ for (1).)

$$\simeq \exists z_{I_1} \dots z_{I_m}.\varphi[z_{[1]_\sim} \dots z_{[n]_\sim}/z_1 \dots z_n].$$

(They are the same up to variable names.) ◀

▶ **Lemma 22.** *For every* PP *formula* φ*, if* $G \models^{\mathrm{GI}} \varphi$*, then* $\varphi \simeq \varphi_G$.

Proof. By induction on the structure of PP formulas. Case $\varphi \equiv \top$. By $\varphi_{\mathrm{G}_\top} \equiv \top * \top \simeq_{(*3)} \top$. Case $\varphi \equiv x = x$. By $\varphi_{\mathrm{G}_{x=x}} \simeq_{(*3)} \exists z.z = x \simeq_{(=4)} x = x$. Case $\varphi \equiv x = y$ where $x \neq y$. By $\varphi_{\mathrm{G}_{x=y}} \simeq_{(*3)} \exists z.z = x * z = y \simeq_{\text{Lemma 21(1)}} x = y$. Case $\varphi \equiv a(x_{f(1)}, \dots, x_{f(n)})$ where $f\colon [n] \to [k]$ is a surjective map for some k. Then,

$$\varphi_{\mathrm{G}_{a(x_{f(1)},\dots,x_{f(n)})}} \equiv \exists z_k \dots z_1.(\mathop{\ast}_{i=1}^{k} z_i = x_i) * a(z_{f(1)}, \dots, z_{f(n)})$$
$$\simeq_{\text{Lemma 21(1)}} \cdots \simeq_{\text{Lemma 21(1)}} a(z_{f(1)}, \dots, z_{f(n)})[x_1 \dots x_k / z_1 \dots z_k] \equiv \varphi.$$

Case $\varphi \equiv \varphi_1 * \varphi_2$. Let G_1 and G_2 be such that $G \cong G_1 \otimes G_2$, $G_1 \models^{\mathrm{GI}} \varphi_1$, $G_2 \models^{\mathrm{GI}} \varphi_2$. By I.H., $\varphi_1 \simeq \varphi_{G_1}$ and $\varphi_2 \simeq \varphi_{G_2}$. We denote them by $\varphi_{G_1} \equiv \exists z_1 \dots z_{n'}. \mathop{\ast}_{i=1}^{k'} z_{g_1(i)} = x_i * \mathop{\ast}_{i=1}^{m'} \tilde{\varphi}_i$ and $\varphi_{G_2} \equiv \exists z_{n'+1} \dots z_n. \mathop{\ast}_{i=1}^{k} z_{g_2(i)} = x_i * \mathop{\ast}_{i=m'+1}^{m} \tilde{\varphi}_i$, respectively. Here, $g_1\colon [k'] \to [n']$ and $g_2\colon [k] \to [n]$ are some maps. We assume, without loss of generality that $z_1, \dots, z_n$ are pairwise distinct and $k' \leq k$ (by swapping G_1 and G_2 appropriately using $(*2)$). Then,

$$\begin{aligned}
\varphi &\simeq_{\text{I.H.}} \varphi_{G_1} \otimes \varphi_{G_2} \\
&\simeq_{(\exists 1)(\exists 2)(*1)(*2)} \exists z_1 \dots z_n.(\mathop{\ast}_{i=1}^{k'} z_{g_1(i)} = x_i) * (\mathop{\ast}_{i=1}^{k} z_{g_2(i)} = x_i) * (\mathop{\ast}_{i=1}^{m} \tilde{\varphi}_i) \\
&\simeq_{(*1)(*2)(=3)} \exists z_1 \dots z_n.(\mathop{\ast}_{i=1}^{k'} z_{g_1(i)} = z_{g_2(i)}) * (\mathop{\ast}_{i=1}^{k} z_{g_2(i)} = x_i) * (\mathop{\ast}_{i=1}^{m} \tilde{\varphi}_i) \\
&\simeq_{\text{Lem. 21(2)}} \exists z_{I_1} \dots z_{I_m}.(\mathop{\ast}_{i=1}^{k} z_{[g_2(i)]_\sim} = x_i) * (\mathop{\ast}_{i=1}^{m} \tilde{\varphi}_i[[z_1]_\sim \dots [z_n]_\sim / z_1 \dots z_n]) \simeq \varphi_{G_1 \otimes G_2}
\end{aligned}$$

Here, $\sim$ and $I_1 \dots I_m$ the ones obtained from Lemma 21(2). Case $\varphi \equiv \exists y.\varphi_1$. Let G_1 be such that $G \cong G_1[\mathfrak{f}/y]$ and $G_1 \models \varphi_1$. By I.H., $\varphi_1 \simeq \varphi_{G_1}$. We denote it by $\varphi_{G_1} \equiv \exists z_1 \dots z_n. \mathop{\ast}_{i=1}^{k} z_{g(i)} = x_i * \mathop{\ast}_{i=1}^{m} \tilde{\varphi}_i$. Here, $g\colon [k] \to [n]$ is a map, and we assume, without loss of generality that $y, z_1, \dots, z_n$ are pairwise distinct. Then, $y = x_l$ for some $l \in [k]$ (note $y \in \mathsf{FV}_1(\varphi_1)$). We assume, without loss of generality by $(*1)(*2)$ that $y = x_k$. Then, $\varphi \simeq_{\text{I.H.}} \exists y.\varphi_{G_1} \simeq_{(\exists 1)(=1)\ \text{Lem. 21(1)}} \exists z_1 \dots z_n.(\mathop{\ast}_{i=1}^{k-1} z_{g(i)} = x_i) * (\mathop{\ast}_{i=1}^{m} \tilde{\varphi}_i) \simeq \varphi_G$. ◀

Proof of Theorem 19 for PP formulas. Assume $\psi \cong^{\mathrm{GI}} \rho$. By Proposition 17(1), $\mathcal{G}(\psi) = \mathcal{G}(\rho) = \{G\}^{\cong}$ for some G. Then, $\psi \simeq_{\text{Lemma 22}} \varphi_G \simeq_{\text{Lemma 22}} \rho$. ◀

In the following, we consider EP formulas.

▶ **Lemma 23.** *If* $\{G_1, \dots, G_n\}^{\cong} = \{H_1, \dots, H_m\}^{\cong}$, *then* $\varphi_{\langle G_i \rangle_{i=1}^n} \simeq \varphi_{\langle H_i \rangle_{i=1}^m}$.

Proof. By the assumption, let $f\colon [n] \to [m]$ be a map such that $G_i \cong H_{f(i)}$ for every $i \in [n]$. Then, $\varphi_{\langle G_i \rangle_{i=1}^n} \equiv \bigvee_{i=1}^n \varphi_{G_i} \simeq_{\text{Prop. 20}} \bigvee_{i=1}^n \varphi_{H_{f(i)}} \simeq_{(\vee 1)(\vee 2)(\vee 4)} \bigvee_{i=1}^m \varphi_{H_i} \equiv \varphi_{\langle H_i \rangle_{i=1}^m}$. ◀

▶ **Lemma 24.** *For all* $\varphi \in \mathsf{EP}_A^\tau$, *there exists some* $\langle \varphi_i \rangle_{i=1}^n \in (\mathsf{PP}_A^\tau)^*$ *such that* $\varphi \simeq \bigvee_{i=1}^n \varphi_i$.

Proof. By induction on the structure of φ. Case $\varphi \equiv \mathsf{ff}$. By letting $n = 0$. Case $\varphi \equiv \tilde{\varphi}$. By letting $n = 1$. Case $\varphi \equiv \varphi^{(1)} * \varphi^{(2)}$. For $l \in [2]$, let $\langle \varphi_i^{(l)} \rangle_{i=1}^{n_l}$ be the one obtained by I.H. w.r.t. $\varphi^{(l)}$. If $n_1 = 0$ or $n_2 = 0$, then $\varphi \simeq_{(*2)(\mathsf{ff})} \mathsf{ff}$. Otherwise, $\varphi \simeq_{(\vee 1)(\vee 2)(\vee 6)} \bigvee_{i=1}^{n_1} \bigvee_{j=1}^{n_2} (\varphi_i^{(1)} * \varphi_j^{(2)})$ (and apply $(\vee 1)(\vee 2)$). Case $\varphi \equiv \varphi^{(1)} \vee \varphi^{(2)}$. Let $\langle \varphi_i \rangle_{i=1}^{n'}$ and $\langle \varphi_i \rangle_{i=n'+1}^{n}$ be the ones obtained by I.H. w.r.t. $\varphi^{(1)}$ and $\varphi^{(2)}$, respectively. Then, $\varphi \simeq_{(\vee 1)(\vee 2)(\vee 3)} \bigvee_{i=1}^n \varphi_i$. Case $\varphi \equiv \exists x.\varphi^{(1)}$. Let $\langle \varphi_i^{(1)} \rangle_{i=1}^n$ be the one obtained by I.H. w.r.t. $\varphi^{(1)}$. If $n = 0$, then $\varphi \equiv \exists x.\mathsf{ff} \simeq_{(\mathsf{ff})} \exists x.\mathsf{ff} * \mathsf{ff} \simeq_{(\exists 2)} (\exists x.\mathsf{ff}) * \mathsf{ff} \simeq_{(*2)(\mathsf{ff})} \mathsf{ff}$. Otherwise, $\varphi \equiv \exists x. \bigvee_{i=1}^n \varphi_i^{(1)} \simeq_{(\vee 5)} \bigvee_{i=1}^n \exists x.\varphi_i^{(1)}$. ◀

▶ **Lemma 25.** *For every* EP *formula* φ *and finite sequence* $\vec{G}$ *s.t.* $\mathcal{G}(\varphi) = \mathrm{Occ}(\vec{G})^{\cong}$, $\varphi \simeq \varphi_{\vec{G}}$.

Proof. By $\varphi \simeq_{\text{Lemma 24}} \bigvee_{i=1}^{n} \varphi_i \simeq_{\text{Lemma 22}} \bigvee_{i=1}^{n} \varphi_{G_i} \simeq_{\text{Lemma 23}} \varphi_{\vec{G}}$. Here, for each $i \in [n]$, φ_i is a PP formula and G_i is a graph such that $G_i \models^{\mathrm{GI}} \varphi_i$. ◀

Proof of Theorem 19 for EP formulas. Assume $\psi \cong^{\mathrm{GI}} \rho$. Let $\vec{G}$ be a finite sequence such that $\mathcal{G}(\psi) = \mathcal{G}(\rho) = \mathrm{Occ}(\vec{G})^{\cong}$. Then, $\psi \simeq_{\text{Lemma 25}} \varphi_{\vec{G}} \simeq_{\text{Lemma 25}} \rho$. ◀

5 Kleene Theorems Between EPs and HRGs

In this section, we show that EP(LFP) (resp. EP(TC)) has the same expressive power as the class of HRGs (resp. linear HRGs). To this end, we introduce *term (formula) rewriting systems* [2] (FRSs) and show the equivalence above via FRSs. Intuitively, FRSs play the same role as finite automata with transitions labelled by regular expressions [7] (so-called *extended finite automata*) in translating finite automata into regular expressions.[4]

5.1 Formula Rewriting Systems (FRSs)

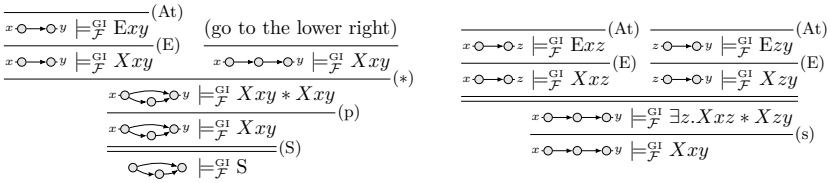
▶ **Definition 26.** *A* formula rewriting system (FRS[$\mathscr{C}$]) $\mathcal{F}$ *over an ordinal-typed alphabet* A *is a tuple* $\langle \mathcal{X}^{\mathcal{F}}, \mathcal{R}^{\mathcal{F}}, \mathfrak{s}^{\mathcal{F}} \rangle$, *where* $\mathcal{X}^{\mathcal{F}}$ *is an ordinal-typed alphabet disjoint with* A *for denoting* (non-terminal) labels, $\mathcal{R}^{\mathcal{F}}$ *is a finite set of pairs* $r = \langle X\vec{x}, \varphi \rangle$ *(written* $X\vec{x} \leftarrow \varphi$*) of a strictly atomic* $\mathcal{X}^{\mathcal{F}}$*-formula* $X\vec{x}$ *and a* $\mathscr{C}^{\mathrm{Occ}(\vec{x})}_{A \cup \mathcal{X}^{\mathcal{F}}}$*-formula* φ *for denoting* rewriting rules, *and* $\mathfrak{s}^{\mathcal{F}}$ *is a strictly atomic* $\mathcal{X}^{\mathcal{F}}$*-formula for denoting the* source formula. *Here, for an ordinal-typed alphabet* $\mathcal{X}$, *we say that* φ *is a* strictly atomic $\mathcal{X}$*-formula if* φ *is of the form* $X\vec{x}$, *where* $X \in \mathcal{X}$ *and the elements of* $\vec{x}$ *are pairwise distinct.*

▶ **Definition 27.** *For an* FRS[$\mathscr{C}$] $\mathcal{F} = \langle \mathcal{X}, \mathcal{R}, \mathfrak{s} \rangle$ *over an ordinal-typed alphabet* A, *the binary relation* $\models^{\mathrm{GI}}_{\mathcal{F}} \subseteq \bigcup_{\tau \subseteq \mathcal{Y}_1;\, \mathcal{X} \subseteq A \cup \mathcal{Y}_2} \mathsf{GR}^{\tau}_{\mathcal{X}} \times \mathsf{Fml}^{\tau}_{\mathcal{X}}$ *is defined as the least* $\cong$*-closed relation closed under all the rules of* $\models^{\mathrm{GI}}$ *(in Definition 14) and the following rule: If* $X\vec{x} \leftarrow \varphi \in \mathcal{R}$, *then* $\dfrac{G \models^{\mathrm{GI}}_{\mathcal{F}} \varphi[\vec{y}/\vec{x}]}{G \models^{\mathrm{GI}}_{\mathcal{F}} X\vec{y}}$. *We write* $G \models^{\mathrm{GI}} \mathcal{F}$ *for* $G \models^{\mathrm{GI}}_{\mathcal{F}} \mathfrak{s}$. *The graph language of* $\mathcal{F}$ *is defined by* $\mathcal{G}(\mathcal{F}) \triangleq \{G \mid G \models^{\mathrm{GI}} \mathcal{F}\}$.

▶ **Example 28** (cf. Example 12)**.** Let $\mathcal{F}$ be the FRS[PP] over A_{E}, defined by $\mathrm{ty}^{\mathcal{X}^{\mathcal{F}}} = \{\mathtt{S} \mapsto [0], X \mapsto [2]\}$, $\mathcal{R}^{\mathcal{F}} = \{(\mathtt{S}), (\mathrm{E}), (\mathrm{s}), (\mathrm{p})\}$, $\mathfrak{s}^{\mathcal{F}} = \mathtt{S}$, where each rule in $\mathcal{R}^{\mathcal{F}}$ is as follows:

$$(\mathtt{S})\ \mathtt{S} \leftarrow \exists xy.Xxy \quad (\mathrm{E})\ Xxy \leftarrow \mathrm{E}xy \quad (\mathrm{s})\ Xxy \leftarrow \exists z.Xxz * Xzy \quad (\mathrm{p})\ Xxy \leftarrow Xxy * Xxy$$

Then, $\mathcal{G}(\mathcal{F})$ is the set of all series-parallel graphs. For example,  $\models^{\mathrm{GI}} \mathcal{F}$ is shown by:[5]



In general, the following proposition is immediate from the translations between graphs and PP formulas in Proposition 17(1). Also, we use *linear*/*(n-)recursive* for FRS[PP]s in the same manner as for HRGs.

[4] FRS[$\mathscr{C}$] is essentially the same as *positive Datalog* [20, Section 9] if $\mathscr{C}$ is the class of conjunctive queries.
[5] Double line denotes that 0 or more rules are applied in the place.

▶ **Proposition 29.** *For every $\mathcal{G}$, some HRG (resp. linear HRG) recognizes $\mathcal{G}$ iff some* FRS[PP] *(resp. linear* FRS[PP]*) recognizes $\mathcal{G}$.*

An FRS $\mathcal{F}$ is *deterministic* if for every $X \in \mathcal{X}^{\mathcal{F}}$, the number of rules of the form $X\vec{x} \leftarrow \varphi$ is at most one. In Example 28, we can put together the three rules for X as follows in FRS[EP]:

$$(\mathtt{S})\ \mathtt{S} \leftarrow \exists xy.Xxy \qquad (X)\ Xxy \leftarrow (\mathrm{E}xy) \vee (Xxy * Xxy) \vee (\exists z.Xxz * Xzy).$$

▶ **Proposition 30.** *For every $\mathcal{G}$,* (i) *some* FRS[PP] *recognizes $\mathcal{G}$ iff* (ii) *some deterministic* FRS[EP] *recognizes $\mathcal{G}$ iff* (iii) *some* FRS[EP] *recognizes $\mathcal{G}$.*

Proof. (i) ⇒ (ii): By the same argument as above. (ii) ⇒ (iii): Trivial. (iii) ⇒ (i): By replacing each rule $X\vec{x} \leftarrow \varphi$ with $X\vec{x} \leftarrow \psi_1, \dots, X\vec{x} \leftarrow \psi_n$. Here, $\psi_1, \dots \psi_n$ are PP formulas such that $\varphi \cong^{\mathrm{GI}} \bigvee_{i=1}^{n} \psi_i$ (Lemma 24). ◀

The following are useful properties of hyperedge replacing and glueing.

▶ **Proposition 31.** *For every* FRS[EP(LFP)] $\mathcal{F}$*: (1): If there is a derivation tree that shows $G \models^{\mathrm{GI}}_{\mathcal{F}} \varphi$ from the assumptions $\langle H_i \models^{\mathrm{GI}}_{\mathcal{F}} \psi_i \rangle_{i=1}^{n}$ and $H_1, \dots, H_n$ don't contain any $\mathsf{FV}_2(\varphi)$-labelled edges and have an ordinal type, then there exist some G' and $e_1 \dots e_n$ such that $G \cong G'[H_1 \dots H_n / e_1 \dots e_n]$. (2): If there is a derivation tree that shows $G[H_1 \dots H_n/\vec{e}] \models^{\mathrm{GI}}_{\mathcal{F}} \varphi$ from the assumptions $\langle H_i \models^{\mathrm{GI}}_{\mathcal{F}} \psi_i \rangle_{i=1}^{n}$ and $H_1, \dots, H_n, H'_1, \dots, H'_n$ don't contain any $\mathsf{FV}_2(\varphi)$-labelled edges and have an ordinal type, then there is a derivation tree that shows $G[H'_1 \dots H'_n/\vec{e}] \models^{\mathrm{GI}}_{\mathcal{F}} \varphi$ from the assumptions $\langle H'_i \models^{\mathrm{GI}}_{\mathcal{F}} \psi_i \rangle_{i=1}^{n}$. For every* FRS[EP(TC)] $\mathcal{F}$*: (3): If there is a derivation tree that shows $G \models^{\mathrm{GI}}_{\mathcal{F}} \varphi$ from $H \models^{\mathrm{GI}}_{\mathcal{F}} \psi$ and $\mathrm{ty}(H) \cap \mathsf{BV}_1(\varphi) = \emptyset$, then there exist some G' such that $G \cong G' \otimes H$. (4): If there is a derivation tree that shows $G \otimes H \models^{\mathrm{GI}}_{\mathcal{F}} \varphi$ from $G' \otimes H \models^{\mathrm{GI}}_{\mathcal{F}} \psi$, $\mathrm{ty}(H) \cap \mathsf{BV}_1(\varphi) = \emptyset$, $\mathrm{ty}(H') \cap \mathsf{BV}_1(\varphi) = \emptyset$, and $\mathrm{ty}(H) = \mathrm{ty}(H')$, then there is a derivation tree that shows $G \otimes H' \models^{\mathrm{GI}}_{\mathcal{F}} \varphi$ from $G' \otimes H' \models^{\mathrm{GI}}_{\mathcal{F}} \psi$.*

Proof Sketch. By a straightforward induction on the structure of the derivation tree using Proposition 9. See [33, Appendix B] for more details. ◀

5.2 Equivalence of EP(LFP) formulas and HRGs (Theorem 1(2))

In the following, by using Proposition 29 and 30, we show that EP(LFP) has the same expressive power as (deterministic) FRS[EP].

From EP(LFP) formulas to FRS[EP]s. We say that an EP(LFP) formula φ is *simple* if (a) all the second-order variables X occurring in the form $[\mathsf{LFP}_{\vec{x},X}(\varphi)]\vec{y}$ are pairwise distinct, (b) $\vec{x} = \vec{y} = \iota$ for each subformula of the form $[\mathsf{LFP}_{\vec{x},X}(\varphi)]\vec{y}$, and (c) $\vec{x} = \iota$ for each subformula of the form $X\vec{x}$. This restriction simplifies the translation and the proof.

▶ **Lemma 32.** *Every* EP(LFP) *formula φ has a GI-equivalent simple* EP(LFP) *formula.*

Proof Sketch. For (a), rename variables appropriately. For (b)(c), use the following translations, respectively: $[\mathsf{LFP}_{\vec{x},X}(\varphi)]\vec{y} \rightsquigarrow \exists \vec{z}.\vec{z} = \vec{y} * \exists \iota.\iota = \vec{z} * [\mathsf{LFP}_{\iota,X}(\exists \vec{z}.\vec{z} = \iota * \exists \vec{x}.\vec{x} = \vec{z} * \varphi)]\iota$ and $X\vec{x} \rightsquigarrow \exists \vec{z}.\vec{z} = \vec{x} * \exists \iota.\iota = \vec{z} * X\iota$. Here, $\vec{z}$ is a sequence of fresh variables. ◀

Let $\vec{z}_\bullet$ be a map from each EP(LFP) formula φ to a permutation $\vec{z}_\varphi$ of $\mathsf{FV}_1(\varphi)$. Figure 4 gives a translation from a simple EP(LFP) formula φ into an FRS[EP] $\mathcal{F}_\varphi = \langle \mathcal{X}_\varphi, \mathcal{R}_\varphi, \mathfrak{s}_\varphi \rangle$.[6]

[6] This translation is essentially the same as the translation from existential fixpoint logic to Datalog, see, e.g., [20, Theorem 9.1.4]. The only difference is the semantics.

$$\mathcal{F}_{\tilde{\varphi}} \triangleq \langle \{\mathtt{S}_\varphi\}, \{\mathfrak{s}_\varphi \leftarrow \tilde{\varphi}\}, \mathtt{S}_\varphi \vec{z}_\varphi \rangle \quad \mathcal{F}_{\exists x.\psi} \triangleq \langle \{\mathtt{S}_\varphi\} \cup \mathcal{X}_\psi, \{\mathfrak{s}_\varphi \leftarrow \exists x.\mathfrak{s}_\psi\} \cup \mathcal{R}_\psi, \mathtt{S}_\varphi \vec{z}_\varphi \rangle$$
$$\mathcal{F}_{\psi \bullet \rho} \triangleq \langle \{\mathtt{S}_\varphi\} \cup \mathcal{X}_\psi \cup \mathcal{X}_\rho, \{\mathfrak{s}_\varphi \leftarrow \mathfrak{s}_\psi \bullet \mathfrak{s}_\rho\} \cup \mathcal{R}_\psi \cup \mathcal{R}_\rho, \mathtt{S}_\varphi \vec{z}_\varphi \rangle \quad (\bullet \in \{*, \vee\})$$
$$\mathcal{F}_{[\mathsf{LFP}_{\iota,X}(\psi)]\iota} \triangleq \langle \{\mathtt{S}_\varphi, X\} \cup \mathcal{X}_\psi, \{\mathfrak{s}_\varphi \leftarrow X\iota,\ X\iota \leftarrow \mathfrak{s}_\psi\} \cup \mathcal{R}_\psi, \mathtt{S}_\varphi \vec{z}_\varphi \rangle$$

Figure 4 A translation from EP(LFP) formulas into (deterministic) FRS[EP]s.

▶ **Lemma 33.** *For every simple* EP(LFP) *formula* φ, $\mathcal{G}(\varphi) = \mathcal{G}(\mathcal{F}_\varphi)$.

Proof. $G \models^{\mathrm{GI}} \varphi \Rightarrow G \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi$: By induction on the size of the derivation tree of $G \models^{\mathrm{GI}} \varphi$. The only nontrivial case is when the last derivation rule is (LFP). Let $\varphi = [\mathsf{LFP}_{\iota,X}(\psi)]\iota$ (by the condition (b)) and let $G \cong H[G_1 \dots G_n / e_1 \dots e_n]$ be such that $H \models^{\mathrm{GI}} \psi$ and $G_i \models^{\mathrm{GI}} \varphi$ for $i \in [n]$. By I.H., $H \models^{\mathrm{GI}}_{\mathcal{F}_\psi} \mathfrak{s}_\psi$. Its derivation tree forms the left-hand side in the following (by the condition (c)). Also for $i \in [n]$, by I.H., $G_i \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi$, so by the construction of $\mathcal{F}_\varphi$, ($\heartsuit$-i) $G_i \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} X\iota$. Then, $G \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi$ is shown by the right-hand side tree (Proposition 31(2)).

$$\frac{\dfrac{}{\mathrm{G}_{X\iota} \models^{\mathrm{GI}}_{\mathcal{F}_\psi} X\iota} \quad \dots \quad \dfrac{}{\mathrm{G}_{X\iota} \models^{\mathrm{GI}}_{\mathcal{F}_\psi} X\iota}}{\vdots\ (\spadesuit)}}{H \models^{\mathrm{GI}}_{\mathcal{F}_\psi} \mathfrak{s}_\psi} \quad \rightsquigarrow \quad \frac{\dfrac{\dfrac{}{\vdots\ (\heartsuit\text{-}1)} \quad \dots \quad \dfrac{}{\vdots\ (\heartsuit\text{-}n)}}{\mathrm{G}_{X\iota}[G_1/\mathrm{e}] \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} X\iota \quad \dots \quad \mathrm{G}_{X\iota}[G_n/\mathrm{e}] \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} X\iota}}{\vdots\ (\spadesuit)}}{\dfrac{H[G_1 \dots G_n/e_1 \dots e_n] \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\psi}{H[G_1 \dots G_n/e_1 \dots e_n] \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi}(\mathfrak{s}_\varphi \leftarrow X\iota)(X\iota \leftarrow \mathfrak{s}_\psi)}}$$

$G \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi \Rightarrow G \models^{\mathrm{GI}} \varphi$: By induction on the size of the derivation tree of $G \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi$. We do a case analysis on the structure of φ. The only nontrivial case is when $\varphi = [\mathsf{LFP}_{\iota,X}(\psi)]\iota$. The derivation tree of $G \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi$ should form the right-hand side above, where the rule for X is not applied in ($\spadesuit$). Note that $G \cong H[G_1 \dots G_n / e_1 \dots e_n]$ for some H and $e_1 \dots e_n$ (Proposition 31(1)). Then, from the derivation tree, we can obtain the derivation tree of the form on the left-hand side above (Proposition 31(2)). Thus by I.H., $H \models^{\mathrm{GI}} \psi$. Also by using ($\heartsuit$-$i$), $G_i \models^{\mathrm{GI}}_{\mathcal{F}_\varphi} \mathfrak{s}_\varphi$, and thus by I.H., $G_i \models^{\mathrm{GI}} \varphi$. Hence, $G \models^{\mathrm{GI}} \varphi$. ◀

Proof of Theorem 1(2)⇒. By Lemma 32 and 33 (with Proposition 29 and 30). ◀

From FRS[EP]s to EP(LFP) formulas. This part is shown by folding non-terminal labels for a given *deterministic* FRS[EP] as follows: for non-0-recursive labels X, replace each occurrence of X with the formula corresponding to X in the rule; for 0-recursive labels, use the LFP. Note that by Proposition 30, from an FRS[EP], we can obtain a deterministic one.

▶ **Lemma 34.** *Every deterministic* FRS[EP(LFP)] *has a GI-equivalent* EP(LFP) *formula.*

Proof. Let $\mathcal{F} = \langle \mathcal{X}, \mathcal{R}, \mathtt{S}\vec{z} \rangle$. Let $\#_{\mathrm{n}}(\mathcal{F}) \triangleq \#(\mathcal{X} \setminus \{\mathtt{S}\})$ and $\#_{\mathrm{r}}(\mathcal{F})$ be the number of 0-recursive labels in $\mathcal{F}$. We prove by induction on the pair $\langle \#_{\mathrm{n}}(\mathcal{F}), \#_{\mathrm{r}}(\mathcal{F}) \rangle$. Case $\#_{\mathrm{n}}(\mathcal{F}) = \#_{\mathrm{r}}(\mathcal{F}) = 0$. Let $\mathcal{R} = \{\mathtt{S}\vec{x} \leftarrow \psi\}$. Then, $\mathcal{G}(\mathcal{F}) = \mathcal{G}(\psi[\vec{z}/\vec{x}])$. Case $\#_{\mathrm{n}}(\mathcal{F}) > \#_{\mathrm{r}}(\mathcal{F})$. Then, there exists a non-0-recursive label $X_0 \in \mathcal{X} \setminus \{\mathtt{S}\}$. Let $X_0\vec{x}_0 \leftarrow \psi_0 \in \mathcal{R}$. Let $\mathcal{F}' \triangleq \langle \mathcal{X} \setminus \{X_0\}, \{X\vec{x} \leftarrow \psi[\psi_0[-/\vec{x}_0]/X_0-] \mid X\vec{x} \leftarrow \psi \in \mathcal{R}, X \neq X_0\}, \mathtt{S}\vec{z} \rangle$, where $\psi[\psi_0[-/\vec{x}_0]/X_0-]$ denotes the formula ψ in which each $X_0\vec{y}$ has been replaced with $\psi_0[\vec{y}/\vec{x}_0]$. Then, $\mathcal{G}(\mathcal{F}) = \mathcal{G}(\mathcal{F}')$ because there is a trivial transformation between derivation trees of $\mathcal{F}$ and those of $\mathcal{F}'$. Also by I.H., there exists an EP(LFP) formula φ such that $\mathcal{G}(\mathcal{F}') = \mathcal{G}(\varphi)$. Hence, $\mathcal{G}(\mathcal{F}) = \mathcal{G}(\varphi)$. For the other case (i.e., $\#_{\mathrm{r}}(\mathcal{F}) \geq 1$), there exists a 0-recursive label $X_0 \in \mathcal{X}$. Let

$X_0\vec{x}_0 \leftarrow \psi_0 \in \mathcal{R}$. Let $\mathcal{F}' \triangleq \langle \mathcal{X}, \{X\vec{x} \leftarrow \psi \in \mathcal{R} \mid X \neq X_0\} \cup \{X_0\vec{x}_0 \leftarrow [\mathsf{LFP}_{\vec{x}_0,X_0}(\psi_0)]\vec{x}_0\}, \mathtt{S}\vec{z}\rangle$. Then, $\mathcal{G}(\mathcal{F}) = \mathcal{G}(\mathcal{F}')$ because there exists a transformation between derivation trees of $\mathcal{F}'$ and those of $\mathcal{F}$ in the same manner as the proof of Lemma 33. Also by I.H., there exists an $\mathsf{EP}(\mathsf{LFP})$ formula φ such that $\mathcal{G}(\mathcal{F}') = \mathcal{G}(\varphi)$. Hence, $\mathcal{G}(\mathcal{F}) = \mathcal{G}(\varphi)$. ◀

Proof of Theorem 1(2)⇐. By Lemma 34 (with Proposition 29 and 30). ◀

5.3 Equivalence of EP(TC) formulas and linear HRGs (Theorem 1(3)).

In the following, by using Proposition 29 and 30, we show that $\mathsf{EP}(\mathsf{TC})$ has the same expressive power as the class of linear FRS[PP].

From EP(TC) formulas to linear FRS[PP]s. We say that an $\mathsf{EP}(\mathsf{TC})$ formula φ is *simple* if all the variables x occurring in the form $\exists x.\psi$, the variables in $\vec{x}\vec{y}\vec{u}\vec{w}$ occurring in the form $[\varphi]^+_{\vec{x}\vec{y}}\vec{u}\vec{w}$, and the free variables in φ are pairwise distinct. As with Lemma 32, from a given $\mathsf{EP}(\mathsf{TC})$ formula, we can obtain a GI-equivalent simple one by renaming variables and using the following translation: $[\varphi]^+_{\vec{x}\vec{y}}\vec{u}\vec{w} \rightsquigarrow \exists\vec{z}.\vec{z} = \vec{u}\vec{w} * [\varphi[\vec{z}'/\vec{x}\vec{y}]]^+_{\vec{z}'}\vec{z}$. Here, elements of $\vec{z}$ and $\vec{z}'$ are fresh variables. Furthermore, the following holds.

▶ **Lemma 35.** *Every* $\mathsf{EP}(\mathsf{TC})$ *formula* φ *has a GI-equivalent simple* $\mathsf{EP}(\mathsf{TC})$ *formula of the form* $\exists z_0.\varphi_0$ *or* $\top \vee \exists z_0.\varphi_0$.

Proof. If $\mathsf{FV}_1(\varphi) \neq \emptyset$, then $\varphi \cong^{\mathrm{GI}} \exists z_0.z_0 = x * \varphi$, where $x \in \mathsf{FV}_1(\varphi)$ and z_0 is a fresh variable. Otherwise, let $\bigvee_{i=1}^n \varphi_i$ be a disjunctive normal form of φ, where each φ_i is a prenex normal form $\mathsf{EP}(\mathsf{TC})$ formula. Let $\rho_i \equiv \exists z_0.\psi_i$ if φ_i is of the form $\exists x.\psi_i$ and $\rho_i \equiv \top$ otherwise (note that then $\varphi_i \equiv \top$ should because $\mathsf{FV}_1(\varphi_i) = \emptyset$). Note that $\varphi_i \cong^{\mathrm{GI}} \rho_i$. Let $l_1 \dots l_m$ be the subsequence of ι_n such that for each $i \in [n]$, $i \in \{l_1, \dots, l_m\}$ iff $\rho_i \not\equiv \top$. If $m < n$, then $\varphi \cong^{\mathrm{GI}} \top \vee \bigvee_{j=1}^m \exists z_0.\psi_{l_j}$ ($\cong^{\mathrm{GI}} \top \vee \exists z_0.\bigvee_{j=1}^m \psi_{l_j}$). Otherwise, $\varphi \cong^{\mathrm{GI}} \bigvee_{i=1}^n \exists z_0.\psi_i$ ($\cong^{\mathrm{GI}} \exists z_0.\bigvee_{i=1}^n \psi_i$). Hence, it has been proved. ◀

Let $\vec{z}$ be a sequence of pairwise distinct variables. For a simple $\mathsf{EP}(\mathsf{TC})$ formula φ such that $\mathsf{V}_1(\varphi) \subseteq \mathrm{Occ}(\vec{z})$, we define the linear FRS[PP] $\dot{\mathcal{F}}_\varphi = \langle \mathcal{X}_\varphi, \mathcal{R}_\varphi, \mathfrak{s}_\varphi\rangle$ (we may explicitly write $\dot{\mathcal{F}}^{\vec{z}}_\varphi = \langle \mathcal{X}^{\vec{z}}_\varphi, \mathcal{R}^{\vec{z}}_\varphi, \mathfrak{s}^{\vec{z}}_\varphi\rangle$) in Figure 5. Our construction is based on Thompson's construction [42] and the product construction (in translating regular expressions into finite automata), but is generalized for first-order variables.

$$
\begin{aligned}
\dot{\mathcal{F}}_{\bar{\varphi}} &\triangleq \langle \{\mathtt{S}_\varphi, \mathtt{T}_\varphi\}, \{\mathtt{S}_\varphi\vec{z} \leftarrow \bar{\varphi} * \mathtt{T}_\varphi\vec{z}\}, \mathtt{S}_\varphi\vec{z}\rangle \\
\dot{\mathcal{F}}_{\exists x.\psi} &\triangleq \langle \{\mathtt{S}_\varphi, \mathtt{T}_\varphi\} \cup \mathcal{X}_\psi, \{\mathtt{S}_\varphi\vec{z} \leftarrow x = x * \exists x.\mathtt{S}_\psi\vec{z}, \mathtt{T}_\psi\vec{z} \leftarrow \mathtt{T}_\varphi\vec{z}\} \cup \mathcal{R}_\psi, \mathtt{S}_\varphi\vec{z}\rangle \\
\dot{\mathcal{F}}_{\psi * \rho} &\triangleq \langle \{\mathtt{S}_\varphi, \mathtt{T}_\varphi\} \cup (\mathcal{X}_\psi \times \mathcal{X}_\rho), \{\mathtt{S}_\varphi\vec{z} \leftarrow \langle \mathtt{S}_\psi, \mathtt{S}_\rho\rangle\vec{z}, \langle \mathtt{T}_\psi, \mathtt{T}_\rho\rangle\vec{z} \leftarrow \mathtt{T}_\varphi\vec{z}\} \cup \\
&\qquad \{r[\langle -, Y\rangle/-] \mid r \in \mathcal{R}_\psi, Y \in \mathcal{X}_\rho\} \cup \{r[\langle X, -\rangle/-] \mid r \in \mathcal{R}_\rho, X \in \mathcal{X}_\psi\}, \mathtt{S}_\varphi\vec{z}\rangle^{\dagger 1} \\
\dot{\mathcal{F}}_{\psi \vee \rho} &\triangleq \langle \{\mathtt{S}_\varphi, \mathtt{T}_\varphi\} \cup \mathcal{X}_\psi \cup \mathcal{X}_\rho, \{\mathtt{S}_\varphi\vec{z} \leftarrow \mathtt{S}_\psi\vec{z}, \mathtt{S}_\varphi\vec{z} \leftarrow \mathtt{S}_\rho\vec{z}, \mathtt{T}_\psi\vec{z} \leftarrow \mathtt{T}_\varphi\vec{z}, \mathtt{T}_\rho\vec{z} \leftarrow \mathtt{T}_\varphi\vec{z}\} \cup \mathcal{R}_\psi \cup \mathcal{R}_\rho, \mathtt{S}_\varphi\vec{z}\rangle \\
\dot{\mathcal{F}}_{[\psi]^+_{\vec{x}\vec{y}}\vec{u}\vec{w}} &\triangleq \langle \{\mathtt{S}_\varphi, \mathtt{T}_\varphi\} \cup \mathcal{X}_\psi, \{\mathtt{S}_\varphi\vec{z} \leftarrow \vec{x}\vec{y} = \vec{x}\vec{y} * \exists\vec{x}.\vec{x} = \vec{u} * \exists\vec{y}.\mathtt{S}_\psi\vec{z}\} \cup \\
&\qquad \{\mathtt{T}_\psi\vec{z} \leftarrow \vec{x}\vec{y} = \vec{x}\vec{y} * \exists\vec{x}.\vec{x} = \vec{y} * \exists\vec{y}.\mathtt{S}_\psi\vec{z}, \mathtt{T}_\psi\vec{z} \leftarrow \vec{y} = \vec{w} * \mathtt{T}_\varphi\vec{z}\} \cup \mathcal{R}_\psi, \mathtt{S}_\varphi\vec{z}\rangle
\end{aligned}
$$

†1: $r[\langle -, Y\rangle/-]$ (resp. $r[\langle X, -\rangle/-]$) is the rule r in which each X (resp. Y) has been replaced with $\langle X, Y\rangle$.

Figure 5 Definition of linear FRS[PP] $\dot{\mathcal{F}}_\varphi$.

▶ **Lemma 36.** *For every simple* EP(TC) *formula* φ *and every* $G \in \mathsf{GR}^{\tau}_{A}$ *(where* $\varphi \in \mathsf{Fml}^{\tau}_{A}$*),* $G \models^{\mathrm{GI}} \varphi$ *iff there is a derivation tree that shows* $G \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{S}_{\varphi}\vec{z}$ *from* $\mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{T}_{\varphi}\vec{z}$.

Proof. $\Rightarrow$: By induction on the structure of φ. The essential case is when $\varphi = [\psi]^{+}_{\vec{x}\vec{y}}\vec{u}\vec{w}$. Let $G \cong (G_1 \odot_{\vec{y}\vec{x}} \dots \odot_{\vec{y}\vec{x}} G_n)[\vec{u}\vec{w}/\vec{x}\vec{y}]$ be such that $G_i \models^{\mathrm{GI}} \psi$ for $i \in [n]$. For notational simplicity, let $G_{[i,n]} \triangleq G_i \odot_{\vec{y}\vec{x}} \dots \odot_{\vec{y}\vec{x}} G_n[\vec{w}/\vec{y}]$ for $i \in [n]$. Note that $G \cong G_{[1,n]}[\vec{u}/\vec{x}]$ and $G_{[i,n]} \cong (G_i \otimes G_{[i+1,n]}[\vec{y}/\vec{x}])[\mathtt{f}\dots\mathtt{f}/\vec{y}]$. For each $i \in [n]$, by I.H., there is a derivation tree (♣-i) that shows $G_i \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\psi}} \mathtt{S}_{\psi}\vec{z}$ from $\mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\psi}} \mathtt{T}_{\psi}\vec{z}$. Then, we obtain a derivation tree that shows $G \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{S}_{\varphi}\vec{z}$ from $\mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{T}_{\varphi}\vec{z}$ by concatenating (♣-1)-(♣-n) using Proposition 31(4) as follows.

$$\dfrac{\dfrac{\dfrac{\text{(go to the lower right)}}{G_{[2,n]}[\vec{y}/\vec{x}] \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{T}_{\psi}\vec{z}}}{\vdots\ (\clubsuit\text{-}1)}}{G_1 \otimes G_{[2,n]}[\vec{y}/\vec{x}] \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{S}_{\psi}\vec{z}} \quad \dfrac{}{G_{[1,n]}[\vec{u}/\vec{x}] \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{S}_{\varphi}\vec{z}} \quad \dots \quad \dfrac{\dfrac{\dfrac{\text{(go to the lower right)}}{\mathrm{G}_{\vec{y}=\vec{w}} \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{T}_{\psi}\vec{z}}}{\vdots\ (\clubsuit\text{-}n)}}{G_n \otimes \mathrm{G}_{\vec{y}=\vec{w}} \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{S}_{\psi}\vec{z}} \quad \dfrac{}{G_{[n,n]}[\vec{y}/\vec{x}] \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{T}_{\psi}\vec{z}} \quad \dfrac{\mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{T}_{\varphi}\vec{z}}{\mathrm{G}_{\vec{y}=\vec{w}} \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi}} \mathtt{T}_{\psi}\vec{z}}\,.$$

$\Leftarrow$: By induction on the structure of φ. We do case analysis on the structure of φ. The essential case is when $\varphi = [\psi]^{+}_{\vec{x}\vec{y}}\vec{u}\vec{w}$. Then, the derivation tree should be of the form like the above (by using Proposition 31(3)), where the rules for $\mathtt{T}_{\psi}$ are not applied in each (♣-i). Then by Proposition 31(4), each (♣-i) also shows $G_i \otimes \mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\psi}} \mathtt{S}_{\psi}\vec{z}$ from $\mathrm{G}^{\mathrm{Occ}(\vec{z})}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\psi}} \mathtt{T}_{\psi}\vec{z}$. By I.H., $G_i \models^{\mathrm{GI}} \psi$. Thus, $G \models^{\mathrm{GI}} \varphi$. ◀

▶ **Lemma 37.** *Every simple* EP(TC) *formula of the form* $\exists z_0.\varphi_0$ *or* $\top \vee \exists z_0.\varphi_0$ *has a GI-equivalent linear* FRS[PP].

Proof. We only write the case of $\exists z_0.\varphi_0$ (the case of $\top \vee \exists z_0.\varphi_0$ is shown in the same way). Let us recall the linear FRS[PP] $\dot{\mathcal{F}}^{\vec{z}}_{\varphi_0} = \langle \mathcal{X}^{\vec{z}}_{\varphi_0}, \mathcal{R}^{\vec{z}}_{\varphi_0}, \mathfrak{s}^{\vec{z}}_{\varphi_0} \rangle$ in Figure 5, where $\vec{z}'z_0 \in \mathrm{Perm}(\mathsf{FV}_1(\varphi_0))$, $\vec{z}'' \in \mathrm{Perm}(\mathsf{BV}_1(\varphi_0))$, and $\vec{z} = \vec{z}'z_0\vec{z}''$. Let $\bar{\mathcal{F}}$ be the linear FRS[PP] $\langle \{\mathtt{S}\} \cup \mathcal{X}^{\vec{z}}_{\varphi_0}, \{\mathtt{S}\vec{z}' \leftarrow \exists z_0.\mathtt{S}_{\varphi_0}\vec{z}'z_0 \dots z_0, \mathtt{T}_{\varphi_0}\vec{z} \leftarrow \vec{z} = \vec{z}\} \cup \mathcal{R}^{\vec{z}}_{\varphi_0}, \mathtt{S}\vec{z}' \rangle$. Then, $G[\mathtt{f}/z_0] \models^{\mathrm{GI}} \exists z_0.\varphi_0$ iff $G \models^{\mathrm{GI}} \varphi_0$ iff there exists a derivation tree that shows $G \otimes \mathrm{G}^{\vec{z}}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi_0}} \mathtt{S}_{\varphi_0}\vec{z}$ from $\mathrm{G}^{\vec{z}}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi_0}} \mathtt{T}_{\varphi_0}\vec{z}$ (Lemma 36) iff there exists a derivation tree that shows $G \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi_0}} \mathtt{S}_{\varphi_0}\vec{z}'z_0 \dots z_0$ from $\mathrm{G}^{\vec{z}}_{\top} \models^{\mathrm{GI}}_{\dot{\mathcal{F}}^{\vec{z}}_{\varphi_0}} \mathtt{T}_{\varphi_0}\vec{z}$ (because the name differences in the part $\vec{z}''$ do not affect to the construction of the derivation tree by $\vec{z}'' \in \mathrm{Perm}(\mathsf{BV}_1(\varphi_0))$) iff $G[\mathtt{f}/z_0] \models^{\mathrm{GI}}_{\bar{\mathcal{F}}} \mathtt{S}\vec{z}'$. Hence, $\mathcal{G}(\bar{\mathcal{F}}) = \mathcal{G}(\exists z_0.\varphi_0)$. ◀

Proof of Theorem 1(3)$\Rightarrow$. By Lemma 35 and 37 (with Proposition 29 and 30). ◀

From linear FRS[PP]s to EP(TC) formulas. This part is shown by generalizing *the state elimination method* in finite automata theory for linear FRS[PP]s. To this end, we introduce the following class based on transitions in finite automata. We say that an FRS[EP(TC)] $\mathcal{F}$ is *FA-linear* if (a) there is a non-terminal label T (denoted by $\mathrm{T}^{\mathcal{F}}$) not equivalent to $\mathrm{S}^{\mathcal{F}}$ such that the label T has the single rule $\mathrm{T}\vec{x} \leftarrow \vec{x} = \vec{x}$; and (b) for every pair of $X \in \mathcal{X}^{\mathcal{F}} \setminus \{\mathrm{T}\}$ and $Y \in \mathcal{X}^{\mathcal{F}}$, there is exactly one rule of the form $X\vec{x} \leftarrow \exists \vec{y}.\psi * Y\vec{y}$ (we denote this ψ by $\varphi^{\mathcal{F}}_{X,Y}\vec{x}\vec{y}$; note that ψ does not have non-terminal labels), where the elements of $\vec{x}\vec{y}$ are pairwise distinct.

▶ **Lemma 38.** *Every linear* FRS[PP] *has a GI-equivalent FA-linear* FRS[EP].

Proof. For the condition (a), we introduce a fresh non-terminal label T and introduce the rule $\mathrm{T}\vec{x} \leftarrow \vec{x} = \vec{x}$. For the condition (b), for each rule $X\vec{x} \leftarrow \varphi$, if φ does not have non-terminal labels, then we replace the rule with $X\vec{x} \leftarrow \exists\vec{z}.(\vec{z} = \vec{x} * \varphi) * \mathrm{T}\vec{z}$, where $\vec{z}$ is a sequence of fresh variables. Otherwise, let Y be the non-terminal label and transform the PP formula φ into a GI-equivalent formula of the form $\exists\vec{z}.\varphi' * Y\vec{u}$ by taking its prenex normal form and reordering the inner formulas appropriately. Then, transform it into the following formula: $\exists\vec{y}.(\exists\vec{z}.\vec{y} = \vec{u} * \varphi') * Y\vec{y}$, where $\vec{y}$ is a sequence of fresh variables. Next, for each pair $\langle X, Y\rangle$, let $\langle X\vec{x}_i \leftarrow \exists\vec{y}_i.\psi_i * Y\vec{y}_i\rangle_{i=1}^{n}$ be a permutation of all the rules of the form $X\vec{x} \leftarrow \exists\vec{y}.\psi * Y\vec{y}$. Without loss of generality, we can assume that $\vec{x}_1\vec{y}_1 = \cdots = \vec{x}_n\vec{y}_n$ (so we denote it by $\vec{x}\vec{y}$) by renaming variables. Then, replace these rules with the single rule $X\vec{x} \leftarrow \exists\vec{y}.(\bigvee_{i=1}^{n}\psi_i) * Y\vec{y}$. ◀

Finally, we present a translation from FA-linear FRS[EP]s into EP(TC) formulas.

▶ **Lemma 39.** *Every FA-linear* FRS[EP(TC)] $\mathcal{F}$ *has a GI-equivalent* EP(TC) *formula.*

Proof. By induction on $\#(\mathcal{X}^{\mathcal{F}})$. If $\mathcal{X}^{\mathcal{F}} = \{\mathtt{S}^{\mathcal{F}}, \mathtt{T}^{\mathcal{F}}\}$, then $\mathcal{F}$ is denoted by $\langle\{\mathtt{S}^{\mathcal{F}}, \mathtt{T}^{\mathcal{F}}\}, \{\mathtt{S}^{\mathcal{F}}\vec{z} \leftarrow \exists\vec{x}.\varphi * \mathtt{T}^{\mathcal{F}}\vec{x}, \mathtt{T}^{\mathcal{F}}\vec{x} \leftarrow \vec{x} = \vec{x}\}, \mathfrak{s}^{\mathcal{F}}\rangle$. Thus, $\mathcal{F}$ is GI-equivalent to the EP(TC) formula $\exists\vec{x}.\varphi * \vec{x} = \vec{x}$ ($\cong^{\mathrm{GI}} \exists\vec{x}.\varphi$). Otherwise, there exists $Y_0 \in \mathcal{X}^{\mathcal{F}} \setminus \{\mathtt{S}^{\mathcal{F}}, \mathtt{T}^{\mathcal{F}}\}$. We define $\mathcal{F}' \triangleq \langle \mathcal{X}^{\mathcal{F}} \setminus \{Y_0\}, \{X\vec{x} \leftarrow \exists\vec{z}.(\varphi^{\mathcal{F}}_{X,Z}\vec{x}\vec{z} \vee \exists\vec{y}.\varphi^{\mathcal{F}}_{X,Y_0}\vec{x}\vec{y} * \exists\vec{y}'.[\varphi^{\mathcal{F}}_{Y_0,Y_0}\vec{y}\vec{y}']^{*}_{\vec{y}\vec{y}'}\vec{y}\vec{y}' * \varphi^{\mathcal{F}}_{Y_0,Z}\vec{y}'\vec{z}) * Z\vec{z} \mid X, Z \in \mathcal{X}^{\mathcal{F}} \setminus \{Y_0\}, X \neq \mathtt{T}^{\mathcal{F}}\} \cup \{\mathrm{T}^{\mathcal{F}}\vec{x} \leftarrow \vec{x} = \vec{x}\}, \mathfrak{s}^{\mathcal{F}}\rangle$, where elements of $\vec{x}\vec{z}\vec{y}\vec{y}'$ are pairwise distinct. Here, $[\varphi]^{*}_{\vec{x}\vec{y}}\vec{u}\vec{w}$ abbreviates the formula $\vec{u} = \vec{w} \vee [\varphi]^{+}_{\vec{x}\vec{y}}\vec{u}\vec{w}$. Then, the FA-linear FRS[EP(TC)] $\mathcal{F}'$ is GI-equivalent to $\mathcal{F}$ because there are transformations between derivation trees of $\mathcal{F}$ and those of $\mathcal{F}'$ in the same manner as the proof of Lemma 36. By I.H., $\mathcal{F}'$ has some GI-equivalent EP(TC) formula φ. Thus by using this φ, it has been proved. ◀

Proof of Theorem 1(3)⇐. By Lemma 38 and 39 (with Proposition 29 and 30). ◀

6 Conclusion

We have presented a perspective on graph languages via logical formulas by introducing *GI-semantics*. We have presented an axiomatization of the equational theory of PP/EP formulas under GI-semantics, and we have shown that several classes of existential positive logic formulas under GI-semantics have the same expressive power as those of HRGs. One future work is to find some axiomatization or some proof system of the (in)equational theory of EP(TC), or EP(LFP). Another possible future work is to study some classes of (bounded treewidth) graph languages by considering syntactic fragments, e.g., for finding decidable (or tractable) fragments of graph language problems. It would also be interesting to extend this logic to higher-order fixpoint logic (for a graph extension of higher-order grammars [17, 22]).

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi:10.1017/cbo9781139172752.

3 Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(1):83–127, 1987. doi:10.1007/BF01692060.

4 Filippo Bonchi, Jens Seeber, and Pawel Sobocinski. Graphical Conjunctive Queries. In *27th EACSL Annual Conference on Computer Science Logic (CSL '18)*, volume 119, pages 13:1–13:23. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CSL.2018.13.

5 Francis Bossut, Max Dauchet, and Bruno Warin. A Kleene Theorem for a Class of Planar Acyclic Graphs. *Information and Computation*, 117(2):251–265, 1995. doi:10.1006/inco.1995.1043.

6 Paul Brunet and Damien Pous. Petri automata. *Logical Methods in Computer Science*, 13(3), 2017. doi:10.23638/LMCS-13(3:33)2017.

7 Janusz A. Brzozowski and Edward J. McCluskey. Signal Flow Graph Techniques for Sequential Circuit State Diagrams. *IEEE Transactions on Electronic Computers*, EC-12(2):67–76, 1963. doi:10.1109/PGEC.1963.263416.

8 J. Richard Büchi. Weak Second-Order Arithmetic and Finite Automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6(1-6):66–92, 1960. doi:10.1002/malq.19600060105.

9 J. Richard Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science 1960*, pages 1–11. Stanford University Press, 1962.

10 Luca Cardelli, Philippa Gardner, and Giorgio Ghelli. A Spatial Logic for Querying Graphs. In *International Colloquium on Automata, Languages, and Programming (ICALP '02)*, pages 597–610. Springer Verlag, 2002. doi:10.1007/3-540-45465-9_51.

11 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *9th annual ACM symposium on Theory of computing (STOC '77)*, pages 77–90. ACM Press, 1977. doi:10.1145/800105.803397.

12 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000. doi:10.1016/S0304-3975(99)00220-0.

13 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications, 2007. URL: http://www.grappa.univ-lille3.fr/tata.

14 Enric Cosme Llópez and Damien Pous. K4-free Graphs as a Free Algebra. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS '17)*, pages 76:1–76::14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICS.MFCS.2017.76.

15 Bruno Courcelle and Joost Engelfriet. A Logical Characterization of the Sets of Hypergraphs Defined by Hyperedge Replacement Grammars. *Mathematical Systems Theory*, 28(6):515–552, 1995.

16 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic*. Cambridge University Press, 2012. doi:10.1017/CBO9780511977619.

17 Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20(2):95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

18 Christian Doczkal and Damien Pous. Treewidth-Two Graphs as a Free Algebra. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS '18)*, volume 117, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.60.

19 Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge Replacement Graph Grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, chapter 2, pages 95–162. World Scientific, 1997. doi:10.1142/9789812384720_0002.

20 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer Monographs in Mathematics. Springer, 2 edition, 1995. doi:10.1007/3-540-28788-4.

21 Calvin C. Elgot. Decision Problems of Finite Automata Design and Related Arithmetics. *Transactions of the American Mathematical Society*, 98(1):21, 1961. doi:10.2307/1993511.

22 Joost Engelfriet. Iterated stack automata and complexity classes. *Information and Computation*, 95(1):21–75, 1991. doi:10.1016/0890-5401(91)90015-T.

23 Joost Engelfriet and Vincent van Oostrom. Logical Description of Context-free Graph Languages. *Journal of Computer and System Sciences*, 55(3):489–503, 1997. doi:10.1006/jcss.1997.1510.

24 David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992. doi:10.1016/0890-5401(92)90041-D.

25 Jerome Feder. Plex languages. *Information Sciences*, 3(3):225–241, 1971. doi:10.1016/S0020-0255(71)80008-7.

26 Tomás Feder and Moshe Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *25th annual ACM symposium on Theory of computing (STOC '93)*, volume Part F1295, pages 612–622. ACM Press, 1993. doi:10.1145/167088.167245.

27 Stephen C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies. (AM-34)*, pages 3–42. Princeton University Press, 1956.

28 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000. doi:10.1006/jcss.2000.1713.

29 Haas Leiß. Towards Kleene Algebra with recursion. In *5th International Workshop on Computer Science Logic (CSL '91)*, pages 242–256. Springer, 1991. doi:10.1007/BFb0023771.

30 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1-2):347–380, 2000. doi:10.1016/S0304-3975(00)00031-1.

31 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966. doi:10.1016/S0019-9958(66)80013-X.

32 Yoshiki Nakamura. Expressive Power and Succinctness of the Positive Calculus of Relations. In *18th International Conference on Relational and Algebraic Methods in Computer Science (RAMiCS '20)*, volume 12062 of *LNCS*, pages 204–220. Springer, Cham, 2020. doi:10.1007/978-3-030-43520-2_13.

33 Yoshiki Nakamura. A full version of this paper, 2021. URL: https://yoshikinakamura.bitbucket.io/papers/ep_for_graph.

34 Mizuhito Ogawa, Zhenjiang Hu, and Isao Sasano. Iterative-free program analysis. In *8th ACM SIGPLAN international conference on Functional programming (ICFP '03)*, volume 8, pages 111–123. ACM Press, 2003. doi:10.1145/944705.944716.

35 Peter W. O'Hearn and David J. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. doi:10.2307/421090.

36 Theodosios Pavlidis. Linear and Context-Free Graph Grammars. *Journal of the ACM*, 19(1):11–22, 1972. doi:10.1145/321679.321682.

37 Andrew M. Pitts. *Nominal Sets*. Cambridge University Press, 2013. doi:10.1017/CBO9781139084673.

38 David Pym. Resource semantics: logic as a modelling technology. *ACM SIGLOG News*, 6(2):5–41, 2019. doi:10.1145/3326938.3326940.

39 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

40 Benjamin Rossman. Homomorphism preservation theorems. *Journal of the ACM*, 55(3):1–53, 2008. doi:10.1145/1379759.1379763.

41 Alfred Tarski. On the Calculus of Relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941. doi:10.2307/2268577.

42 Ken Thompson. Programming Techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.

43 Boris A. Trakhtenbrot. Finite automata and the logic of monadic predicates (in Russian). *Doklady Akademii Nauk SSSR*, 140:326–329, 1961.

Structural Properties of the First-Order Transduction Quasiorder

Jaroslav Nešetřil
Computer Science Institute, Charles University (IUUK), Prague, Czech Republic

Patrice Ossona de Mendez
Centre d'Analyse et de Mathématiques Sociales (CNRS, UMR 8557), Paris, France
Computer Science Institute, Charles University, Prague, Czech Republic

Sebastian Siebertz
University of Bremen, Germany

Abstract

Logical transductions provide a very useful tool to encode classes of structures inside other classes of structures. In this paper we study first-order (FO) transductions and the quasiorder they induce on infinite classes of finite graphs. Surprisingly, this quasiorder is very complex, though shaped by the locality properties of first-order logic. This contrasts with the conjectured simplicity of the monadic second order (MSO) transduction quasiorder. We first establish a local normal form for FO transductions, which is of independent interest. Then we prove that the quotient partial order is a bounded distributive join-semilattice, and that the subposet of *additive* classes is also a bounded distributive join-semilattice. The FO transduction quasiorder has a great expressive power, and many well studied class properties can be defined using it. We apply these structural properties to prove, among other results, that FO transductions of the class of paths are exactly perturbations of classes with bounded bandwidth, that the local variants of monadic stability and monadic dependence are equivalent to their (standard) non-local versions, and that the classes with pathwidth at most k, for $k \geq 1$ form a strict hierarchy in the FO transduction quasiorder.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Mathematics of computing → Graph theory

Keywords and phrases Finite model theory, first-order transductions, structural graph theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.31

Related Version *Full Version*: https://arxiv.org/abs/2010.02607

Funding This paper is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 810115 – DYNASNET) and from the German Research Foundation (DFG) with grant agreement No 444419611.

1 Introduction and statement of results

Transductions provide a model theoretical tool to encode relational structures (or classes of relational structures) inside other (classes of) relational structures. Transductions naturally induce a quasiorder, that is, a reflexive and transitive binary relation, on classes of relational structures. We study here the first-order (FO) and monadic second-order (MSO) transduction quasiorders $\sqsubseteq_{\mathrm{FO}}$ and $\sqsubseteq_{\mathrm{MSO}}$ on infinite classes of finite graphs. These quasiorders are very different and both have a sound combinatorial and model theoretic relevance, as we will outline below. To foster the further discussion, let us (slightly informally) introduce the concept of transductions. Formal definitions will be given in Section 2.

A *transduction* T (on graphs) is the composition of a copying operation, a coloring operation, and a simple interpretation. The *copying operation* C_k maps a graph G to the graph $\mathsf{C}_k(G)$ obtained by taking k disjoint copies of G and making all the copies of a single vertex adjacent; the *coloring operation* maps a graph G to the set $\Gamma(G)$ of all possible colorings of G; a *simple interpretation* I maps a colored graph G^+ to a graph H, whose vertex set (resp. edge set) is a definable subset of $V(G^+)$ (resp. of $V(G^+) \times V(G^+)$). In this way, the transduction T maps a graph G to a set $\mathsf{T}(G)$ of graphs defined as $\mathsf{T}(G) \coloneqq \mathsf{I} \circ \Gamma \circ \mathsf{C}_k(G) = \{\mathsf{I}(H^+) : H^+ \in \Gamma(\mathsf{C}_k(G))\}$. This naturally extends to a graph class[1] $\mathscr{C}$ by $\mathsf{T}(\mathscr{C}) \coloneqq \bigcup_{G \in \mathscr{C}} \mathsf{T}(G)$.

We say that a class $\mathscr{C}$ *is a transduction of* a class $\mathscr{D}$ if there exists a transduction T with $\mathscr{C} \subseteq \mathsf{T}(\mathscr{D})$, and we denote this by $\mathscr{C} \sqsubseteq \mathscr{D}$. We write $\mathscr{C} \equiv \mathscr{D}$ for $\mathscr{C} \sqsubseteq \mathscr{D}$ and $\mathscr{D} \sqsubseteq \mathscr{C}$, $\mathscr{C} \sqsubset \mathscr{D}$ for $\mathscr{C} \sqsubseteq \mathscr{D}$ and $\mathscr{C} \not\equiv \mathscr{D}$, and $\mathscr{C} \lhd \mathscr{D}$ for the property that $(\mathscr{C}, \mathscr{D})$ is a *cover*, that is, that $\mathscr{C} \sqsubset \mathscr{D}$ and there is no class $\mathscr{F}$ with $\mathscr{C} \sqsubset \mathscr{F} \sqsubset \mathscr{D}$. For a logic $\mathcal{L}$ we write $\mathscr{C} \sqsubseteq_{\mathcal{L}} \mathscr{D}$ to stress that the simple interpretation of the transduction uses $\mathcal{L}$-formulas.

For most commonly studied logics $\mathcal{L}$ transductions compose and in this case $\sqsubseteq_{\mathcal{L}}$ is a quasiorder. We study here mainly the first-order (FO) and monadic second-order (MSO) transduction quasiorders $\sqsubseteq_{\mathrm{FO}}$ and $\sqsubseteq_{\mathrm{MSO}}$. As with the colorings all vertex subsets become definable, it follows that we can restrict our attention to infinite *hereditary* classes, that is, infinite classes that are closed under taking induced subgraphs.

MSO transductions are basically understood. Let us write $\mathcal{E}$ for the class of edgeless graphs, $\mathcal{T}_n$ for the class of forests of depth n (where the depth of a (rooted) tree is the maximum number of vertices on a root-leaf path, hence $\mathcal{T}_1 = \mathcal{E}$), $\mathcal{P}$ for the class of all paths, $\mathcal{T}$ for the class of all trees and $\mathcal{G}$ for the class of all graphs. The MSO transduction quasiorder is conjectured to be simply the chain $\mathcal{E} \lhd_{\mathrm{MSO}} \mathcal{T}_2 \lhd_{\mathrm{MSO}} \dots \lhd_{\mathrm{MSO}} \mathcal{T}_n \lhd_{\mathrm{MSO}} \dots \sqsubseteq_{\mathrm{MSO}} \mathcal{P} \lhd_{\mathrm{MSO}} \mathcal{T} \lhd_{\mathrm{MSO}} \mathcal{G}$ [2].

In a combinatorial setting this hierarchy has a very concrete meaning and it was investigated using the following notions: a class $\mathscr{C}$ has *bounded shrubdepth* if $\mathscr{C} \sqsubseteq_{\mathrm{MSO}} \mathcal{T}_n$ for some n; $\mathscr{C}$ has *bounded linear cliquewidth* if $\mathscr{C} \sqsubseteq_{\mathrm{MSO}} \mathcal{P}$; $\mathscr{C}$ has *bounded cliquewidth* if $\mathscr{C} \sqsubseteq_{\mathrm{MSO}} \mathcal{T}$. These definitions very nicely illustrate the treelike structure of graphs from the above mentioned classes from a logical point of view, which is combinatorially captured by the existence of treelike decompositions with certain properties. It is still open whether the MSO transduction quasiorder is as shown above [2, Open Problem 9.3], though the initial fragment $\mathcal{E} \lhd_{\mathrm{MSO}} \mathcal{T}_2 \lhd_{\mathrm{MSO}} \mathcal{T}_3 \lhd_{\mathrm{MSO}} \dots \lhd_{\mathrm{MSO}} \mathcal{T}_n$ has been proved to be as stated in [8]. Thus we are essentially left with the following three questions: Does $\mathscr{C} \sqsubset_{\mathrm{MSO}} \mathcal{P}$ imply $(\exists n)\ \mathscr{C} \sqsubseteq_{\mathrm{MSO}} \mathcal{T}_n$? This is equivalent to the question whether one can transduce with MSO arbitrary long paths from any class of unbounded shrubdepth (see [11] for a proof of the CMSO version). Is the pair $(\mathcal{P}, \mathcal{T})$ a cover? Is the pair $(\mathcal{T}, \mathcal{G})$ a cover? This last question is related to a famous conjecture of Seese [18] and the CMSO version has been proved in [4].

As in the MSO case, the FO transduction quasiorder allows to draw important algorithmic and structural dividing lines. For instance MSO collapses to FO on classes of bounded shrubdepth [8]. Classes of bounded shrubdepth are also characterized as being FO transductions of classes of trees of bounded depth [9]. FO transductions give alternative characterizations of other graph class properties mentioned above: a class $\mathscr{C}$ has bounded linear cliquewidth if and only if $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{H}$, where $\mathcal{H}$ denotes the class of *half-graphs* (bipartite graphs with vertex set $\{a_1, \dots, a_n\} \cup \{b_1, \dots, b_n\}$ and edge set $\{a_i b_j \,:\, 1 \le i \le j \le n\}$ for some n) [3], and bounded cliquewidth if and only if $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{TP}$, where $\mathcal{TP}$ denotes the class of *trivially perfect graphs* (comparability graphs of rooted forests) [3]. Also, it follows from [1] that FO transductions

[1] By a class we always mean a set of finite graphs, where we identify isomorphic graphs.

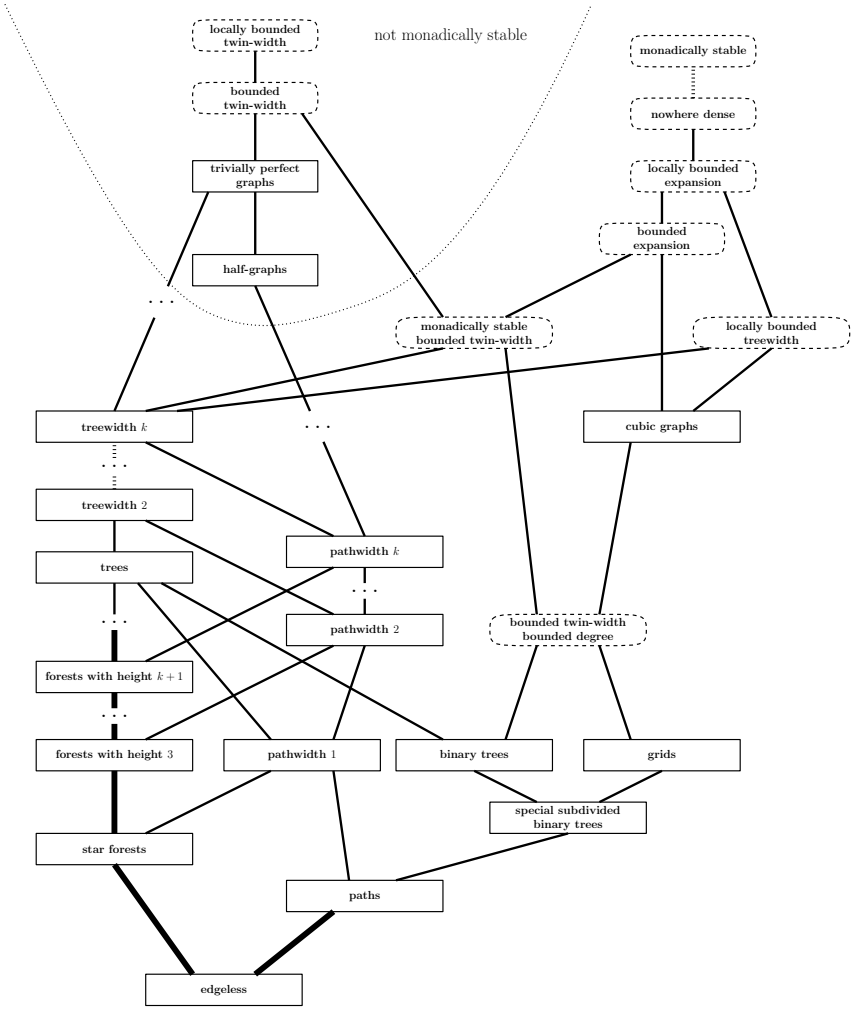


Figure 1 Partial outline of the FO transduction quasiorder. The special subdivided binary trees are those subdivisions of binary trees that are subgraphs of the grid. Dashed boxes correspond to families of not necessarily transduction equivalent graph classes sharing a common property. Fat lines correspond to covers, normal lines correspond to strict containment $\sqsubset$, dotted lines correspond to containment (with a possible collapse). Some parts of Figure 1 will be refined in Figure 2.

allow to give an alternative characterizations of classical model theoretical properties: A class $\mathscr{C}$ is *monadically stable* if $\mathscr{C} \not\sqsupseteq_{\mathrm{FO}} \mathcal{H}$ and *monadically dependent* if $\mathscr{C} \not\sqsupseteq_{\mathrm{FO}} \mathcal{G}$. We further call a class $\mathscr{C}$ *monadically straight* if $\mathscr{C} \not\sqsupseteq_{\mathrm{FO}} \mathcal{TP}$. To the best of our knowledge this property has not been studied in the literature but seems to play a key role in the study of FO transductions.

The FO transduction quasiorder has not been studied in detail previously and it turns out that it is much more complicated than the MSO transduction quasiorder. This is outlined in Figure 1, and it is the goal of this paper to explore this quasiorder.

We are motivated by three aspects of the $\sqsubseteq_{\mathrm{FO}}$ quasiorder that have been specifically considered in the past and appeared to be highly non-trivial. The first aspect is the conjectured property that every class that cannot FO transduce paths has bounded shrubdepth (hence is an FO transduction of a class of bounded height trees). The second aspect that was studied in detail concerns the chain formed by classes with bounded pathwidth, which is

eventually covered by the class of half-graphs. This is related to the fact that in the FO transduction quasiorder there is no class between the classes with bounded pathwidth and the class $\mathcal{H}$ of half-graphs [15, 16]. The third aspect concerns the chain of classes with bounded treewidth, which is eventually covered by the class of trivially perfect graphs. This is related to the fact that if $\mathcal{H} \not\sqsubseteq_{\mathrm{FO}} \mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{TP}$ (that is, $\mathscr{C}$ is a monadically stable class with bounded cliquewidth), then $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{TW}_n$ for some n, where $\mathcal{TW}_n$ denotes the class of graphs with treewidth at most n [14].

In this paper, we establish the three kinds of results and show that despite its complexity the FO transduction quasiorder is strongly structured.

A local normal form for transductions

In Section 3.1 we introduce a normal form for FO transductions that captures the local character of first-order logic, by proving that every FO transduction can be written as the composition of a copying operation, a transduction that connects only vertices at a bounded distance, and a perturbation, which is a sequence of subset complementations (Theorem 2). In Section 4 we give two applications of this normal form. We first characterize the equivalence class of the class of paths in the FO transduction quasiorder (Theorem 8). Then, we prove that the local versions of monadic stability, monadic straightness, and monadic dependence are equivalent to the non-local versions (Theorem 9). This result is of independent interest and may be relevant e.g. for locality based FO model-checking on these classes.

Structural properties of the transduction quasiorder

In Section 5 we prove that the partial orders obtained as the quotient of the transduction quasiorder and the non-copying transduction quasiorder are bounded distributive join-semilattices (Theorem 14) and discuss some of their properties. In particular we prove that every class closed under disjoint union is join-irreducible. Recall that a partial order $(X, \leq)$ is a *join semi-lattice* if for all $x, y \in X$ there exists a least upper bound $x \vee y$ of $\{x, y\}$, called the join of x and y. It is *distributive* if, for all $a, b, x \in X$ with $x \leq a \vee b$ there exist $x_1 \leq a$, $x_2 \leq b$, with $x = x_1 \vee x_2$. An element $x \in X$ is *join-irreducible* if x is not the join of two incomparable elements. Then we consider the subposets induced by *additive* classes, which are the classes equivalent to the class of disjoint unions of pairs of graphs in the class. We prove that these subposets are also bounded distributive join-semilattices (Theorem 23), but with a different join. We discuss some properties of these subposets and in particular prove that every class closed under disjoint union and equivalent to its subclass of connected graphs is join-irreducible.

The transduction quasiorder on some classes

In Section 6 we focus on the transduction quasiorders on the class of paths, the class of trees, classes of bounded height trees, classes with bounded pathwidth, classes with bounded treewidth, and derivatives. In particular we prove that classes with bounded pathwidth form a strict hierarchy (Theorem 30). This result was the main motivation for this study, and we conjecture that a similar statement holds with treewidth. This would be a consequence of the conjecture that the class of all graphs with treewidth at most n is incomparable with the class of all graphs with pathwidth at most $n + 1$, for every positive integer n.

2 Preliminaries and basic properties of transductions

We assume familiarity with first-order logic and graph theory and refer e.g. to [5, 10] for background and for all undefined notation. The vertex set of a graph G is denoted as $V(G)$ and its edge set $E(G)$. All graphs considered in this paper are finite. The *complement* of a graph G is the graph $\overline{G}$ with the same vertex set, in which two vertices are adjacent if they are not adjacent in G. The disjoint union of two graphs G and H is denoted as $G \cup H$, and their *complete join* $\overline{\overline{G} \cup \overline{H}}$ as $G + H$. We denote by K_t the complete graph on t vertices. Hence, $G + K_1$ is obtained from G by adding a new vertex, called an *apex*, that is connected to all vertices of G. For a class $\mathscr{C}$ of graphs we denote by $\mathscr{C} + K_1$ the class obtained from $\mathscr{C}$ by adding an apex to each graph of $\mathscr{C}$. The *lexicographic product* $G \bullet H$ of two graphs G and H is the graph with vertex set $V(G) \times V(H)$, in which (u, v) is adjacent to (u', v') if either u is adjacent to u' in G or $u = u'$ and v is adjacent to v' in H. The *pathwidth* $\mathrm{pw}(G)$ of a graph G is equal to one less than the smallest clique number of an interval graph that contains G as a subgraph, that is, $\mathrm{pw}(G) = \min\{\omega(H) - 1 :$ for an interval graph H with $H \supseteq G\}$. The *treewidth* $\mathrm{tw}(G)$ of a graph G is equal to one less than the smallest clique number of a chordal graph that contains G as a subgraph, that is, $\mathrm{tw}(G) = \min\{\omega(H) - 1 :$ for a chordal graph H with $H \supseteq G\}$. We write G^k for the *k-th power* of G (which has the same vertex set as G and two vertices are connected if their distance is at most k in G). The *bandwidth* of a graph G is $\mathrm{bw}(G) = \min\{\ell :$ for $P \in \mathcal{P}$ with $P^\ell \supseteq G, \}$.

In this paper we consider either graphs or Σ-*expanded graphs*, that is, graphs with additional unary relations in Σ (for a set Σ of unary relation symbols). We usually denote graphs by $G, H, \ldots$ and Σ-expanded graphs by $G^+, H^+, G^*, H^*, \ldots$, but sometimes we will use $G, H, \ldots$ for Σ-expanded graphs as well. We shall often use the term "colored graph" instead of Σ-expanded graph. In formulas, the adjacency relation will be denoted as $E(x, y)$. For each non-negative integer r we can write a formula $\delta_{\leq r}(x, y)$ such that for every graph G and all $u, v \in V(G)$ we have $G \models \delta_{\leq r}(u, v)$ if and only if the distance between u and v in G is at most r. For improved readability we write $\mathrm{dist}(x, y) \leq r$ for $\delta_{\leq r}(x, y)$. The *open neighborhood* $N^G(v)$ of a vertex v is the set of neighbors of v. For $U \subseteq V(G)$ we write $B_r^G(U)$ for the subgraph of G induced by the vertices at distance at most r from some vertex of U. For the sake of simplicity we use for balls of radius r the notation $B_r^G(v)$ instead of $B_r^G(\{v\})$ and, if G is clear from the context, we drop the superscript G. For a class $\mathscr{C}$ and an integer r, we denote by $\mathcal{B}_r^{\mathscr{C}}$ the class of all the balls of radius r of graphs in $\mathscr{C}$: $\mathcal{B}_r^{\mathscr{C}} = \{B_r^G(v) \mid G \in \mathscr{C}$ and $v \in V(G)\}$. For a formula $\varphi(x_1, \ldots, x_k)$ and a graph (or a Σ-expanded graph) G we define

$$\varphi(G) \coloneqq \{(v_1, \ldots, v_k) \in V(G)^k \ : \ G \models \varphi(v_1, \ldots, v_k)\}.$$

For a positive integer k, the *k-copy operation* C_k maps a graph G to the graph $\mathsf{C}_k(G)$ consisting of k copies of G where the copies of each vertex are made adjacent (that is, the copies of each vertex induce a clique and there are no other edges between the copies of G). Note that for $k = 1$, C_k maps each graph G to itself. (Thus C_1 is the identity mapping.)

For a set Σ of unary relations, the *coloring operation* Γ_Σ maps a graph G to the set $\Gamma_\Sigma(G)$ of all its Σ-expansions.

A *simple interpretation* I of graphs in Σ-expanded graphs is a pair $(\nu(x), \eta(x, y))$ consisting of two formulas (in the first-order language of Σ-expanded graphs), where η is symmetric and anti-reflexive (i.e. $\models \eta(x, y) \leftrightarrow \eta(y, x)$ and $\models \eta(x, y) \rightarrow \neg(x = y)$). If G^+ is a Σ-expanded graph, then $H = \mathsf{I}(G^+)$ is the graph with vertex set $V(H) = \nu(G^+)$ and edge set $E(H) = \eta(G) \cap \nu(G)^2$.

A *transduction* T is the composition $\mathsf{I} \circ \Gamma_\Sigma \circ \mathsf{C}_k$ of a copy operation C_k, a coloring operation Γ_Σ, and a simple interpretation I of graphs in Σ-expanded graphs. In other words, for every graph G we have $\mathsf{T}(G) = \{\mathsf{I}(H^+) : H \in \Gamma_\Sigma(\mathsf{C}_k(G))\}$. A transduction T is *non-copying* if it is the composition of a coloring operation and a simple interpretation, that is if it can written as $\mathsf{I} \circ \Gamma_\Sigma \circ \mathsf{C}_1$ $(= \mathsf{I} \circ \Gamma_\Sigma)$. We say that a transduction T' *subsumes* a transduction T if for every graph G we have $\mathsf{T}'(G) \supseteq \mathsf{T}(G)$. We denote by $\mathsf{T}' \geq \mathsf{T}$ the property that T' subsumes T.

For a class $\mathscr{D}$ and a transduction T we define $\mathsf{T}(\mathscr{D}) = \bigcup_{G \in \mathscr{D}} \mathsf{T}(G)$ and we say that a class $\mathscr{C}$ is a T-*transduction* of $\mathscr{D}$ if $\mathscr{C} \subseteq \mathsf{T}(\mathscr{D})$. We also say that T *encodes* $\mathscr{C}$ in $\mathscr{D}$. A class $\mathscr{C}$ of graphs is a *(non-copying) transduction* of a class $\mathscr{D}$ of graphs if it is a T-transduction of $\mathscr{D}$ for some (non-copying) transduction T. We denote by $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$ (resp. $\mathscr{C} \sqsubseteq^\circ_{\mathrm{FO}} \mathscr{D}$) the property that the class $\mathscr{C}$ is an FO transduction (resp. a non-copying FO transduction) of the class $\mathscr{D}$. It is easily checked that the composition of two (non-copying) transductions is a (non-copying) transduction (see, for instance [7]). Thus the relations $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$ and $\mathscr{C} \sqsubseteq^\circ_{\mathrm{FO}} \mathscr{D}$ are quasiorders on classes of graphs Intuitively, if $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$, then $\mathscr{C}$ is at most as complex as $\mathscr{D}$. Equivalences for $\sqsubseteq_{\mathrm{FO}}$ and $\sqsubseteq^0_{\mathrm{FO}}$ are defined naturally.

We say that a class $\mathscr{C}$ *does not need copying* if for every integer k the class $\mathsf{C}_k(\mathscr{C})$ is a non-copying transduction of $\mathscr{C}$. For example, as a matching cannot be transduced from an edgeless graph without copying, the class of edgeless graphs needs copying. To the opposite, the reader can easily check that the class of paths does not need copying.

We take time for some observations.

▶ **Observation 1.** *If $\mathscr{C}$ does not need copying and $\mathscr{C} \equiv_{\mathrm{FO}} \mathscr{D}$, then $\mathscr{D}$ does not need copying.*

Proof. This follows from the fact that every class $\mathscr{C}$ is a non-copying transduction of $\mathsf{C}_k(\mathscr{C})$. ◀

▶ **Observation 2.** *A class $\mathscr{C}$ does not need copying if and only if $\mathsf{C}_2(\mathscr{C}) \equiv^\circ_{\mathrm{FO}} \mathscr{C}$.*

Proof. It is easily checked that for every positive integer k there is a non-copying transduction T_k such that $\mathsf{T}_k \circ \mathsf{C}_k \circ \mathsf{C}_2$ subsumes C_{2k}. Assume $\mathsf{C}_2(\mathscr{C}) \equiv^\circ_{\mathrm{FO}} \mathscr{C}$. Then if $\mathsf{C}_k(\mathscr{C}) \sqsubseteq^\circ_{\mathrm{FO}} \mathscr{C}$ we deduce from $\mathsf{C}_2(\mathscr{C}) \sqsubseteq^\circ_{\mathrm{FO}} \mathscr{C}$ that $\mathsf{C}_{2k}(\mathscr{C}) \sqsubseteq^\circ_{\mathrm{FO}} \mathscr{C}$. By induction we get $\mathsf{C}_k(\mathscr{C}) \equiv^\circ_{\mathrm{FO}} \mathscr{C}$ for every positive integer k. ◀

▶ **Observation 3.** *A class $\mathscr{D}$ does not need copying if and only if for every class $\mathscr{C}$ we have $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$ if and only if $\mathscr{C} \sqsubseteq^\circ_{\mathrm{FO}} \mathscr{D}$.*

▶ **Observation 4.** *If a class $\mathscr{C}$ is closed under adding pendant vertices (that is, if $G \in \mathscr{C}$ and $v \in V(G)$, then G', which is obtained from G by adding a new vertex adjacent only to v, is also in $\mathscr{C}$) then $\mathscr{C}$ does not need copying.*

A *subset complementation* transduction is defined by the quantifier-free interpretation on a Σ-expansion (with $\Sigma = \{M\}$) by $\eta(x, y) := (x \neq y) \wedge \neg\big(E(x, y) \leftrightarrow (M(x) \wedge M(y))\big)$. In other words, the subset complementation transduction complements the adjacency inside the subset of the vertex set defined by M. We denote by $\oplus M$ the subset complementation defined by the unary relation M. A *perturbation* is a composition of (a bounded number of) subset complementations. Let r be a non-negative integer. A formula $\varphi(x_1, \ldots, x_k)$ is *r-local* if for every (Σ-expanded) graph G and all $v_1, \ldots, v_k \in V(G)$ we have $G \models \varphi(v_1, \ldots, v_k) \iff B^G_r(\{v_1, \ldots, v_k\}) \models \varphi(v_1, \ldots, v_k)$. An r-local formula $\varphi(x_1, \ldots, x_k)$ is *strongly r-local* if $\models \varphi(x_1, \ldots, x_k) \to \mathrm{dist}(x_i, x_j) \leq r$ for all $1 \leq i < j \leq k$ (see [13]).

▶ **Lemma 1** (Gaifman's Locality Theorem [6]). *Every formula $\varphi(x_1,\dots,x_m)$ is equivalent to a Boolean combination of t-local formulas and so-called* basic local sentences *of the form*

$$\exists x_1 \dots \exists x_k \Big(\bigwedge_{1\le i\le k} \chi(x_i) \wedge \bigwedge_{1\le i<j\le k} \operatorname{dist}(x_i,x_j) > 2r \Big) \quad \textit{(where } \chi \textit{ is } r\textit{-local).}$$

Furthermore, if the quantifier-rank of φ is q, then $r \le 7^{q-1}$, $t \le 7^{q-1}/2$, and $k \le q+m$.

We call a transduction T *immersive* if it is non-copying and the formulas in the interpretation associated to T are strongly local.

3 Local properties of FO transductions

3.1 A local normal form

We now establish a normal form for first-order transductions that captures the local character of first-order logic and further study the properties of immersive transductions. The normal form is based on Gaifman's Locality Theorem and uses only *strongly local formulas*, while the basic-local sentences are handled by subset complementations. This normal form will be one of the main tools to establish results in the paper.

▶ **Theorem 2.** *Every non-copying transduction T is subsumed by the composition of an immersive transduction $\mathsf{T}_{\mathrm{imm}}$ and a perturbation P, that is $\mathsf{T} \le \mathsf{P} \circ \mathsf{T}_{\mathrm{imm}}$.*

Consequently, every transduction T is subsumed by the composition of a copying operation C, an immersive transduction $\mathsf{T}_{\mathrm{imm}}$ and a perturbation P, that is $\mathsf{T} \le \mathsf{P} \circ \mathsf{T}_{\mathrm{imm}} \circ \mathsf{C}$.

Proof. Let $\mathsf{T} = \mathsf{I}_{\mathsf{T}} \circ \Gamma_{\Sigma_{\mathsf{T}}}$ be a non-copying transduction. Without loss of generality, we may assume that the interpretation I_{T} defines the domain directly from the Σ_{T}-expansion. Then the only non-trivial part of the interpretation is the adjacency relation, which is defined by a symmetric and anti-reflexive formula $\eta(x,y)$. We shall prove that the transduction T is subsumed by the composition of an immersive transduction T_ψ and a perturbation P.

We define Σ_{T_ψ} as the disjoint union of Σ_{T} and a set $\Sigma_\psi = \{T_i \mid 1 \le i \le n_1\}$ for some integer n_1 we shall specify later and let $\Sigma_{\mathsf{P}} = \{Z_j \mid 1 \le j \le n_2\}$ for some integer n_2 we shall also specify later. Let q be the quantifier rank of $\eta(x,y)$. According to Lemma 1, η is logically equivalent to a formula in Gaifman normal form, that is, to a Boolean combination of t-local formulas and basic-local sentences $\theta_1,\dots,\theta_{n_1}$. To each θ_i we associate a unary predicate $T_i \in \Sigma_\psi$. We consider the formula $\widetilde{\eta}(x,y)$ obtained from the Gaifman normal form of $\eta(x,y)$ by replacing the sentence θ_i by the atomic formula $T_i(x)$. Note that $\widetilde{\eta}$ is t-local.

Under the assumption that $\operatorname{dist}(x,y) > 2t$ every t-local formula $\chi(x,y)$ is equivalent to $\chi_1(x) \wedge \chi_2(y)$ for t-local formulas $\chi_1(x)$ and $\chi_2(y)$. Furthermore, t-local formulas are closed under boolean combinations. By bringing $\widetilde{\eta}$ into disjunctive normal form and grouping conjuncts appropriately, it follows that under the assumption $\operatorname{dist}(x,y) > 2t$ the formula $\widetilde{\eta}$ is equivalent to a formula $\widetilde{\varphi}(x,y)$ of the form $\bigvee_{(i,j)\in\mathcal{F}} \zeta_i(x) \wedge \zeta_j(y)$, where $\mathcal{F} \subseteq [n_2] \times [n_2]$ for some integer n_2 and the formulas ζ_i $(1 \le i \le n_2)$ are t-local. By considering appropriate boolean combinations (or, for those familiar with model theory, by assuming that the ζ_i define local types) we may assume that $\models \forall x \bigwedge_{i\neq j} \neg(\zeta_i(x) \wedge \zeta_j(x))$, that is, every element of a graph satisfies at most one of the ζ_i. Note also that $\mathcal{F}$ is symmetric as η (hence $\widetilde{\eta}$ and $\widetilde{\varphi}$) are symmetric.

We define $\psi(x,y) := \neg(\widetilde{\eta}(x,y) \leftrightarrow \widetilde{\varphi}(x,y)) \wedge (\operatorname{dist}(x,y) \le 2t)$, which is $2t$-strongly local, and we define $\mathsf{I}_{\mathsf{T}_\psi}$ as the interpretation of graphs in Σ_{T_ψ}-structures by using the same definitions as in I_{T} for the domain, then defining the adjacency relation by $\psi(x,y)$. To

each formula ζ_i we associate a unary predicate $Z_i \in \Sigma_{\mathsf{P}}$. We define the perturbation P as the sequence of subset complementations $\oplus Z_i$ (for $(i,i) \in \mathcal{F}$) and of $\oplus Z_i \oplus Z_j \oplus (Z_i \cup Z_j)$ (for $(i,j) \in \mathcal{F}$ and $i < j$). Denote by $\varphi(x,y)$ the formula defining the edges in the interpretation I_{P}. Note that when the Z_i are pairwise disjoint, then P complements exactly the edges of Z_i or between Z_i and Z_j, respectively. The operation $\oplus(Z_i \cup Z_j)$ complements all edges between Z_i and Z_j, but also inside Z_i and Z_j, which is undone by $\oplus Z_j$ and $\oplus Z_i$.

Now assume that a graph H is a T-transduction of a graph G, and let G^+ be a Σ_{T}-expansion of G such that $H = \mathsf{I}_{\mathsf{T}}(G^+)$. We define the Σ_ψ-expansion G^* of G^+ (which is thus a Σ_{T_ψ}-expansion of G) by defining, for each $i \in [n_1]$, $T_i(G^*) = V(G)$ if $G^+ \models \theta_i$ and $T_i(G^*) = \emptyset$ otherwise. Let $K = \mathsf{I}_{\mathsf{T}_\psi}(G^*)$. We define the Σ_{P}-expansion K^+ of K by defining, for each $j \in [n_2]$, $Z_j(K^+) = \zeta_j(G^+)$. By the assumption that $\models \forall x \bigwedge_{i \neq j} \neg(\zeta_i(x) \wedge \zeta_j(x))$ the Z_j are pairwise disjoint. Now, when $\mathrm{dist}(x,y) > 2t$ there is no edge between x and y in K, hence φ on K^+ is equivalent to $\widetilde{\varphi}$ on G^*, which in turn in this case is equivalent to $\widetilde{\eta}(x,y)$ on G^*. On the other hand, when $\mathrm{dist}(x,y) \leq 2t$, then the perturbation is applied to the edges defined by $\neg(\widetilde{\eta}(x,y) \leftrightarrow \widetilde{\varphi}(x,y))$, which yields exactly the edges defined by $\widetilde{\eta}$ on G^*. Thus we have $\eta(G^+) = \widetilde{\eta}(G^*) = \varphi(K^+)$, hence $\mathsf{I}_{\mathsf{P}}(K^+) = H$.

It follows that the transduction T is subsumed by the composition of the immersive transduction T_ψ and a sequence of subset complementations, the perturbation P. ◀

▶ **Corollary 3.** *For every immersive transduction T and every perturbation P, there exist immersive transduction T' and a perturbation P', such that $\mathsf{P}' \circ \mathsf{T}'$ subsumes $\mathsf{T} \circ \mathsf{P}$.*

3.2 Immersive transductions

Intuitively, copying operations and perturbations are simple operations. The main complexity of a transduction is captured by its immersive part. The strongly local character of immersive transductions is the key tool in our further analysis. It will be very useful to give another (seemingly) weaker property for the existence of an immersive transduction in another class, which is the existence of a transduction that does not shrink the distances too much, as we prove now.

▶ **Lemma 4.** *Assume there is a non-copying transduction T encoding $\mathscr{C}$ in $\mathscr{D}$ with associated interpretation I and an $\epsilon > 0$ with the property that for every $H \in \mathscr{C}$ and $G \in \mathscr{D}$ with $H \in \mathsf{T}(G)$ we have $\mathrm{dist}_H(u,v) \geq \epsilon\, \mathrm{dist}_G(u,v)$ (for all $u, v \in V(H)$). Then there exists an immersive transduction encoding $\mathscr{C}$ in $\mathscr{D}$ that subsumes T.*

Proof. Let $\mathsf{T} = \mathsf{I} \circ \Gamma_\Sigma$ with $\mathsf{I} = (\nu(x), \eta(x,y))$. By Gaifman's locality theorem, there is a set $\Sigma' \supseteq \Sigma$ of unary relations and a formula $\varphi(x,y)$, such that for every Σ-expanded graph G^+ there is a Σ'-expansion G^* of G^+ with $G^* \models \varphi(x,y)$ if and only if $G^+ \models \eta(x,y)$, where φ is t-local for some t (as in the proof of Theorem 2). We further define a new mark M and let $\mathsf{I}' = (M(x), \varphi(x,y) \wedge \mathrm{dist}(x,y) \leq 1/\epsilon)$. The transduction $\mathsf{T}' = \mathsf{I}' \circ \Gamma_{\Sigma' \cup \{M\}}$ is immersive and subsumes the transduction T. ◀

Recall that $G + K_1$ is obtained from G by adding a new vertex, called an *apex*, that is connected to all vertices of G. Of course, by adding an apex we shrink all distances in G. The next lemma shows that when we can transduce $\mathscr{C} + K_1$ in a class $\mathscr{F}$ with an immersive transduction, then we can in fact transduce $\mathscr{C}$ in the local balls of $\mathscr{F}$.

▶ **Lemma 5.** *Let $\mathscr{C}, \mathscr{F}$ be graph classes, and let T be an immersive transduction encoding a class $\mathscr{D}$ in $\mathscr{F}$ with $\mathscr{D} \supseteq \{G + K_1 \mid G \in \mathscr{C}\}$. Then there exists an integer r such that $\mathscr{C} \sqsubseteq^\circ_{\mathrm{FO}} \mathcal{B}^{\mathscr{F}}_r$.*

Proof. Let $\mathsf{T} = \mathsf{I} \circ \Gamma_\Sigma$ be an immersive transduction encoding $\mathscr{D}$ in $\mathscr{F}$. For every graph $G \in \mathscr{C}$ there exists a graph $F \in \mathscr{F}$ such that $G + K_1 = \mathsf{I}(F^+)$, where F^+ is a Σ-expansion of F. Let v be the apex of $G + K_1$. By the strong locality of I we get $\mathsf{I}(F^+) = \mathsf{I}(B_r^{F^+}(v))$ for some fixed r depending only on T. Let U be a transduction allowing to take an induced subgraph, then G can be encoded in the class $\mathcal{B}_r^{\mathscr{C}}$ by the non-copying transduction $\mathsf{U} \circ \mathsf{T}$. ◀

Finally, we show that when transducing an additive class $\mathscr{C}$ in a class $\mathscr{D}$, then we do not need perturbations at all.

▶ **Lemma 6.** *Let $\mathscr{C}$ be an additive class with $\mathscr{C} \sqsubseteq^\circ_{\mathrm{FO}} \mathscr{D}$. Then there exists an immersive transduction encoding $\mathscr{C}$ in $\mathscr{D}$.*

Proof. According to Theorem 2, the transduction of $\mathscr{C}$ in $\mathscr{D}$ is subsumed by the composition of an immersive transduction T (with associated interpretation $\mathsf{I} = (\nu, \eta)$) and a perturbation (with associated interpretation I_P). As η is strongly local there exists r such that for all $G \in \mathscr{D}$ and Σ_{T}-expansions G^+ and all $u, v \in \mathsf{I}(G^+)$ we have $\mathrm{dist}_{\mathsf{I}(G^+)}(u, v) \geq \mathrm{dist}_G(u, v)/r$. Let c be the number of unary relations used in the perturbation. Let H be a graph in $\mathscr{C}$, let $n > 3 \cdot c^{|H|}$ and let $K = nH$ (n disjoint copies of H). By assumption there exists an expansion G^+ of a graph G in $\mathscr{D}$ with $K = \mathsf{I}_P \circ \mathsf{I}(G^+)$. By the choice of n, at least 3 copies H_1, H_2, and H_3 of H in K satisfy the same unary predicates at the same vertices. For $a \in \{1, 2, 3\}$ and $v \in V(H_1)$, we denote by $\tau_a(v)$ the vertex of H_a corresponding to the vertex v of H_1 ($\tau_1(v)$ being the vertex v itself). Let u, v be adjacent vertices of H_1. Assume that u and v have distance greater than r in G. Then u and v are made adjacent in K by the perturbation P (the edge cannot have been created by η as it is strongly r-local). As $\tau_a(u)$ is not adjacent with $\tau_b(v)$ for $b \neq a$ there must be paths of length at most r linking $\tau_a(u)$ with $\tau_b(v)$ in G for $a \neq b$ (the interpretation I must have introduced an edge that the perturbation removed again). This however implies that there is a path of length at most $3r$ between u and v in G (going from u to $\tau_2(v)$ to $\tau_3(u)$ to v). It follows that for all $u, v \in V(K)$ we have $\mathrm{dist}_K(u, v) \geq \mathrm{dist}_G(u, v)/(3r)$. Hence the transduction obtained by composing T with the extraction of the induced subgraph H_1 implies the existence of an immersive transduction of $\mathscr{C}$ in $\mathscr{D}$, according to Lemma 4. ◀

▶ **Corollary 7** (Elimination of the perturbation)**.** *Let $\mathscr{C}$ be an additive class with $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$. Then there exists a copy operation C and an immersive transduction $\mathsf{T}_{\mathrm{imm}}$ such that $\mathsf{T}_{\mathrm{imm}} \circ \mathsf{C}$ is a transduction encoding $\mathscr{C}$ in $\mathscr{D}$.*

4 Some applications of the local normal form

4.1 Transductions in paths

▶ **Theorem 8.** *A class $\mathscr{C}$ is* FO *transduction equivalent to the class of paths if and only if it is a perturbation of a class with bounded bandwidth that contains graphs with arbitrarily large connected components.*

Proof. Assume T is a transduction of $\mathscr{C}$ in $\mathcal{P}$. According to Theorem 2, $\mathsf{T} \leq \mathsf{P} \circ \mathsf{T}_{\mathrm{imm}} \circ \mathsf{C}_k$, where $k \geq 1$, $\mathsf{T}_{\mathrm{imm}}$ is immersive, and P is a perturbation. Observe first that $\mathsf{C}_k(\mathcal{P})$ is included in the class of all subgraphs of the $(k+1)$-power of paths. By the strong locality property of immersive transductions, every class obtained from $\mathcal{P}$ by the composition of a copy operation and an immersive transduction has its image included in the class of all the subgraphs of the ℓ-power of paths, for some integer ℓ depending only on the transduction, hence, in a class of bounded bandwidth. Conversely, assume that $\mathscr{C}$ is a perturbation of a

class $\mathscr{D}$ containing graphs with bandwidth at most ℓ that contains graphs with arbitrarily large connected components. Then $\mathscr{D}$ is a subclass of the monotone closure (containing all subgraphs of the class) of the class $\mathcal{P}^\ell$ of ℓ-powers of paths, which has bounded star chromatic number. We show in the full version of this paper [17] that we can obtain the monotone closure of a class with bounded star chromatic number as a transduction. By this result and the observation that taking the ℓ-power is obviously a transduction, we get that $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{P}$. To see that vice versa $\mathcal{P} \sqsubseteq_{\mathrm{FO}} \mathscr{C}$ observe that we can first undo the perturbation by carrying out the edge complementations in reverse order. Then we have arbitrarily large connected components, which in a graph of bounded bandwidth have unbounded diameter. From this we can transduce arbitrarily long paths by extracting an induced subgraph. ◀

4.2 Local monadically stable, straight, and dependent classes

A class $\mathscr{C}$ is *locally monadically dependent* if, for every integer r, the class $\mathcal{B}_r^{\mathscr{C}}$ is monadically dependent; a class $\mathscr{C}$ is *locally monadically stable* if, for every integer r, the class $\mathcal{B}_r^{\mathscr{C}}$ is monadically stable; a class $\mathscr{C}$ is *locally monadically straight* if, for every integer r, the class $\mathcal{B}_r^{\mathscr{C}}$ is monadically straight.

▶ **Theorem 9.** *For a class $\mathscr{C}$ of graphs we have the following equivalences:*

1. *$\mathscr{C}$ is locally monadically dependent if and only if $\mathscr{C}$ is monadically dependent;*
2. *$\mathscr{C}$ is locally monadically straight if and only if $\mathscr{C}$ is monadically straight;*
3. *$\mathscr{C}$ is locally monadically stable if and only if $\mathscr{C}$ is monadically stable.*

Proof. The proof will follow from the following claim.

▷ Claim 10. Let $\mathscr{C}$ be a class such that the class $\mathscr{C}' = \{n(G + K_1) \mid n \in \mathbb{N}, G \in \mathscr{C}\}$ is a transduction of $\mathscr{C}$. Then, for every class $\mathscr{D}$ we have $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$ if and only if there exists some integer r with $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{B}_r^{\mathscr{D}}$.

Proof. Obviously, if there exists some integer r with $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{B}_r^{\mathscr{D}}$, then $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$. Now assume $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{D}$. As $\mathscr{C}' \equiv_{\mathrm{FO}} \mathscr{C}$ we prove as in Lemma 6 that there is a transduction of $\mathscr{C}'$ in $\mathscr{D}$ that is the composition of a copy operation C and an immersive transduction T. Let $\mathscr{D}' = \mathsf{C}(\mathscr{D})$. According to Lemma 5, there is an integer r such that $\mathscr{C} \sqsubseteq^{\circ}_{\mathrm{FO}} \mathcal{B}_r^{\mathscr{D}'}$ thus, as $\mathcal{B}_r^{\mathscr{D}'} = \mathsf{C}(\mathcal{B}_r^{\mathscr{D}})$, we have $\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathcal{B}_r^{\mathscr{D}}$. ◁

The class $\{n(G + K_1) \mid n \in \mathbb{N}, G \in \mathcal{G}\}$ is obviously a transduction of $\mathcal{G}$. Hence, according to Claim 10, a class $\mathscr{C}$ is locally monadically dependent if and only if it is monadically dependent. The class $\{n(G + K_1) \mid n \in \mathbb{N}, G \in \mathcal{TP}\}$ is a transduction of $\mathcal{TP}$. Hence, according to Claim 10, a class $\mathscr{C}$ is locally monadically straight if and only if it is monadically straight. The class $\{n(G + K_1) \mid n \in \mathbb{N}, G \in \mathcal{H}\}$ is a transduction of $\mathcal{H}$. Hence, according to Claim 10, a class $\mathscr{C}$ is locally monadically stable if and only if it is monadically stable. ◀

▶ **Example 11.** Although the class of unit interval graphs has unbounded clique-width, every proper hereditary subclass of unit interval graphs has bounded clique-width [12]. This is in particular the case for the class of unit interval graphs with bounded radius. As classes with bounded clique-width are monadically dependent, the class of unit interval graphs is locally monadically dependent, hence monadically dependent.

5 Structural properties of the transduction quasiorders

Many properties will be similar when considering $\sqsubseteq_{\mathrm{FO}}$ and $\sqsubseteq^{\circ}_{\mathrm{FO}}$. To avoid unnecessary repetitions of the statements and arguments, we shall use the notations $\sqsubseteq, \sqsubset, \equiv$ to denote either $\sqsubseteq^{\circ}_{\mathrm{FO}}, \sqsubset^{\circ}_{\mathrm{FO}}, \equiv^{\circ}_{\mathrm{FO}}$ or $\sqsubseteq_{\mathrm{FO}}, \sqsubset_{\mathrm{FO}}, \equiv_{\mathrm{FO}}$.

For two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ define $\mathscr{C}_1 + \mathscr{C}_2 = \{G_1 \cup G_2 : G_1 \in \mathscr{C}_1, G_2 \in \mathscr{C}_2\}$. A class $\mathscr{C}$ is *additive* if $\mathscr{C} + \mathscr{C} \equiv \mathscr{C}$. For instance, every class closed under disjoint union is additive, while the class of all stars and all paths is not additive. Note that if $\mathscr{C}_1$ and $\mathscr{C}_2$ are additive then $\mathscr{C}_1 + \mathscr{C}_2$ is also additive. We further say that a class $\mathscr{C}$ is *essentially connected* if it is equivalent to the subclass $\mathrm{Conn}(\mathscr{C})$ of all its connected graphs.

In this section we will consider the quasiorders $\sqsubseteq^{\circ}_{\mathrm{FO}}$ and $\sqsubseteq_{\mathrm{FO}}$, as well as their restrictions to additive classes of graphs. Let $\mathfrak{C}$ be the collection of all graph classes, and let $\mathfrak{A}$ be the collection of all additive graph classes. While speaking about these quasiorders, we will implicitly consider their quotient by the equivalence relation $\equiv$, which are partial orders. For instance, when we say that $(\mathfrak{C}, \sqsubseteq)$ is a join-semilattice, we mean that $(\mathfrak{C} / \equiv, \sqsubseteq)$ is a join semilattice. The symbol $\lhd$ will always been used with reference to $(\mathfrak{C}, \sqsubseteq)$, $\mathscr{C} \lhd \mathscr{D}$ expressing that there exist no class $\mathscr{F}$ with $\mathscr{C} \sqsubset \mathscr{F} \sqsubset \mathscr{D}$. When we shall consider covers in $(\mathfrak{A}, \sqsubseteq)$ we will say explicitly that $(\mathscr{C}, \mathscr{D})$ is a cover in $(\mathfrak{A}, \sqsubseteq)$, expressing that there exists no additive class $\mathscr{F}$ with $\mathscr{C} \sqsubset \mathscr{F} \sqsubset \mathscr{D}$.

5.1 The transduction semilattices $(\mathfrak{C}, \sqsubseteq^{\circ}_{\mathrm{FO}})$ and $(\mathfrak{C}, \sqsubseteq_{\mathrm{FO}})$

The aim of this section is to prove that $(\mathfrak{C}, \sqsubseteq^{\circ}_{\mathrm{FO}})$ and $(\mathfrak{C}, \sqsubseteq_{\mathrm{FO}})$ are distributive join-semilattices and to state some of their properties.

▶ **Lemma 12.** *If $\mathscr{D} \sqsubseteq \mathscr{C}_1 \cup \mathscr{C}_2$, then there is a partition $\mathscr{D}_1 \cup \mathscr{D}_2$ of $\mathscr{D}$ with $\mathscr{D}_1 \sqsubseteq \mathscr{C}_1$ and $\mathscr{D}_2 \sqsubseteq \mathscr{C}_2$. If $\mathscr{D}$ is additive, then $\mathscr{D} \sqsubseteq \mathscr{C}_1 \cup \mathscr{C}_2 \iff \mathscr{D} \sqsubseteq \mathscr{C}_1$ or $\mathscr{D} \sqsubseteq \mathscr{C}_2$.*

Proof. The first statement is straightforward. We now prove the second statement. For an integer n, let G_n be the disjoint union of all the graphs in $\mathscr{D}$ with at most n vertices.

Assume $\mathscr{D}$ is additive and $\mathscr{D} \sqsubseteq \mathscr{C}_1 \cup \mathscr{C}_2$. According to the first statement, there exists a partition $\mathscr{D}_1, \mathscr{D}_2$ of $\mathscr{D}$ with $\mathscr{D}_1 \sqsubseteq \mathscr{C}_1$ and $\mathscr{D}_2 \sqsubseteq \mathscr{C}_2$. For $G \in \mathscr{D}$ define $\mathscr{S}(G) = \{H \cup G : H \in \mathscr{D}\}$. Note that $\mathscr{S}(G) \subseteq \mathscr{D} + \mathscr{D}$. Let $\mathscr{D}' = \mathscr{D} + \mathscr{D}$. As $\mathscr{D}' \sqsubseteq \mathscr{D}_1 \cup \mathscr{D}_2$ there exists a partition $\mathscr{D}'_1, \mathscr{D}'_2$ of $\mathscr{D}'$ with $\mathscr{D}'_1 \sqsubseteq \mathscr{D}_1$ and $\mathscr{D}'_2 \sqsubseteq \mathscr{D}_2$. If, for every $G \in \mathscr{D}$ we have $\mathscr{S}(G) \cap \mathscr{D}'_1 \neq \emptyset$ then $\mathscr{D} \sqsubseteq \mathscr{D}'_1$ (by the generic transduction extracting an induced subgraph) thus $\mathscr{D} \equiv \mathscr{D}_1 \sqsubseteq \mathscr{C}_1$. Similarly, if for every $G \in \mathscr{C}$ we have $\mathscr{S}(G) \cap \mathscr{D}'_2 \neq \emptyset$ then $\mathscr{D} \sqsubseteq \mathscr{C}_2$. Assume for contradiction that there exist $G_1, G_2 \in \mathscr{D}$ with $\mathscr{S}(G_i) \cap \mathscr{D}'_i = \emptyset$. Then $G_1 \cup G_2$ belongs neither to $\mathscr{D}'_1$ nor to $\mathscr{D}'_2$, contradicting the assumption that $\mathscr{D}'_1, \mathscr{D}'_2$ is a partition of $\mathscr{D}' = \mathscr{D} + \mathscr{D}$. ◀

▶ **Lemma 13.** *If $\mathscr{C}_1$ and $\mathscr{C}_2$ are incomparable, then $\mathscr{C}_1 \cup \mathscr{C}_2$ is not equivalent to an additive class. In particular, $\mathscr{C}_1 \cup \mathscr{C}_2 \not\equiv \mathscr{C}_1 + \mathscr{C}_2$.*

Proof. We prove by contradiction that $\mathscr{C}_1 \cup \mathscr{C}_2$ is not equivalent to an additive class. Assume that we have $\mathscr{D} \sqsubseteq \mathscr{C}_1 \cup \mathscr{C}_2$, where $\mathscr{D}$ is additive. According to Lemma 12 we have $\mathscr{D} \sqsubseteq \mathscr{C}_1$ or $\mathscr{D} \sqsubseteq \mathscr{C}_2$ thus if $\mathscr{C}_1 \cup \mathscr{C}_2 \sqsubseteq \mathscr{D}$, then $\mathscr{C}_2 \sqsubseteq \mathscr{C}_1$ or $\mathscr{C}_1 \sqsubseteq \mathscr{C}_2$, contradicting the hypothesis that $\mathscr{C}_1$ and $\mathscr{C}_2$ are incomparable. ◀

▶ **Theorem 14.** *The quasiorder $(\mathfrak{C}, \sqsubseteq)$ is a distributive join-semilattice, where the join of $\mathscr{C}_1$ and $\mathscr{C}_2$ is $\mathscr{C}_1 \cup \mathscr{C}_2$. In this quasiorder, additive classes are join-irreducible. This quasiorder has a minimum $\mathcal{E}$ and a maximum $\mathcal{G}$.*

Proof. Of course we have $\mathscr{C}_1 \sqsubseteq \mathscr{C}_1 \cup \mathscr{C}_2$ and $\mathscr{C}_2 \sqsubseteq \mathscr{C}_1 \cup \mathscr{C}_2$. Now assume $\mathscr{D}$ is such that $\mathscr{C}_1 \sqsubseteq \mathscr{D}$ and $\mathscr{C}_2 \sqsubseteq \mathscr{D}$. Let T_1 and T_2 be transductions encoding $\mathscr{C}_1$ and $\mathscr{C}_2$ in $\mathscr{D}$, with associated interpretations $\mathsf{I}_1 = (\nu_1, \eta_1)$ and $\mathsf{I}_2 = (\nu_1, \eta_1)$. By relabeling the colors, we can assume that the set Σ_1 of unary relations used by I_1 is disjoint from the set Σ_2 of unary relations used by I_2. Without loss of generality, we have $\mathsf{T}_1 = \mathsf{I}_1 \circ \Gamma_{\Sigma_1} \circ \mathsf{C}$ and $\mathsf{T}_2 = \mathsf{I}_2 \circ \Gamma_{\Sigma_2} \circ \mathsf{C}$, where C is a copying operation if $\sqsubseteq$ is $\sqsubseteq_{\mathrm{FO}}$, or the identity mapping if $\sqsubseteq$ is $\sqsubseteq^\circ_{\mathrm{FO}}$. Let M be a new unary relation. We define the interpretation $\mathsf{I} = (\nu, \eta)$ by $\nu := \big((\exists v\ M(v)) \wedge \nu_1\big) \vee \big(\neg(\exists v\ M(v)) \wedge \nu_2\big)$ and $\eta := \big((\exists v\ M(v)) \wedge \eta_1\big) \vee \big(\neg(\exists v\ M(v)) \wedge \eta_2\big)$. Let $G \in \mathscr{C}_1 \cup \mathscr{C}_2$. If $G \in \mathscr{C}_1$, then there exists a coloring H^+ of $H \in \mathsf{C}(\mathscr{D})$ with $G = \mathsf{I}_1(H^+)$. We define H^* as the expansion of H^+ where all vertices also belong to the unary relation M. Then $G = \mathsf{I}(H^*)$. Otherwise, if $G \in \mathscr{C}_2$, then there exists a coloring H^+ of $H \in \mathsf{C}(\mathscr{D})$ with $G = \mathsf{I}_2(H^+)$ thus $G = \mathsf{I}(H^+)$. As we did not introduce new copying transductions we deduce $\mathscr{C}_1 \cup \mathscr{C}_2 \sqsubseteq \mathscr{D}$. It follows that $(\mathfrak{C}, \sqsubseteq)$ is a join semi-lattice, which is distributive according to Lemma 12.

That additive classes are join-irreducible follows from Lemma 13. ◀

We now state an easy lemma on covers in distributive join-semilattices.

▶ **Lemma 15.** *Let $(X, \leq)$ be a distributive join-semilattice (with join $\vee$). If $a \lhd b$ and $b \not\leq a \vee c$, then $a \vee c \lhd b \vee c$.*

Proof. Assume $a \vee c \leq x \leq b \vee c$. As $(X, \leq)$ is distributive there exist $b' \leq b$ and $c' \leq c$ with $x = b' \vee c'$. Thus $a \leq a \vee b' \leq b$. As $a \lhd b$, either $a = a \vee b'$ (thus $b' \leq a$) and thus $x = a \vee c$, or $a \vee b' = b$ and then $b \vee c \leq a \vee b' \vee c = a \vee x \vee c = x \leq b \vee c$ thus $x = b \vee c$. Hence either $a \vee c = b \vee c$ (which would contradict $b \not\leq a \vee c$), or $a \vee c \lhd b \vee c$. ◀

▶ **Corollary 16.** *If $\mathscr{C}_1 \lhd \mathscr{C}_2$ and $\mathscr{C}_2 \not\sqsubseteq \mathscr{C}_1 \cup \mathscr{D}$, then $\mathscr{C}_1 \cup \mathscr{D} \lhd \mathscr{C}_2 \cup \mathscr{D}$.*

▶ **Corollary 17.** *If $\mathscr{C}_1 \lhd \mathscr{C}_2$, $\mathscr{C}_1 \sqsubseteq \mathscr{D}$, and $\mathscr{C}_2$ and $\mathscr{D}$ are incomparable, then $\mathscr{D} \lhd \mathscr{D} \cup \mathscr{C}_2$.*

Proof. As $\mathscr{C}_2 \not\sqsubseteq \mathscr{D}$ and $\mathscr{C}_1 \sqsubseteq \mathscr{D}$ we have $\mathscr{C}_2 \not\sqsubseteq \mathscr{D} \cup \mathscr{C}_1$. ◀

▶ **Corollary 18.** *If $\mathscr{C}_1 \lhd \mathscr{C}_2$, $\mathscr{C}_2$ and $\mathscr{D}$ are incomparable and $\mathscr{C}_2$ is additive, then $\mathscr{C}_1 \cup \mathscr{D} \lhd \mathscr{C}_2 \cup \mathscr{D}$.*

5.2 The transduction semilattices $(\mathfrak{A}, \sqsubseteq^\circ_{\mathrm{FO}})$ and $(\mathfrak{A}, \sqsubseteq_{\mathrm{FO}})$

The aim of this section is to prove that $(\mathfrak{A}, \sqsubseteq^\circ_{\mathrm{FO}})$ and $(\mathfrak{A}, \sqsubseteq_{\mathrm{FO}})$ are distributive join-semilattices and to state some of their properties.

▶ **Lemma 19.** *If $\mathscr{C}_1$ and $\mathscr{C}_2$ are incomparable, then $\mathscr{C}_1 + \mathscr{C}_2$ is not essentially connected.*

Proof. We prove by contradiction that $\mathscr{C}_1 + \mathscr{C}_2$ is not essentially connected. It is immediate that $\mathrm{Conn}(\mathscr{C}_1 + \mathscr{C}_2) \subseteq \mathscr{C}_1 \cup \mathscr{C}_2$. So if $\mathscr{C}_1 + \mathscr{C}_2$ is essentially connected, then $\mathscr{C}_1 \cup \mathscr{C}_2$ and $\mathscr{C}_1 + \mathscr{C}_2$ are equivalent, contradicting Lemma 13. ◀

▶ **Lemma 20.** *A class $\mathscr{D}$ is additive if and only if for all classes $\mathscr{C}_1, \mathscr{C}_2$ we have*

$$\mathscr{C}_1 + \mathscr{C}_2 \sqsubseteq \mathscr{D} \quad \Longleftrightarrow \quad \mathscr{C}_1 \sqsubseteq \mathscr{D} \text{ and } \mathscr{C}_2 \sqsubseteq \mathscr{D}.$$

Proof. Assume $\mathscr{D}$ is additive. If $\mathscr{C}_1 + \mathscr{C}_2 \sqsubseteq \mathscr{D}$, then $\mathscr{C}_1 \cup \mathscr{C}_2 \sqsubseteq \mathscr{D}$ thus $\mathscr{C}_1 \sqsubseteq \mathscr{D}$ and $\mathscr{C}_2 \sqsubseteq \mathscr{D}$. Conversely, assume $\mathscr{C}_1 \sqsubseteq \mathscr{D}$ and $\mathscr{C}_2 \sqsubseteq \mathscr{D}$. Then $\mathscr{C}_1 \cup \mathscr{C}_2 \sqsubseteq \mathscr{D}$. Let $\mathsf{T} = \mathsf{I} \circ \Gamma_\Sigma \circ \mathsf{C}$ be a transduction such that $\mathscr{C}_1 \cup \mathscr{C}_2 \subseteq \mathsf{T}(\mathscr{D})$, where $\mathsf{I} = (M(x), \varphi(x, y))$ with $M \in \Sigma$, and where C is either a copying operation if $\sqsubseteq$ is $\sqsubseteq_{\mathrm{FO}}$, or the identity mapping if $\sqsubseteq$ is $\sqsubseteq^\circ_{\mathrm{FO}}$. Let Σ' be

the signature obtained from Σ by adding two unary predicates $A(x)$ and $B(x)$. We define $\varphi_A(x,y)$ (resp. $\varphi_B(x,y)$) by replacing in $\varphi(x,y)$ the predicate M by the predicate A (resp. by the predicate B). Let $\varphi'(x,y) = \big(A(x) \wedge A(y) \wedge \varphi_A(x,y)\big) \vee \big(B(x) \wedge B(y) \wedge \varphi_B(x,y)\big)$, let $\mathsf{I}' = (A(x) \vee B(x), \varphi'(x,y))$, and let $\mathsf{T} = \mathsf{I}' \circ \Gamma_{\Sigma'} \circ \mathsf{C}$. Then it is easily checked that $\mathscr{C}_1 + \mathscr{C}_2 \subseteq \mathsf{T}'(\mathscr{D})$. Conversely, assume that for all classes $\mathscr{C}_1, \mathscr{C}_2$ we have $\mathscr{C}_1 + \mathscr{C}_2 \sqsubseteq \mathscr{D} \iff \mathscr{C}_1 \sqsubseteq \mathscr{D}$ and $\mathscr{C}_2 \sqsubseteq \mathscr{D}$. Then (by choosing $\mathscr{C}_1 = \mathscr{C}_2 = \mathscr{D}$) we deduce $\mathscr{D} + \mathscr{D} \equiv \mathscr{D}$. ◀

▶ **Lemma 21.** *Assume $\mathscr{D}$ is additive and $\mathscr{C}_1$ and $\mathscr{C}_2$ are incomparable. If $\mathscr{D} \sqsubseteq \mathscr{C}_1 + \mathscr{C}_2$, then there exist classes $\mathscr{D}_1$ and $\mathscr{D}_2$ such that $\mathscr{D} \equiv \mathscr{D}_1 + \mathscr{D}_2$, $\mathscr{D}_1 \sqsubseteq \mathscr{C}_1$ and $\mathscr{D}_2 \sqsubseteq \mathscr{C}_2$. Moreover, if $\mathscr{C}_1$ and $\mathscr{C}_2$ are additive we can require that $\mathscr{D}_1$ and $\mathscr{D}_2$ are additive.*

Proof. According to Corollary 7 there exists a copy operation C (which reduces to the identity if $\sqsubseteq$ is $\sqsubseteq^{\circ}_{\mathrm{FO}}$) and an immersive transduction $\mathsf{T}_{\mathrm{imm}}$ such that $\mathscr{D} \subseteq \mathsf{T}_{\mathrm{imm}} \circ \mathsf{C}(\mathscr{C}_1 + \mathscr{C}_2)$. Let I be the interpretation part of $\mathsf{T}_{\mathrm{imm}}$. Let $G \in \mathscr{D}$ and let H^+ be a coloring of $H = \mathsf{C}(K)$, with $K \in \mathscr{C}_1 + \mathscr{C}_2$ and $G = \mathsf{I}(H^+)$. As $\mathsf{T}_{\mathrm{imm}}$ is immersive, each connected component of G comes from a connected component of H^+ hence from a connected component of K. By grouping the connected components used in K by their origin ($\mathscr{C}_1$ or $\mathscr{C}_2$) we get that G is the disjoint union of $G_1 \in \mathsf{T}_{\mathrm{imm}} \circ C(K_1)$ and $G_2 \in \mathsf{T}_{\mathrm{imm}} \circ C(K_2)$, where $K_1 \in \mathscr{C}_1$ and $K_2 \in \mathscr{C}_2$. So $\mathscr{D} \sqsubseteq \mathscr{D}_1 + \mathscr{D}_2$, where $\mathscr{D}_1 \sqsubseteq \mathscr{C}_1$ and $\mathscr{D}_2 \sqsubseteq \mathscr{C}_2$. Moreover, as obviously $\mathscr{D}_1 \sqsubseteq \mathscr{D}$ and $\mathscr{D}_2 \sqsubseteq \mathscr{D}$ we derive from Lemma 20 that we have $\mathscr{D}_1 + \mathscr{D}_2 \sqsubseteq \mathscr{D}$. Hence $\mathscr{D} \equiv \mathscr{D}_1 + \mathscr{D}_2$. For $i = 1, 2$, if $\mathscr{C}_i$ is additive, then we can assume that $\mathscr{D}_i$ is also additive. ◀

▶ **Corollary 22.** *If $\mathscr{D}$ is additive and essentially connected, then*

$$\mathscr{D} \sqsubseteq \mathscr{C}_1 + \mathscr{C}_2 \quad \iff \quad \mathscr{D} \sqsubseteq \mathscr{C}_1 \text{ or } \mathscr{D} \sqsubseteq \mathscr{C}_2.$$

Proof. According to Lemma 21 there exist $\mathscr{D}_1, \mathscr{D}_2$ with $\mathscr{D} \equiv \mathscr{D}_1 + \mathscr{D}_2$, $\mathscr{D}_1 \sqsubseteq \mathscr{C}_1$ and $\mathscr{D}_2 \sqsubseteq \mathscr{C}_2$. However, as $\mathscr{D}$ is essentially connected, $\mathscr{D}_1$ and $\mathscr{D}_2$ cannot be incomparable. Thus $\mathscr{D} \sqsubseteq \mathscr{C}_1$ or $\mathscr{D} \sqsubseteq \mathscr{C}_2$. ◀

▶ **Theorem 23.** *The quasiorder $(\mathfrak{A}, \sqsubseteq)$ is a distributive join-semilattice, where the join of $\mathscr{C}_1$ and $\mathscr{C}_2$ is $\mathscr{C}_1 + \mathscr{C}_2$. In this quasiorder, essentially connected (additive) classes are join-irreducible. This quasiorder has a minimum $\mathcal{E}$ and a maximum $\mathcal{G}$.*

Proof. That $(\mathfrak{A}, \sqsubseteq)$ is a join-semilattice follows from Lemma 20; that it is distributive follows from Lemma 21. The last statement follows from Lemma 19. ◀

▶ **Corollary 24.** *Assume that $\mathscr{C}_1$ and $\mathscr{C}_2$ are incomparable and additive, $\mathscr{D}$ is additive and essentially connected, $\mathscr{C}_1 \sqsubseteq \mathscr{D}$, and $\mathscr{C}_2 \sqsubseteq \mathscr{D}$. Then we have*

$$\mathscr{C}_1 \cup \mathscr{C}_2 \sqsubset \mathscr{C}_1 + \mathscr{C}_2 \sqsubset \mathscr{D}.$$

▶ **Corollary 25.** *Assume that $\mathscr{C}_1$ and $\mathscr{C}_2$ are incomparable and additive, $\mathscr{D}$ is additive and essentially connected, $\mathscr{D}$ is incomparable with $\mathscr{C}_1$ and $\mathscr{C}_2 \sqsubset \mathscr{D}$. Then $\mathscr{C}_1 + \mathscr{C}_2$ is incomparable with $\mathscr{D}$.*

Proof. Assume for contradiction that $\mathscr{C}_1 + \mathscr{C}_2 \sqsubseteq \mathscr{D}$. According to Theorem 23, we have $\mathscr{C}_1 \sqsubseteq \mathscr{D}$, contradicting the assumption that $\mathscr{D}$ is incomparable with $\mathscr{C}_1$.

Assume for contradiction that $\mathscr{D} \sqsubseteq \mathscr{C}_1 + \mathscr{C}_2$. According to Theorem 23 there exists (by distributivity) classes $\mathscr{D}_1$ and $\mathscr{D}_2$ with $\mathscr{D}_1 \sqsubseteq \mathscr{C}_1$, $\mathscr{D}_2 \sqsubseteq \mathscr{C}_2$, and $\mathscr{D} = \mathscr{D}_1 + \mathscr{D}_2$. As $\mathscr{D}$ is essentially connected, according to Theorem 23 it is join-irreducible. Thus $\mathscr{D}_1$ and $\mathscr{D}_2$ are comparable. Thus $\mathscr{D} \subseteq \mathscr{C}_1$ or $\mathscr{D} \subseteq \mathscr{C}_2$. The first case does not hold as $\mathscr{D}$ is incomparable with $\mathscr{C}_1$, and the second case does not hold as $\mathscr{C}_2 \sqsubset \mathscr{D}$. ◀

Using the distributive join-semillatice structure of $(\mathfrak{A}, \sqsubseteq)$, the following corollaries follow from Lemma 15.

▶ **Corollary 26.** *In the poset $(\mathfrak{A}, \sqsubseteq)$, if $(\mathscr{C}_1, \mathscr{C}_2)$ is a cover and $\mathscr{C}_2 \not\sqsubseteq \mathscr{C}_1 + \mathscr{D}$ then $(\mathscr{C}_1 + \mathscr{D}, \mathscr{C}_2 + \mathscr{D})$ is a cover.*

▶ **Corollary 27.** *If $(\mathscr{C}_1, \mathscr{C}_2)$ is a cover of $(\mathfrak{A}, \sqsubseteq)$, $\mathscr{C}_1 \sqsubseteq \mathscr{D}$, and $\mathscr{C}_2$ and $\mathscr{D}$ are incomparable, then $(\mathscr{D}, \mathscr{D} + \mathscr{C}_2)$ is a cover of $(\mathfrak{A}, \sqsubseteq)$.*

▶ **Corollary 28.** *If $(\mathscr{C}_1, \mathscr{C}_2)$ is a cover of $(\mathfrak{A}, \sqsubseteq)$, $\mathscr{C}_2$ and $\mathscr{D}$ are incomparable, and $\mathscr{C}_2$ is essentially connected, then $(\mathscr{D}, \mathscr{D} + \mathscr{C}_2)$ is a cover of $(\mathfrak{A}, \sqsubseteq)$.*

6 The transduction quasiorder on some classes

In this section we consider the poset $(\mathfrak{C}, \sqsubseteq_{\mathrm{FO}})$. We focus on the structure of the partial order in the region of classes with bounded tree-width. A schematic view of the structure of $(\mathfrak{C}, \sqsubseteq_{\mathrm{FO}})$ on classes with tree-width at most 2 is shown Figure 2

Recall that since MSO collapses to FO on colored trees of bounded depth we have the following chain of covers $\mathcal{E} \lhd_{\mathrm{FO}} \mathcal{T}_2 \lhd_{\mathrm{FO}} \mathcal{T}_3 \lhd_{\mathrm{FO}} \ldots$. We first prove that parallel to this chain we have a chain of covers $\mathcal{E} \lhd_{\mathrm{FO}} \mathcal{P} \lhd_{\mathrm{FO}} \mathcal{P} \cup \mathcal{T}_2 \lhd_{\mathrm{FO}} \mathcal{P} \cup \mathcal{T}_3 \lhd_{\mathrm{FO}} \ldots$ and for all $n \geq 1$ we have $\mathcal{T}_n \lhd_{\mathrm{FO}} \mathcal{P} \cup \mathcal{T}_n$.

▶ **Theorem 29** (see [17] for the proof). *We have $\mathcal{E} \lhd_{\mathrm{FO}} \mathcal{P}$ and, for every $n \geq 1$, the chain of covers*

$$(\mathcal{P} + \mathcal{T}_n) \lhd_{\mathrm{FO}} (\mathcal{P} + \mathcal{T}_n) \cup \mathcal{T}_{n+1} \lhd_{\mathrm{FO}} (\mathcal{P} + \mathcal{T}_n) \cup \mathcal{T}_{n+2} \lhd_{\mathrm{FO}} \ldots$$

In particular, for $n = 1$ we get $\mathcal{P} \lhd_{\mathrm{FO}} \mathcal{P} \cup \mathcal{T}_2 \lhd_{\mathrm{FO}} \mathcal{P} \cup \mathcal{T}_3 \lhd_{\mathrm{FO}} \ldots$
Moreover, for all $n \geq 1$ we have $\mathcal{T}_n \lhd_{\mathrm{FO}} \mathcal{P} \cup \mathcal{T}_n$.

The difficult part of the next theorem is to prove that $\mathcal{T}_{n+2} \not\sqsubseteq_{\mathrm{FO}} \mathcal{PW}_n$. We use that the class $\mathcal{T}_{n+2}$ is additive, which by Corollary 7 implies that we can eliminate perturbations and focus on immersive transductions. This allows us to consider host graphs in $\mathcal{PW}_n$ that have bounded radius, where we can find a small set of vertices whose removal decreases the pathwidth. We encode the adjacency to these vertices by colors and proceed by induction.

▶ **Theorem 30** (see [17] for the proof). *For $n \geq 1$ we have $\mathcal{T}_{n+1} \sqsubset_{\mathrm{FO}} \mathcal{PW}_n$ but $\mathcal{T}_{n+2} \not\sqsubseteq_{\mathrm{FO}} \mathcal{PW}_n$. Consequently, for $m > n \geq 1$ we have*

$$\mathcal{T}_m + \mathcal{PW}_n \lhd_{\mathrm{FO}} (\mathcal{T}_m + \mathcal{PW}_n) \cup \mathcal{T}_{m+1} \lhd_{\mathrm{FO}} (\mathcal{T}_m + \mathcal{PW}_n) \cup \mathcal{T}_{m+2} \lhd_{\mathrm{FO}} \ldots$$
$$\mathcal{T}_m + \mathcal{PW}_n \lhd_{\mathrm{FO}} (\mathcal{T}_m + \mathcal{PW}_n) \cup \mathcal{T}_{m+1} \sqsubset_{\mathrm{FO}} \mathcal{T}_{m+1} + \mathcal{PW}_n.$$

In particular, fixing $m = n + 1$ we get that for $n \geq 1$ we have

$$\mathcal{PW}_n \lhd_{\mathrm{FO}} \mathcal{PW}_n \cup \mathcal{T}_{n+2} \lhd_{\mathrm{FO}} \mathcal{PW}_n \cup \mathcal{T}_{n+3} \lhd_{\mathrm{FO}} \cdots \sqsubset_{\mathrm{FO}} \mathcal{PW}_n \cup \mathcal{T}$$
$$\mathcal{PW}_n \lhd_{\mathrm{FO}} \mathcal{PW}_n \cup \mathcal{T}_{n+2} \sqsubset_{\mathrm{FO}} \mathcal{T}_{n+2} + \mathcal{PW}_n \sqsubset_{\mathrm{FO}} \mathcal{PW}_{n+1}.$$

▶ **Theorem 31** (see [17] for the proof). *For $m > n \geq 2$, $\mathcal{T}_m + \mathcal{PW}_n$ is incomparable with $\mathcal{T}$. Consequently, we have*

$$\mathcal{T} \sqsubset_{\mathrm{FO}} \mathcal{T} \cup \mathcal{PW}_2 \sqsubset_{\mathrm{FO}} \mathcal{T} \cup (\mathcal{T}_4 + \mathcal{PW}_2) \sqsubset_{\mathrm{FO}} \cdots \sqsubset_{\mathrm{FO}} \mathcal{T} + \mathcal{PW}_2 \sqsubset_{\mathrm{FO}} \mathcal{TW}_2.$$

With the above results in hand we obtain for $(\mathfrak{C}, \sqsubseteq_{\mathrm{FO}})$ and $(\mathfrak{A}, \sqsubseteq_{\mathrm{FO}})$ the structures sketched in Figure 2.

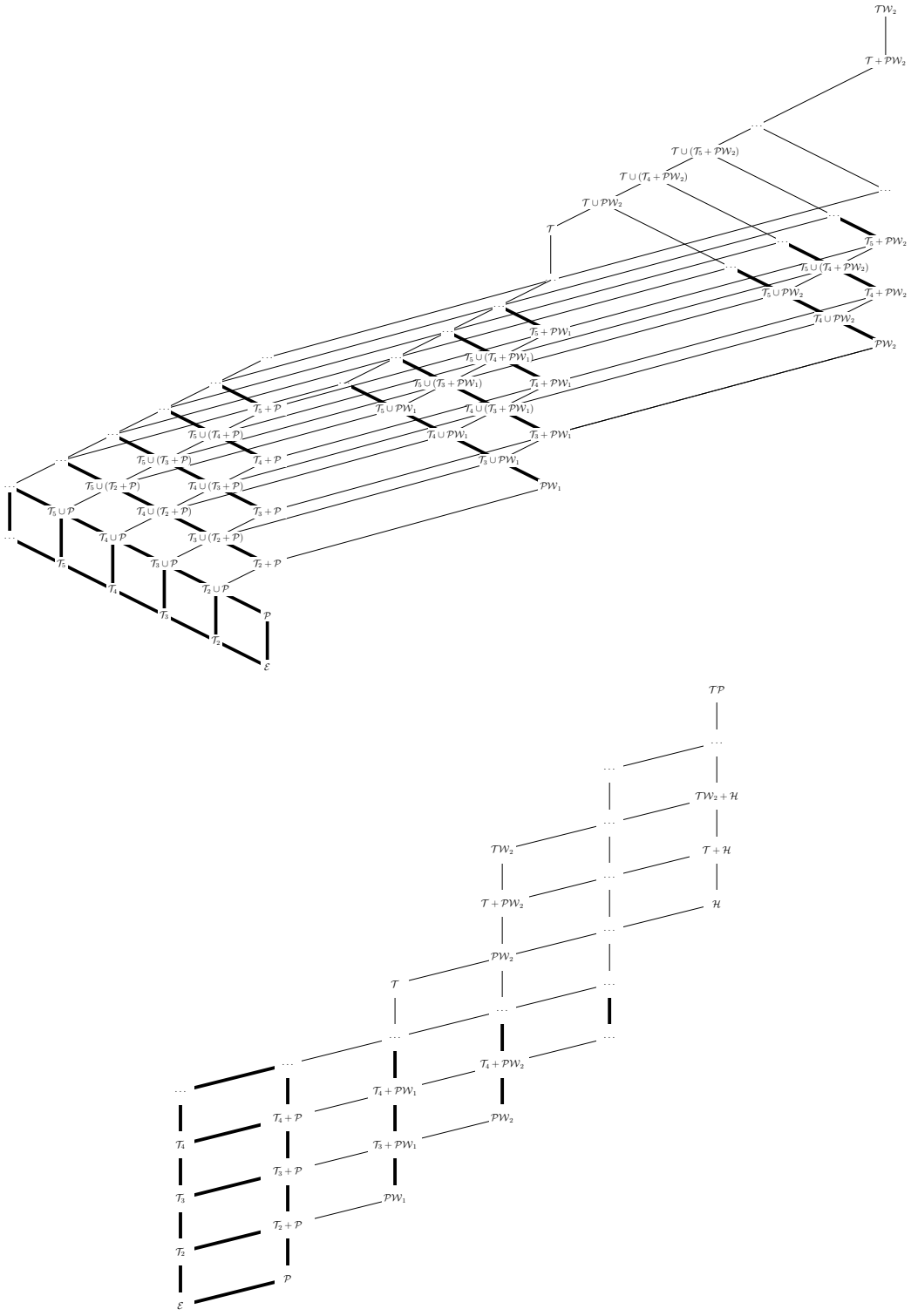


Figure 2 A fragment of $(\mathfrak{C}, \sqsubseteq_{\mathrm{FO}})$ (top) and a fragment of $(\mathfrak{A}, \sqsubseteq_{\mathrm{FO}})$ (bottom). Thick edges are covers.

References

1 J. T. Baldwin and S. Shelah. Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic*, 26(3):229–303, 1985.

2 A. Blumensath and B. Courcelle. On the monadic second-order transduction hierarchy. *Logical Methods in Computer Science*, 6(2), 2010. doi:10.2168/LMCS-6(2:2)2010.

3 T. Colcombet. A combinatorial theorem for trees. In *International Colloquium on Automata, Languages, and Programming*, pages 901–912. Springer, 2007.

4 B. Courcelle and S. Oum. Vertex-minors, monadic second-order logic, and a conjecture by seese. *Journal of Combinatorial Theory, Series B*, 97(1):91–126, 2007.

5 R. Diestel. Graph theory, volume 173 of. *Graduate texts in mathematics*, page 7, 2012.

6 H. Gaifman. On local and non-local properties. *Studies in Logic and the Foundations of Mathematics*, 107:105–135, 1982.

7 J. Gajarský, S. Kreutzer, J. Nešetřil, P. Ossona de Mendez, M. Pilipczuk, S. Siebertz, and S. Toruńczyk. First-order interpretations of bounded expansion classes. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 126:1–126:14, 2018.

8 R. Ganian, P. Hliněný, J. Nešetřil, J. Obdržálek, and P. Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Logical Methods in Computer Science*, 15(1), 2019.

9 R. Ganian, P. Hliněný, J. Nešetřil, J. Obdržálek, P. Ossona de Mendez, and R. Ramadurai. When trees grow low: Shrubs and fast MSO_1. In *International Symposium on Mathematical Foundations of Computer Science*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer-Verlag, 2012.

10 W. Hodges. *Model theory*, volume 42. Cambridge University Press, 1993.

11 O. Kwon, R. McCarty, S. Oum, and P. Wollan. Obstructions for bounded shrub-depth and rank-depth. *CoRR*, abs/1911.00230, 2019. arXiv:1911.00230.

12 V. V. Lozin. From tree-width to clique-width: Excluding a unit interval graph. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation*, pages 871–882, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

13 J. Nešetřil and P. Ossona de Mendez. Cluster analysis of local convergent sequences of structures. *Random Structures & Algorithms*, 51(4):674–728, 2017.

14 J. Nešetřil, P. Ossona de Mendez, M. Pilipczuk, R. Rabinovich, and S. Siebertz. Rankwidth meets stability. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2014–2033, 2021. doi:10.1137/1.9781611976465.120.

15 J. Nešetřil, P. Ossona de Mendez, R. Rabinovich, and S. Siebertz. Linear rankwidth meets stability. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 1180–1199, 2020.

16 J. Nešetřil, P. Ossona de Mendez, R. Rabinovich, and S. Siebertz. Classes of graphs with low complexity: The case of classes with bounded linear rankwidth. *European Journal of Combinatorics*, 91:103223, 2021. Special issue dedicated to Xuding Zhu's 60th birthday. doi:10.1016/j.ejc.2020.103223.

17 J. Nešetřil, P. Ossona de Mendez, and S. Siebertz. Structural properties of the first-order transduction quasiorder, 2021. arXiv:2010.02607.

18 D. Seese. The structure of the models of decidable monadic theories of graphs. *Annals of pure and applied logic*, 53(2):169–195, 1991.

BV and Pomset Logic Are Not the Same

Lê Thành Dũng (Tito) Nguyễn
CNRS & IRISA, Rennes, France

Lutz Straßburger
Inria Saclay & École Polytechnique, Palaiseau, France

Abstract

BV and pomset logic are two logics that both conservatively extend unit-free multiplicative linear logic by a third binary connective, which (i) is non-commutative, (ii) is self-dual, and (iii) lies between the "par" and the "tensor". It was conjectured early on (more than 20 years ago), that these two logics, that share the same language, that both admit cut elimination, and whose connectives have essentially the same properties, are in fact the same. In this paper we show that this is not the case. We present a formula that is provable in pomset logic but not in BV.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Proof theory

Keywords and phrases proof nets, deep inference, pomset logic, system BV, cographs, dicographs, series-parallel orders

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.32

1 Introduction

Pomset logic has been discovered by Christian Retoré [21] through the study of coherence spaces which form a semantics of proofs for linear logic. Retoré observed that next to the two operations $\otimes$ (*tensor* or *multiplicative conjunction*) and $\parr$ (*par* or *multiplicative disjunction*) there are two other operations $\triangleleft$ and $\triangleright$, which are non-commutative, obey $A \triangleleft B = B \triangleright A$, and are self-dual, i.e., $\langle A \triangleleft B\rangle^\perp = A^\perp \triangleleft B^\perp$.[1] From this semantic observation, Retoré derived a proof net syntax together with a correctness criterion and a cut elimination theorem. However, he could not provide a sound and complete cut-free sequent calculus for this logic [20]. Nonetheless, pomset logic has found applications in linguistics, as basis of a new categorial grammar [17], similar to the ones based on the Lambek calculus [16].

System BV was found by Alessio Guglielmi [10] through a syntactic investigation of the connectives of pomset logic and a graph theoretic study of series-parallel orders and cographs. The difficulty of presenting this combination of commutative and non-commutative connectives in the sequent calculus triggered the development of the *calculus of structures* [11], the first proper deep inference proof formalism[2]. The mixture of commutative and non-commutative connectives in BV immediately found applications in computer science, in particular, Bruscoli [3] established a strict correspondence between the proof-search space of BV and the computations in a fragment of CCS. This work was later extended by quantifiers to capture private names and to establish a correspondence of implication in (first-order) BV and a form of weak bisimulation in the π-calculus [12, 13].

[1] Observe that the order is *not* inverted, as it is the case with other non-commutative variants of linear logic [29] (see also [9, Section II.9.]).

[2] The basic idea of such a rewriting system goes back to Retoré [22] (see also [4]), but not as a proof system admitting cut-elimination.

This leads to the strange situation that we have two logics, pomset logic and BV, which are both conservative extensions of unit-free multiplicative linear logic with mix (MLL_0) [8, 7] with a non-commutative connective $\triangleleft$ such that $A \otimes B \multimap A \triangleleft B \multimap A \parr B$, which both obey a cut elimination result, and which both have found applications that lie outside of pure proof theory.

The only difference between the two logics is that pomset logic naturally extends the proof net correctness criterion of MLL_0 to the new non-commutative connective, but has no deductive proof system, whereas BV naturally extends a deductive system for MLL_0 with the new non-commutative connective, but has no proof nets. This naturally led to the conjecture that both logics ought to be the same [28]. In fact, most researchers working in this area (including the second author of this paper) believed that the two logics comprise the same set of theorems.

In this paper we show that this is not the case. More precisely, we show that the theorems of BV form a proper subset of the theorems of pomset logic. It has already been observed before [22, 28, 26] that every theorem in BV is also a theorem of pomset logic. However, the converse is not true, and we give an example of a formula that is a theorem of pomset logic but not provable in BV.

Organisation of this paper

In the next two sections we give some preliminaries on pomset logic (Section 2) and BV (Section 3). Then, in Section 4 we show that BV is contained in pomset logic. Even though this has been known since more than 20 years [22, 28], there has been no complete proof published so far. The proof we present here is a simplification of the one suggested in [28]. Next, in Section 5, we give our counterexample showing that the converse is not true, i.e., we present a formula that is a theorem of pomset logic but not provable in BV. Finally, in the conclusion (Section 6), we discuss some complexity results and give some intuition on how the counterexample has been found and why it took so long to find it.

2 Preliminaries on Pomset Logic

The **formulas** of pomset logic and BV are in this paper denoted by capital Latin letters $A, B, C, \ldots$ and are generated from a countable set $\mathcal{V} = \{a, b, c, \ldots\}$ of propositional variables and the **unit** $\mathbb{I}$ via the three binary connectives **tensor** $\otimes$, **par** $\parr$, and **seq** $\triangleleft$, according to the grammar

$$A, B \quad ::= \quad \mathbb{I} \mid a \mid a^{\perp} \mid (A \otimes B) \mid [A \parr B] \mid \langle A \triangleleft B \rangle \tag{1}$$

An **atom** is either a propositional variable or its dual. For a formula A, we define its **size** $|A|$ to be the number of atom occurrences in A. For better readability of large formulas, we use here different kinds of parentheses for the different connectives.[3] In the following, we omit outermost parentheses for better readability. The unit $\mathbb{I}$ behaves as unit for all three connectives. We define the *relation* $\equiv$ *on formulas* to be the smallest congruence generated by associativity of $\otimes, \parr, \triangleleft$, commutativity of $\otimes, \parr$, and the unit equations:

$$\begin{array}{lll} A \otimes (B \otimes C) \equiv (A \otimes B) \otimes C & A \otimes B \equiv B \otimes A & \mathbb{I} \otimes A \equiv A \\ A \parr [B \parr C] \equiv [A \parr B] \parr C & A \parr B \equiv B \parr A & \mathbb{I} \parr A \equiv A \\ A \triangleleft \langle B \triangleleft C \rangle \equiv \langle A \triangleleft B \rangle \triangleleft C & & \mathbb{I} \triangleleft A \equiv A \equiv A \triangleleft \mathbb{I} \end{array} \tag{2}$$

[3] Note that this is redundant and carries no additional meaning. The only purpose is better readability.

The involutive *(linear) negation* $(-)^\perp$ is extended from propositional variables to general formulas by taking De Morgan's laws as its inductive definition, i.e., we define $(a^\perp)^\perp = a$ for all propositional variables a, and

$$\mathbb{I}^\perp = \mathbb{I} \qquad (A \otimes B)^\perp = A^\perp \parr B^\perp \qquad [A \parr B]^\perp = A^\perp \otimes B^\perp \qquad \langle A \triangleleft B\rangle^\perp = A^\perp \triangleleft B^\perp$$

The last equality is what we mean when we say that seq is *self-dual*. Note that the right-hand side is indeed $A^\perp \triangleleft B^\perp$ and not $B^\perp \triangleleft A^\perp$.[4]

We will also need the notion of **sequent**, which has to be generalized from multisets of formulas to series-parallel orders of formulas.[5] We denote a *sequent* in pomset logic by capital Greek letters $\Gamma, \Delta, \ldots$ and they are generated as follows: $\Gamma, \Delta ::= \varnothing \mid A \mid [\Gamma, \Delta] \mid \langle \Gamma; \Delta\rangle$, where $\varnothing$ stands for the empty sequent. We consider sequents equal modulo commutativity of $[\cdot,\cdot]$ and associativity of $[\cdot,\cdot]$ and $\langle\cdot;\cdot\rangle$, and the unit-laws for the empty sequent. In the remainder of this paper we will always omit redundant brackets.

The operations $[\cdot,\cdot]$ and $\langle\cdot;\cdot\rangle$ serve as counterparts on sequents to the connectives $\parr$ and $\triangleleft$ on formulas (just as the sequent $\vdash A, B, C$ morally means $A \parr B \parr C$ in linear logic).

▶ Remark 2.1. Pomset logic is not the only system that features "non-flat" sequents with two distinct connectives. Another famous example is the logic BI of bunched implications [19].

In [21], Retoré presents proof nets for pomset logic as *RB-digraphs*, that is, *directed* graphs equipped with perfect matchings, extending his reformulation of MLL_0 proof nets as *undirected* RB-graphs [23]. We recall these notions below.

▶ **Definition 2.2.** *A* ***digraph*** $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ *consists of a finite set of* ***vertices*** $V_\mathcal{G}$ *and a set of* ***edges*** $E_\mathcal{G} \subseteq V_\mathcal{G}^2 \setminus \{(u,u) \mid u \in V_\mathcal{G}\}$. *A digraph* $\mathcal{G}$ *is* ***labeled*** *if there is a map* $\ell\colon V_\mathcal{G} \to \mathcal{L}$ *assigning each vertex* v *of* $V_\mathcal{G}$ *a* ***label*** $\ell(v) \in \mathcal{L}$ *in the* ***label set*** $\mathcal{L}$. *If* $\mathcal{L}$ *is the set* $\mathcal{V} \cup \mathcal{V}^\perp$ *of atoms, we speak of an* ***atom-labeled*** *digraph.*

In the remainder of this paper, all digraphs are atom-labelled, and for two digraphs $\mathcal{G}$ and $\mathcal{H}$, we write $\mathcal{G} = \mathcal{H}$ iff there is a label-preserving isomophism between them. Also, we often write $uv \in E_\mathcal{G}$ for $(u,v) \in E_\mathcal{G}$, and for a digraph $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$, we define the sets $E_\mathcal{G}^\otimes = \{(u,v) \mid (u,v) \in E_\mathcal{G} \text{ and } (v,u) \in E_\mathcal{G}\}$ and $E_\mathcal{G}^\triangleleft = \{(u,v) \mid (u,v) \in E_\mathcal{G} \text{ and } (v,u) \notin E_\mathcal{G}\}$, allowing us to treat $(V_\mathcal{G}, E_\mathcal{G}^\otimes)$ as undirected graph.

▶ **Definition 2.3.** *Let* $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ *and* $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$ *be disjoint digraphs. We can define the following operations:*

$$\begin{aligned}
\mathcal{G} \parr \mathcal{H} &= (V_\mathcal{G} \cup V_\mathcal{H}, E_\mathcal{G} \cup E_\mathcal{H}) \\
\mathcal{G} \triangleleft \mathcal{H} &= (V_\mathcal{G} \cup V_\mathcal{H}, E_\mathcal{G} \cup E_\mathcal{H} \cup \{(u,v) \mid u \in V_\mathcal{G} \text{ and } v \in V_\mathcal{H}\}) \\
\mathcal{G} \otimes \mathcal{H} &= (V_\mathcal{G} \cup V_\mathcal{H}, E_\mathcal{G} \cup E_\mathcal{H} \cup \{(u,v),(v,u) \mid u \in V_\mathcal{G} \text{ and } v \in V_\mathcal{H}\})
\end{aligned}$$

This allows us to define a mapping $[\![\cdot]\!]$ from formulas to digraphs as follows:

$$[\![\mathbb{I}]\!] = \varnothing \qquad [\![a]\!] = \bullet_a \qquad [\![a^\perp]\!] = \bullet_{a^\perp}$$

$$[\![A \parr B]\!] = [\![A]\!] \parr [\![B]\!] \qquad [\![A \triangleleft B]\!] = [\![A]\!] \triangleleft [\![B]\!] \qquad [\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!]$$

where $\varnothing$ is the empty graph, and $\bullet_a$ (respectively $\bullet_{a^\perp}$) is a single vertex graph whose vertex is labeled by a (respectively $a^\perp$).

[4] In that respect, pomset logic and BV are different from other non-commutative variants of linear logic where $\otimes$ and $\parr$ are non-commutative with $(A \otimes B)^\perp = B^\perp \parr A^\perp$ [29, 1].

[5] We follow here mainly the presentation of [24].

▶ **Proposition 2.4** ([22]). *For all formulas A and B, we have $[\![A]\!] = [\![B]\!]$ iff $A \equiv B$.*

This can be shown by a straightforward induction on the formulas. An immediate consequence of this proposition is that the extension of the mapping $[\![\cdot]\!]$ to sequents is well-defined, i.e., we have $[\![\Gamma, \Delta]\!] = [\![\Gamma]\!] \parr [\![\Delta]\!]$ and $[\![\Gamma; \Delta]\!] = [\![\Gamma]\!] \triangleleft [\![\Delta]\!]$.

▶ **Definition 2.5.** *Let $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ be a digraph and let $V_\mathcal{H} \subseteq V_\mathcal{G}$. The **subdigraph** of $\mathcal{G}$ **induced by** $V_\mathcal{H}$ is $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$, where $E_\mathcal{H} = \{(u, v) \mid (u, v) \in E_\mathcal{G}$ and $u \in V_\mathcal{H}$ and $v \in V_\mathcal{H}\}$. In this case we also say that $\mathcal{H}$ is an **induced subgraph** of $\mathcal{G}$ and denote that by $\mathcal{H} \sqsubseteq \mathcal{G}$. If additionally $V_\mathcal{H} \subset V_\mathcal{G}$ then we write $\mathcal{H} \sqsubset \mathcal{G}$.*

▶ **Definition 2.6.** *An undirected graph is **$\mathbf{P_4}$-free** if it does not contain a $\mathbf{P_4}$ (shown on the left below) as induced subgraph, and a directed graph is **$\mathbf{N}$-free** if it does not contain an $\mathbf{N}$ (shown on the right below) as induced subgraph.*

$\mathbf{P_4}$:  $\mathbf{N}$:  (3)

▶ **Definition 2.7.** *A **dicograph** is a digraph $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$, such that*

1. *the undirected graph $(V_\mathcal{G}, E_\mathcal{G}^\otimes)$ is $\mathbf{P_4}$-free,*
2. *the directed graph $(V_\mathcal{G}, E_\mathcal{G}^\triangleleft)$ is $\mathbf{N}$-free, and*
3. *the relation $E_\mathcal{G}$ is **weakly transitive**:*
   - *if $(u, v) \in E_\mathcal{G}^\triangleleft$ and $(v, w) \in E_\mathcal{G}$ then $(u, w) \in E_\mathcal{G}$, and*
   - *if $(u, v) \in E_\mathcal{G}$ and $(v, w) \in E_\mathcal{G}^\triangleleft$ then $(u, w) \in E_\mathcal{G}$.*

▶ **Proposition 2.8** ([4]). *$\mathcal{G}$ is a dicograph iff there is a formula A with $\mathcal{G} = [\![A]\!]$.*

▶ **Proposition 2.9.** *Let $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ be a dicograph. Then any induced subdigraph of $\mathcal{G}$ is also a dicograph.*

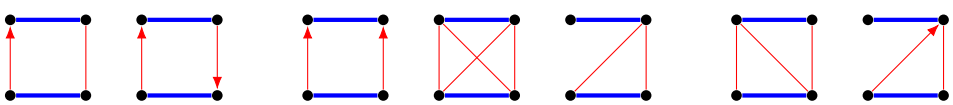
▶ **Definition 2.10.** *Let $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ be a digraph. A **perfect matching** B of $\mathcal{G}$ is a subset of edges such that:*

1. *any vertex has exactly one outgoing edge in B and exactly one incoming edge in B, i.e., for every $u \in V_\mathcal{G}$ there is exactly one pair $(v, w) \in V_\mathcal{G} \times V_\mathcal{G}$ such that $uv \in B$ and $wu \in B$, and*
2. *for all $u, v \in V_\mathcal{G}$, we have that $uv \in B$ iff $vu \in B$.*

*Item 2 means that B consists of bidirectional edges. In particular, this means that $v = w$ in Item 1. An **RB-digraph** $\mathcal{G} = (V_\mathcal{G}, R_\mathcal{G}, B_\mathcal{G})$ is a triple where $(V_\mathcal{G}, R_\mathcal{G} \uplus B_\mathcal{G})$ is a digraph and $B_\mathcal{G}$ is a perfect matching in it. Finally, an RB-digraph $\mathcal{G} = (V_\mathcal{G}, R_\mathcal{G}, B_\mathcal{G})$ is an **RB-dicograph** iff $(V_\mathcal{G}, R_\mathcal{G})$ is a dicograph.*[6]

In all figures representing RB-digraphs, we will (following [22]) draw the edges belonging to the matching (the set B) **bold and blue**, and the other edges (the set R) regular and red.

▶ **Example 2.11.** Below we show 7 examples of RB-digraphs. The first 5 are RB-dicographs, the last 2 are not.

 (4)

[6] Note that the perfect matching $B_\mathcal{G}$ is not part of the dicograph. In particular, we allow that two vertices in $V_\mathcal{G}$ can be connected by an edge in $R_\mathcal{G}$ and in $B_\mathcal{G}$.

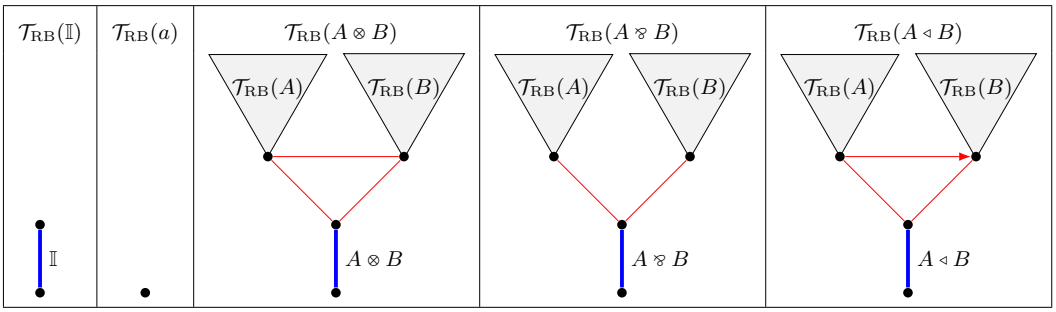


Figure 1 Inductive definition of RB-trees (which are not quite trees in the sense of graph theory, though they resemble the syntax trees of formulas). The root vertex is at the bottom.

▶ **Definition 2.12.** *An **elementary cycle** of length n in a digraph $(V_{\mathcal{G}}, E_{\mathcal{G}})$ is a $\mathbb{Z}/n\mathbb{Z}$-indexed sequence of vertices $u_0, \dots, u_{n-1} \in V_{\mathcal{G}}$ without repetitions such that for all $i \in \mathbb{Z}/n\mathbb{Z}$, $u_i u_{i+1} \in E_{\mathcal{G}}$. An **alternating elementary cycle** (or **æ-cycle**) in an RB-digraph $(V_{\mathcal{G}}, R_{\mathcal{G}}, B_{\mathcal{G}})$ is an elementary cycle $u_0, \dots, u_{n-1}$ in $(V_{\mathcal{G}}, R_{\mathcal{G}} \uplus B_{\mathcal{G}})$, such that for all $i \in \mathbb{Z}/n\mathbb{Z}$, exactly one of $u_{i-1}u_i$ and $u_i u_{i+1}$ is in $B_{\mathcal{G}}$ (so that the other one is in $R_{\mathcal{G}}$). Note that this forces the length n of an æ-cycle to be even. A **chord** in an æ-cycle is an edge $vw \in R_{\mathcal{G}}$ such that $v, w \in \{u_0, \dots, u_{n-1}\}$ but neither vw nor wv are in the æ-cycle. We say an æ-cycle is **chordless** if it does not admit any chord in $\mathcal{G}$. We say that an RB-digraph $(V_{\mathcal{G}}, R_{\mathcal{G}}, B_{\mathcal{G}})$ is an æ-cycle (resp. chordless æ-cycle) if all vertices of $V_{\mathcal{G}}$ participate in the cycle. Finally, an RB-digraph is **æ-acyclic** if it does not contain a chordless æ-cycle as induced subgraph.*

▶ **Example 2.13.** To continue Example 2.11, the first two RB-digraphs in (4) are chordless æ-cycles. The other five are æ-acyclic.

We are now ready to define pomset logic proof nets, which are in fact æ-acyclic RB-dicographs.

A **pomset logic pre-proof** of a sequent Γ is an involution ℓ on its set of atom occurrences such that an atom is always mapped to its dual. This involutive mapping on the atom occurrences is called the **axiom linking**.

In order to define which pre-proofs are proofs, Retoré [21, 22] gave two equivalent correctness criteria, which are in fact two ways of translating the sequent Γ and the linking ℓ into an RB-dicograph.

Let us call the first the **relational RB-prenet**, denoted by $\rho(\Gamma, \ell)$, which is the RB-dicograph $\mathcal{G} = (V_{\mathcal{G}}, R_{\mathcal{G}}, B_{\mathcal{G}})$ where $(V_{\mathcal{G}}, R_{\mathcal{G}}) = [\![\Gamma]\!]$, and we have $xy \in B_{\mathcal{G}}$ iff the atoms in Γ that correspond to x and y are mapped to each other by the axiom linking ℓ.

▶ **Example 2.14.** The first five RB-graphs in (4) are in fact relational RB-prenets for the formulas $\langle a^\perp \triangleleft b^\perp \rangle \parr (a \otimes b)$, $\langle a^\perp \triangleleft b^\perp \rangle \parr \langle b \triangleleft a \rangle$, and $\langle a^\perp \triangleleft b^\perp \rangle \parr \langle a \triangleleft b \rangle$, $[a \parr a^\perp] \otimes [b \parr b^\perp]$, $a \parr (a^\perp \otimes [b \parr b^\perp])$, respectively (with the obvious unique linking).

The second way of translating a sequent Γ and a linking ℓ into an RB-dicograph is based on the formula tree structure. We define inductively for each formula C in Γ its **RB-tree**, denoted as $\mathcal{T}_{\mathrm{RB}}(C)$, as shown in Figure 1.[7]

[7] Technically speaking, this not a tree in the graph-theoretical sense, but we use the name as it carries the structure of the formula tree.

$$\mathsf{ai}{\downarrow}\,\frac{\mathbb{I}}{a \parr a^\perp} \qquad \mathsf{s}\,\frac{[A \parr C] \otimes B}{(A \otimes B) \parr C} \qquad \mathsf{q}{\downarrow}\,\frac{[A \parr C] \triangleleft [B \parr D]}{\langle A \triangleleft B\rangle \parr \langle C \triangleleft D\rangle} \qquad \equiv \frac{A}{B}\ \text{(provided } A{\equiv}B\text{)}$$

Figure 2 System BV.

If we have a sequent Γ, then $\mathcal{T}_{\mathrm{RB}}(\Gamma)$ is obtained from the RB-trees of the formulas in Γ which are connected at the roots via the edges corresponding to the series-parallel order of the sequent structure. In order to obtain an RB-digraph, we need to add the B-edges corresponding to the linking ℓ. We denote this RB-digraph, which is in fact an RB-dicograph, by $\tau(\Gamma, \ell)$ and call it the **tree-like RB-prenet** of Γ and ℓ.

▶ **Definition 2.15.** *A relational RB-prenet (resp. tree-like RB-prenet) is* ***correct*** *if it does not contain any chordless æ-cycle. A correct relational RB-prenet (resp. correct tree-like RB-prenet) is also called a* ***relational RB-net*** *(resp.* ***tree-like RB-net****). In both cases we also speak of* ***(pomset logic) proof nets****. A sequent Γ is provable in pomset logic of there is a linking ℓ, such that $\rho(\Gamma, \ell)$ or $\tau(\Gamma, \ell)$ is a proof net.*

The above definition makes sense because of the following theorem by Retoré:

▶ **Theorem 2.16** ([22, Theorem 7])**.** *For every sequent Γ and linking ℓ, we have that $\rho(\Gamma, \ell)$ is correct if and only if $\tau(\Gamma, \ell)$ is correct.*

▶ **Example 2.17.** The three RB-graphs in the middle of (4) are pomset logic proof nets.

3 Preliminaries on System BV

In [10] Guglielmi introduces **system** BV, which is a deductive system for formulas defined in (1). It is defined in the formalism called the *calculus of structures*, and it works similar to a rewriting system, modulo the equational theory defined in (2).

The inference rules of **system** BV are shown in Figure 2. These rules have to be read as rewriting rule schemes, meaning that (i) the variable a can be substituted by any atom, and the variables A, B, C, D can be substituted by any formula, and that (ii) the rules can be applied inside any (positive) context.

A **(proof) system** is a set of inference rules. We write $\mathsf{S} \Vdash \frac{A}{B}\,\delta$, or more concisely $A \vdash^{\delta}_{\mathsf{S}} B$, if there is a derivation from A to B using only rules from the system S, and that derivation is named δ. If in that situation $A = \mathbb{I}$, then we write it as $\mathsf{S} \Vdash \frac{}{B}\,\delta$ or simply as $\vdash^{\delta}_{\mathsf{S}} B$ and call δ a **proof** of B. In this case we say that B is **provable** S.

▶ **Example 3.1.** Here are three proofs in BV, corresponding to the three proof nets in the middle of (4):

$$\mathsf{q}{\downarrow}\,\frac{\mathsf{ai}{\downarrow}\,\frac{}{a^\perp \parr a} \quad \mathsf{ai}{\downarrow}\,\frac{\equiv\,\frac{}{\mathbb{I} \triangleleft [b^\perp \parr b]}}{[a^\perp \parr a] \triangleleft [b^\perp \parr b]}}{\langle a^\perp \triangleleft b^\perp\rangle \parr \langle a \triangleleft b\rangle} \qquad \mathsf{ai}{\downarrow}\,\frac{\equiv\,\frac{}{\mathbb{I} \otimes \mathbb{I}} \quad \mathsf{ai}{\downarrow}\,\frac{}{\mathbb{I} \otimes [b \parr b^\perp]}}{[a \parr a^\perp] \otimes [b \parr b^\perp]} \qquad \mathsf{ai}{\downarrow}\,\frac{\equiv\,\frac{\mathsf{ai}{\downarrow}\,\frac{}{a \parr a^\perp}}{a \parr (a^\perp \otimes \mathbb{I})} \quad \mathsf{ai}{\downarrow}\,\frac{}{a \parr (a^\perp \otimes [b \parr b^\perp])}}{} \tag{5}$$

$$\mathsf{ai}^{\circ}{\downarrow}\,\frac{}{a \parr a^{\perp}} \qquad \mathsf{ai}^{\otimes}{\downarrow}\,\frac{B}{[a \parr a^{\perp}] \otimes B} \qquad \equiv'\,\frac{A}{B}\ (\text{provided } A \equiv' B)$$

$$\mathsf{ai}^{\triangleleft}_{\mathsf{L}}{\downarrow}\,\frac{B}{[a \parr a^{\perp}] \triangleleft B} \qquad \mathsf{ai}^{\triangleleft}_{\mathsf{R}}{\downarrow}\,\frac{B}{B \triangleleft [a \parr a^{\perp}]}$$

$$\mathsf{q}^{\mathsf{L}}_{3}{\downarrow}\,\frac{[A \parr C] \triangleleft B}{\langle A \triangleleft B\rangle \parr C} \qquad \mathsf{q}^{\mathsf{R}}_{3}{\downarrow}\,\frac{A \triangleleft [B \parr C]}{\langle A \triangleleft B\rangle \parr C} \qquad \mathsf{s}_3\,\frac{[A \parr C] \otimes B}{(A \otimes B) \parr C}$$

$$\mathsf{q}_4{\downarrow}\,\frac{[A \parr C] \triangleleft [B \parr D]}{\langle A \triangleleft B\rangle \parr \langle C \triangleleft D\rangle} \qquad \mathsf{q}_2{\downarrow}\,\frac{A \triangleleft B}{A \parr B} \qquad \mathsf{s}_2\,\frac{A \otimes B}{A \parr B}$$

Figure 3 System BVu.

An inference rule r is **derivable** in a system S iff for every instance $\mathsf{r}\,\frac{A}{B}$ there is a derivation $A \vdash_{\mathsf{S}} B$. An inference rule r is **admissible** for a system S iff for every proof $\vdash_{\mathsf{S}\cup\{\mathsf{r}\}} A$ there is a proof $\vdash_{\mathsf{S}} B$.

▶ **Definition 3.2.** *Two system* S_1 *and* S_2 *are* ***equivalent*** *if they prove the same formulas.*

To simplify the proofs of our main results, we need a unit-free version of BV. We use here a variant of the one proposed by Kahramanoğulları in [14] in order to reduce the non-determinism in proof search in BV.

The system is called BVu, and its formulas are the same as defined in (1), except that we do not allow any occurrence of the unit $\mathbb{I}$. This means that we have to restrict the equivalence $\equiv$ defined in (2) to the unit-free formulas. We define the relation $\equiv'$ to be the smallest congruence generated by

$$\begin{array}{lcllcl} A \otimes (B \otimes C) & \equiv' & (A \otimes B) \otimes C & \qquad A \otimes B & \equiv' & B \otimes A \\ A \parr [B \parr C] & \equiv' & [A \parr B] \parr C & \qquad A \parr B & \equiv' & B \parr A \\ A \triangleleft \langle B \triangleleft C\rangle & \equiv' & \langle A \triangleleft B\rangle \triangleleft C & & & \end{array} \tag{6}$$

The inference rules for BVu are then shown in Figure 3.[8] Note that the rule $\mathsf{ai}^{\circ}{\downarrow}$ has no premise. It is an axiom that is used exactly once in a **proof** which is a derivation without premise (as the unit $\mathbb{I}$ is not present and cannot take this role).

▶ **Proposition 3.3** ([14])**.** *The systems* BVu *and* BV *are equivalent.*

Proof. First, if we have a proof $\vdash_{\mathsf{BVu}} A$ then we can simply replace the top instance of $\mathsf{ai}^{\circ}{\downarrow}$ by $\mathsf{ai}{\downarrow}$ and have a proof of BV. Conversely, assume we have a proof $\vdash^{\delta}_{\mathsf{BV}} B$. Then, in δ, the unit $\mathbb{I}$ can occur. Let δ' be obtained from δ by deleting the unit $\mathbb{I}$ everywhere (which means that the topmost $\mathsf{ai}{\downarrow}$ is replaced by $\mathsf{ai}^{\circ}{\downarrow}$). Then every instance of the rule $\equiv$ becomes an instance of $\equiv'$; every instance of $\mathsf{q}{\downarrow}$ becomes an instance of $\mathsf{q}_2{\downarrow}$ or $\mathsf{q}^{\mathsf{L}}_3{\downarrow}$ or $\mathsf{q}^{\mathsf{R}}_3{\downarrow}$ or $\mathsf{q}_4{\downarrow}$ or trivial (i.e., premise and conclusion of the rule instance become equal); and similarly for s. However, an instance of $\mathsf{ai}{\downarrow}$ can become an instance of $\mathsf{ai}^{\otimes}{\downarrow}$ or $\mathsf{ai}^{\triangleleft}_{\mathsf{L}}{\downarrow}$ or $\mathsf{ai}^{\triangleleft}_{\mathsf{R}}{\downarrow}$ (which are in BVu), or $\mathsf{ai}^{\parr}{\downarrow}$ which is shown below.

$$\mathsf{ai}^{\parr}{\downarrow}\,\frac{B}{a \parr a^{\perp} \parr B} \tag{7}$$

This rule is not in BVu, but can be derived with $\{\mathsf{ai}^{\otimes}{\downarrow}, \mathsf{s}_2\}$. ◀

[8] The rules in the bottom two rows of Figure 3 have have already been studied by Retoré in [22], as part of a rewrite system on digraphs to generate theorems of pomset logic.

▶ Remark 3.4. Our version of BVu is slightly different from the one by Kahramanoğulları [14]. In [14] the rule s_2 is absent, and instead the rule $\mathsf{ai}^{\parr}{\downarrow}$ shown in (7) is part of the system. It is easy to see that the two variants of BVu are equivalent: first, as we have mentioned above, the rule $\mathsf{ai}^{\parr}{\downarrow}$ is derivable in $\{\mathsf{ai}^{\otimes}{\downarrow}, \mathsf{s}_2\}$, and second, the rule s_2 is admissible if $\mathsf{ai}^{\parr}{\downarrow}$ is present. This can be seen by an easy induction on the size of the derivation. However, note that the same trick does not work for the rule $\mathsf{q}_2{\downarrow}$. This rule cannot be shown admissible, as the formula $\langle a \triangleleft [b \parr c]\rangle \parr \langle [a^{\perp} \parr b^{\perp}] \triangleleft c^{\perp}\rangle$ is not provable in BVu without $\mathsf{q}_2{\downarrow}$.

We will also need a variant of BVu that we call BV$\hat{\mathsf{u}}$ and that is obtained from BVu by restricting rules $\mathsf{q}_2{\downarrow}$ and s_2 to cases where neither A nor B has a $\parr$ as main connective, i.e., we replace $\mathsf{q}_2{\downarrow}$ and s_2 by $\hat{\mathsf{q}}_2{\downarrow}$ and $\hat{\mathsf{s}}_2$, respectively:

$$\hat{\mathsf{q}}_2{\downarrow}\,\frac{A \triangleleft B}{A \parr B} \qquad \hat{\mathsf{s}}_2\,\frac{A \otimes B}{A \parr B} \qquad \text{where } A \not\equiv' C \parr D \text{ and } B \not\equiv' C \parr D \text{ for any formulas } C \text{ and } D. \tag{8}$$

and similarly, by restricting the rules $\mathsf{q}_3^{\mathsf{L}}{\downarrow}$, $\mathsf{q}_3^{\mathsf{R}}{\downarrow}$, and s_3 to cases where C does not have a $\parr$ as main connective, i.e., these three rules are replaced by $\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}$, $\hat{\mathsf{q}}_3^{\mathsf{R}}{\downarrow}$, and $\hat{\mathsf{s}}_3$, respectively:

$$\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}\,\frac{[A \parr C] \triangleleft B}{\langle A \triangleleft B\rangle \parr C} \qquad \hat{\mathsf{q}}_3^{\mathsf{R}}{\downarrow}\,\frac{A \triangleleft [B \parr C]}{\langle A \triangleleft B\rangle \parr C} \qquad \hat{\mathsf{s}}_3\,\frac{[A \parr C] \otimes B}{(A \otimes B) \parr C} \qquad \text{where } C \not\equiv' D \parr E \text{ for any formulas } D \text{ and } E. \tag{9}$$

▶ **Proposition 3.5.** *The systems* BVu *and* BV$\hat{\mathsf{u}}$ *are equivalent.*

Proof. Any derivation in BV$\hat{\mathsf{u}}$ is also a derivation in BVu. Conversely, the rules $\mathsf{q}_2{\downarrow}$ and s_2 and s_3 are derivable with $\{\hat{\mathsf{q}}_2{\downarrow}, \hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}, \hat{\mathsf{q}}_3^{\mathsf{R}}{\downarrow}, \equiv'\}$ and $\{\hat{\mathsf{s}}_2, \hat{\mathsf{s}}_3, \equiv'\}$ and $\{\hat{\mathsf{s}}_3, \equiv'\}$, respectively, as shown below:

$$\dfrac{\dfrac{\dfrac{\dfrac{[A' \parr A''] \triangleleft [B' \parr B'']}{\hat{\mathsf{q}}_3^{\mathsf{R}}{\downarrow}}\,\langle [A' \parr A''] \triangleleft B'\rangle \parr B''}{\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}}\,\langle A' \triangleleft B'\rangle \parr A'' \parr B''}{\hat{\mathsf{q}}_2{\downarrow}}\,A' \parr B' \parr A'' \parr B''}{\equiv'}\,[A' \parr A''] \parr [B' \parr B''] \qquad \dfrac{\dfrac{\dfrac{[A' \parr A''] \otimes [B' \parr B'']}{\hat{\mathsf{s}}_3}\,([A' \parr A''] \otimes B') \parr B''}{\hat{\mathsf{s}}_2}\,(A' \otimes B') \parr A'' \parr B''}{\equiv'}\,A' \parr B' \parr A'' \parr B''}{\equiv',\hat{\mathsf{s}}_3,\equiv'}\,[A' \parr A''] \parr [B' \parr B''] \qquad \dfrac{\dfrac{\dfrac{[A \parr [C' \parr C'']] \otimes B}{\hat{\mathsf{s}}_3}\,[[A \parr C'] \parr C''] \otimes B}{\hat{\mathsf{s}}_3}\,([A \parr C'] \otimes B) \parr C''}{\equiv'}\,[(A \otimes B) \parr C'] \parr C''}{\equiv'}\,(A \otimes B) \parr [C' \parr C'']$$

and similarly, the rules $\mathsf{q}_3^{\mathsf{L}}{\downarrow}$ and $\mathsf{q}_3^{\mathsf{R}}{\downarrow}$ are derivable in $\{\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}, \equiv'\}$ and $\{\hat{\mathsf{q}}_3^{\mathsf{R}}{\downarrow}, \equiv'\}$, respectively. ◀

4 BV is Contained in Pomset Logic

In this section we do not only show that every theorem of BV is also a theorem of pomset logic, but also that every proof in BV uniquely determines a pomset logic proof net with the same conclusion.

We have already seen in Section 2 that every formula uniquely determines a dicograph. Furthermore, by inspecting the rules of BV in Figure 2, one can see that the rule $\equiv$ does not change that dicograph, and that the rules s and $\mathsf{q}{\downarrow}$ only change the set of edges but not the set of vertices of the corresponding dicograph. Additionally, every instance of $\mathsf{ai}{\downarrow}$ removes one pair of dual atoms, and in a proof of BV, every atom occurring in the conclusion has to be removed by exactly one instance of $\mathsf{ai}{\downarrow}$ in the proof.

This means that every BV proof δ uniquely determines an axiom linking $\ell(\delta)$ for its conclusion, and hence, by definition a pomset logic pre-proof and also a relational RB-prenet.

We are now going to show that every relational RB-prenet that is obtained from a BV proof in such a way is indeed correct, and therefore every theorem of BV is also a theorem of pomset logic. The proof of the main lemma is based on the construction from [28], but the complete proof has never been published.

To begin, let δ be a BV proof of a formula A. We denote by $(\!|\delta|\!) = \rho(A, \ell(\delta))$ the relational RB-prenet generated from δ as described in Section 2. Then the main result of this section is the following.

▶ **Theorem 4.1.** *For every* BV *proof* δ*, the relational RB-prenet* $(\!|\delta|\!)$ *is correct.*

▶ **Example 4.2.** The three correct relational RB-prenets in the middle of (4) are obtained from the three BV-proofs in Example 3.1.

In order to prove Theorem 4.1, we first introduce an additional definition.

▶ **Definition 4.3.** *A formula is* balanced *if every propositional variable that occurs in* A *occurs exactly once positive and exactly once negative. A balanced formula* A *uniquely determines an axiom linking on* A*, that we denote by* $\ell(A)$*. Then we write* $(\!|A|\!)$ *for the relational RB-prenet* $\rho(A, \ell(A))$*, i.e.,* $(\!|A|\!) = (V_{\mathcal{A}}, R_{\mathcal{A}}, B_{\mathcal{A}})$*, where* $(V_{\mathcal{A}}, R_{\mathcal{A}}) = [\![A]\!]$ *and* $B_{\mathcal{A}}$ *is the matching associated to* $\ell(A)$*.*

Conversely, every RB-dicograph uniquely determines a balanced formula, up to renaming of variables and equivalence under $\equiv$. This gives us immediately the following proposition.

▶ **Proposition 4.4.** *Let* δ *be a proof in* BV*. Then there is a balanced formula* A*, that is provable in* BV *and such that* $(\!|A|\!)$ *and* $(\!|\delta|\!)$ *are isomorphic.*

Proof. Let B be the conclusion of δ. Then A is obtained from B by renaming all variable occurrences such that the result is balanced and the linking is preserved. ◀

▶ **Definition 4.5.** *Let* A *be a formula. A formula* B *is a* ***pseudo-subformula*** *of* A*, written as* $B \sqsubseteq A$*, if it is equivalent under* $\equiv$ *to some* A' *that can be obtained from* A *by replacing some atom occurrences in* A *by* $\mathbb{I}$*. If* $B \sqsubseteq A$ *and* $B \not\equiv A$*, then we say that* B *is a* ***proper pseudo-subformula*** *of* A*, and write it as* $B \sqsubset A$*.*

▶ **Example 4.6.** We have that $\langle (a \otimes b) \triangleleft d \triangleleft e \rangle \parr (b \otimes [(e \otimes f) \parr \langle a \triangleleft b \rangle])$ has $\langle a \triangleleft d \rangle \parr (b \otimes b)$ as pseudo-subformula which is equivalent to $\langle (a \otimes \mathbb{I}) \triangleleft d \triangleleft \mathbb{I} \rangle \parr (b \otimes [(\mathbb{I} \otimes \mathbb{I}) \parr \langle \mathbb{I} \triangleleft b \rangle])$.

The following proposition explains our choice to denote both pseudo-subformulas and induced subgraphs (Definition 2.5) by $\sqsubseteq$.

▶ **Proposition 4.7.** *We have* $B \sqsubseteq A$ *iff* $[\![B]\!] \sqsubseteq [\![A]\!]$ *and* $B \sqsubset A$ *iff* $[\![B]\!] \sqsubset [\![A]\!]$.

Proof. This follows directly from the definitions of $[\![\cdot]\!]$ and $\sqsubseteq$ and Proposition 2.4. ◀

▶ **Lemma 4.8.** *Let* A *be a balanced formula and* B *be a balanced pseudo-subformula of* A*. If* A *is provable in* BV*, then so is* B*.*

Proof. Let δ be the proof of A in BV, and let δ' be obtained by replacing all atoms that do not occur in B in every line of δ by $\mathbb{I}$. Then δ' is a valid derivation of B in BV. ◀

▶ **Definition 4.9.** *A* ***balanced cycle*** *is a balanced formula* H *such that* $(\!|H|\!)$ *is an æ-cycle.*

▶ **Proposition 4.10.** *A formula* H *is a balanced cycle if and only if there are pairwise distinct atoms* $a_1, \ldots, a_n$ *for some* $n \geq 1$*, such that* $H \equiv L_1 \parr L_2 \parr \cdots \parr L_n$*, where* $L_1 = a_n^{\perp} \otimes a_1$ *or* $L_1 = a_n^{\perp} \triangleleft a_1$*, and for every* $i \in \{2, \ldots, n\}$ *we have* $L_i = a_{i-1}^{\perp} \otimes a_i$ *or* $L_i = a_{i-1}^{\perp} \triangleleft a_i$*.*

Proof. This follows immediately from the definitions. ◀

▶ **Definition 4.11.** *We say that a balanced formula A* ***contains a cycle*** *if it has a pseudo-subformula $B \sqsubseteq A$ that is a balanced cycle (or, equivalently if $(\!|A|\!)$ contains a chordless æ-cycle).*

We are now ready to state and prove the central lemma to this section.

▶ **Lemma 4.12.** *Let $\mathsf{r}\dfrac{Q}{P}$ be an instance of an inference rule in* BV$\hat{\mathsf{u}}$*. If P is a balanced cycle then Q contains a cycle. If $r \neq \equiv'$ then the size of the cycle in Q is strictly smaller than $|P|$.*

Proof. By Proposition 4.10 we have that $P \equiv' L_1 \parr L_2 \parr \ldots \parr L_n$, where $L_1 = a_n^\perp \otimes a_1$ or $L_1 = a_n^\perp \triangleleft a_1$, and for every $i \in \{2, \ldots, n\}$ we have $L_i = a_{i-1}^\perp \otimes a_i$ or $L_i = a_{i-1}^\perp \triangleleft a_i$, with all a_i being pairwise distinct. We proceed by case analysis on the rule r. First observe that by Proposition 4.10 the rules $\mathsf{ai}^{\otimes}{\downarrow}$, $\mathsf{ai}^{\triangleleft}_{\mathsf{L}}{\downarrow}$, $\mathsf{ai}^{\triangleleft}_{\mathsf{R}}{\downarrow}$ cannot be applied to P (seen bottom up), and if $\mathsf{r} = \equiv'$, then Q trivially contains a cycle, whose size is equal to $|P|$. Now assume r is

- $\mathsf{q}_4{\downarrow}\dfrac{[A \parr C] \triangleleft [B \parr D]}{\langle A \triangleleft B\rangle \parr \langle C \triangleleft D\rangle}$: Without loss of generality, assume that $A = a_n^\perp$ and $B = a_1$ and $C = a_{i-1}^\perp$ and $D = a_i$ for some $i \in \{2, \ldots, n\}$. Then

  $$Q \equiv' \langle [a_n^\perp \parr a_{i-1}^\perp] \triangleleft [a_1 \parr a_i]\rangle \parr L_2 \parr \cdots \parr L_{i-1} \parr L_{i+1} \parr \cdots \parr L_n$$

  which contains the cycle $\langle a_n^\perp \triangleleft a_i\rangle \parr L_{i+1} \parr \cdots \parr L_n$.
- $\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}\dfrac{[A \parr C] \triangleleft B}{\langle A \triangleleft B\rangle \parr C}$: Without loss of generality, we assume that $A = a_n^\perp$ and $B = a_1$ and $C = L_i$ for some $i \in \{2, \ldots, n\}$. Then

  $$Q \equiv' \langle [a_n^\perp \parr L_i] \triangleleft a_1\rangle \parr L_2 \parr \cdots \parr L_{i-1} \parr L_{i+1} \parr \cdots \parr L_n$$

  which contains the cycle $\langle a_{i-1}^\perp \triangleleft a_1\rangle \parr L_2 \parr \cdots \parr L_{i-1}$.
- $\hat{\mathsf{q}}_3^{\mathsf{R}}{\downarrow}\dfrac{A \triangleleft [B \parr C]}{\langle A \triangleleft B\rangle \parr C}$: As before, without loss of generality, we assume that $A = a_n^\perp$ and $B = a_1$ and $C = L_i$ for some $i \in \{2, \ldots, n\}$. Then

  $$Q \equiv' \langle a_n^\perp \triangleleft [a_1 \parr L_i]\rangle \parr L_2 \parr \cdots \parr L_{i-1} \parr L_{i+1} \parr \cdots \parr L_n$$

  which contains the cycle $\langle a_n^\perp \triangleleft a_i\rangle \parr L_{i+1} \parr \cdots \parr L_n$.
- $\hat{\mathsf{q}}_2{\downarrow}\dfrac{A \triangleleft B}{A \parr B}$: We can assume that $A = L_i$ and $B = L_j$ for some $i, j \in \{1, \ldots, n\}$. There are two subcases:
  - $i < j$: Then $Q = \langle L_i \triangleleft L_j\rangle \parr L_1 \parr \cdots \parr L_{i-1} \parr L_{i+1} \parr \cdots \parr L_{j-1} \parr L_{j+1} \parr \ldots \parr L_n$ which contains the cycle $L_1 \parr \cdots \parr L_{i-1} \parr \langle a_{i-1}^\perp \triangleleft a_j\rangle \parr L_{j+1} \parr \ldots \parr L_n$.
  - $j < i$: Then $Q = \langle L_i \triangleleft L_j\rangle \parr L_1 \parr \cdots \parr L_{j-1} \parr L_{j+1} \parr \cdots \parr L_{i-1} \parr L_{i+1} \parr \ldots \parr L_n$ which contains the cycle $\langle a_{i-1}^\perp \triangleleft a_j\rangle \parr L_{j+1} \parr \ldots \parr L_{i-1}$.
- $\hat{\mathsf{s}}_3\dfrac{[A \parr C] \otimes B}{(A \otimes B) \parr C}$: This case is analogous to the case $\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}$ above.
- $\hat{\mathsf{s}}_2\dfrac{A \otimes B}{A \parr B}$: This case is analogous to the case $\hat{\mathsf{q}}_2{\downarrow}$ above.

In all cases the size of the cycle in Q is strictly smaller than $|Q| = |P|$. ◀

▶ **Lemma 4.13.** *Let P be a balanced formula that contains a cycle. Then P is not provable in* BV*.*

Proof. Let H be the cycle in P, and let $n = |H|$ be its size. We proceed by induction on n. Note that n has to be even. For $n = 2$, we have that $H \equiv a^\perp \triangleleft a$ or $H \equiv a^\perp \otimes a$ for some atom a. By way of contradiction, assume P is provable in BV. By Lemma 4.8, H is also provable in BV, which is impossible. For the inductive case let now $n > 2$. As before, we have by Lemma 4.8 that H is provable in BV. By Proposition 3.3 and Proposition 3.5, H is provable in $\mathsf{BV\hat{u}}$. Let δ be that proof in $\mathsf{BV\hat{u}}$. Let now Q be the premise of the bottommost rule instance r of δ that is not a $\equiv'$ (i.e., the conclusion of r is $H' \equiv' H$ and $Q \not\equiv' H$). By Lemma 4.12, Q contains a cycle whose size is smaller than n. By induction hypothesis Q is not provable in BV, and therefore also not provable in $\mathsf{BV\hat{u}}$, which is a contradiction to the existence to δ. ◀

We can now complete the proof of Theorem 4.1.

Proof of Theorem 4.1. Let δ be a proof in BV. By Proposition 4.4, there is a balanced formula P, such that $(\!|P|\!)$ is isomorphic to $(\!|\delta|\!)$, and such that P is provable in BV. Now assume, by way of contradiction, that $(\!|\delta|\!)$ is incorrect. That means that $(\!|\delta|\!)$ contains a chordless æ-cycle, or equivalently, that P contains a cycle. By Lemma 4.13, P is not provable in BV. Contradiction. ◀

5 Pomset Logic is not Contained in BV

In this section we present a formula that is provable in pomset logic, i.e., has a correct pomset logic proof net, but that is not provable in BV. From what has been said in the previous section, it follows that if such a formula exists then there is also a balanced such formula. The formula we discuss in this section is the formula Q shown below:

$$Q = (\langle a \triangleleft b\rangle \otimes \langle c \triangleleft d\rangle) \parr (\langle e \triangleleft f\rangle \otimes \langle g \triangleleft h\rangle) \parr \langle a^\perp \triangleleft h^\perp\rangle \parr \langle e^\perp \triangleleft b^\perp\rangle \parr \langle g^\perp \triangleleft d^\perp\rangle \parr \langle c^\perp \triangleleft f^\perp\rangle \quad (10)$$

or equivalently, the sequent

$$\Gamma_Q = [\langle a \triangleleft b\rangle \otimes \langle c \triangleleft d\rangle, \langle e \triangleleft f\rangle \otimes \langle g \triangleleft h\rangle, a^\perp \triangleleft h^\perp, e^\perp \triangleleft b^\perp, g^\perp \triangleleft d^\perp, c^\perp \triangleleft f^\perp] \quad (11)$$

Since the formula Q (resp. the sequent Γ_Q) is balanced, there is a unique axiom linking and therefore a unique relational RB-prenet and a unique tree-like RB-prenet. In Figure 4, we show the tree-like RB prenet for Γ_Q, and on the left of Figure 5 we show the relational RB-prenet, which is the same for Q and Γ_Q.

To see that these are provable in pomset logic, we have to show that the RB-prenets do not contain chordless æ-cycles. For this we focus on the tree-like RB-prenet, because in tree-like RB-prenets all æ-paths (and therefore also all æ-cycles) are chordless. Hence, it suffices to show that there are no æ-cycles.

Observe that the B-edges corresponding to the roots of the formulas in Γ_Q cannot participate in an æ-cycle because they have no adjacent R-edge at the bottom. We can therefore remove each of these B-edges, together with the two adjacent R-edges at the top. The resulting graph is shown on the right of Figure 5.

Another simplification we can do without affecting the æ-cycles in the graph is replacing the two B-edges labeled $a \triangleleft b$ and $c \triangleleft d$, together with the connecting R-edge by a single B-edge, and similarly for the two B-edges $g \triangleleft h$ and $e \triangleleft f$. The result is shown on the left of Figure 6.

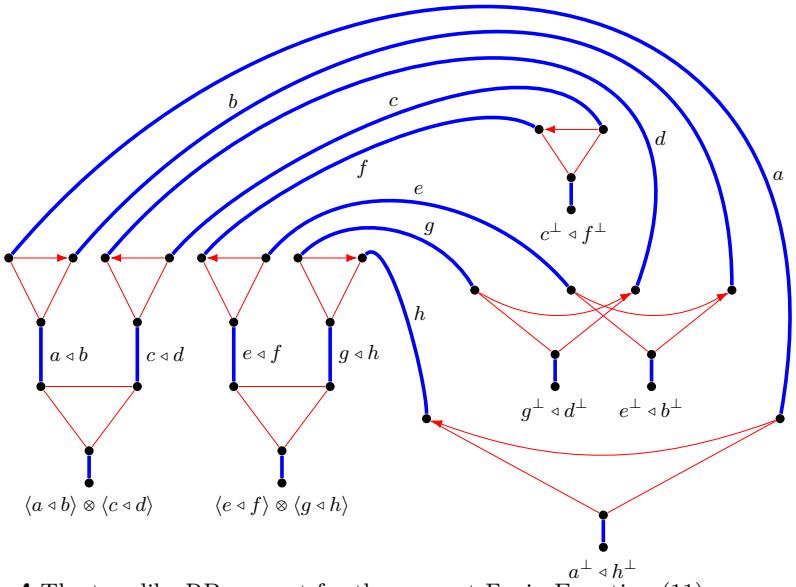


Figure 4 The tree-like RB-preenet for the sequent Γ_Q in Equation (11).

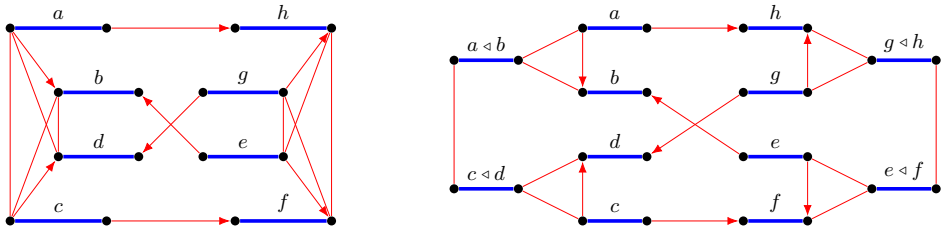


Figure 5 Left: The relational RB-prenet for Q in (10) and Γ_Q in (11).
Right: A simplification of the tree-like RB-prenet in Figure 4.

Finally, observe that there is no æ-cycle that passes trough the two B-edges labeled b and a. The reason is that the directed R-edge between them has the opposite direction of the two adjacent R-edges on the other endpoints of these B-edges. Thus, we can collapse these two edges (and the adjacent "triangle") to a single vertex. The same can be done for the pairs c/d and g/h and e/f. The result of this operation is shown on the right of Figure 6.

▶ **Proposition 5.1.** *The formula Q and the sequent Γ_Q shown in Equation* (10) *and Equation* (11) *above are provable in pomset logic.*

Proof. In the paragraphs above, we have argued that the tree-like RB-prenet in Figure 4 has an æ-cycle if and only if the RB-digraph on the right of Figure 6 has an æ-cycle. Now it is easy to see that this graph has no æ-cycle. Hence, tree-like RB-prenet for Γ_Q is correct. ◀

Let us now show that the formula Q is not provable in BV. To do so we will show that whenever a BV inference has as conclusion Q then its premise defines an incorrect RB-prenet in pomset logic, and is therefore not provable in pomset logic. Since by Theorem 4.1 all BV proofs induce correct pomset proof nets, we can conclude that those premises are not BV-provable, therefore there is no way to build a BV-proof of Q.

The main difficulty here is to make sure that we do not overlook any case when checking all possible inferences that have Q as conclusion. Since the unit $\mathbb{I}$ can make these kind of arguments difficult to check, we use here BVû. Now observe that Q has no subformula of the form $x \parr x^\perp$. This means we only have to consider the non-axiom rules of BVû.

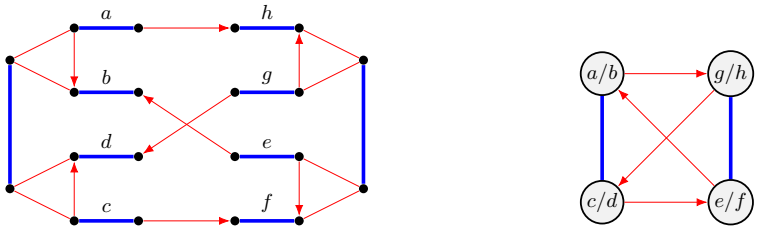


Figure 6 Two further simplifications of the graph on the right of Figure 5.

To cut down the number of cases to consider, we take advantage of the symmetries of Q. Let us first look at the *automorphisms*, i.e., permutations of the variables that results in a formula Q' with $Q' \equiv Q$, which means $(\!|Q'|\!) = (\!|Q|\!)$. The following are automorphisms:

(α) $a \leftrightarrow c$, $b \leftrightarrow d$, $e \leftrightarrow g$, $f \leftrightarrow h$

(β) $a \mapsto e$, $b \mapsto f$, $c \mapsto g$, $d \mapsto h$, $e \mapsto c$, $f \mapsto d$, $g \mapsto a$, $h \mapsto b$

The action of these automorphisms on the subformulas of Q of the form $x^\perp \triangleleft y^\perp$ is transitive: $\alpha(a^\perp \triangleleft h^\perp) = c^\perp \triangleleft f^\perp$, $\beta(a^\perp \triangleleft h^\perp) = e^\perp \triangleleft b^\perp$ and $\alpha \circ \beta(a^\perp \triangleleft h^\perp) = g^\perp \triangleleft d^\perp$.

Another useful symmetry is not quite an automorphism: it is the following *anti-automorphism*:

(γ) $a \leftrightarrow h$, $b \leftrightarrow g$, $c \leftrightarrow f$, $d \leftrightarrow e$

that sends Q to its "conjugate" $Q^\dagger$ defined inductively as follows:

$$x^\dagger = x \text{ when } x \text{ is an atom} \qquad (B \odot C)^\dagger = C^\dagger \odot B^\dagger \text{ for } \odot \in \{\parr, \otimes, \triangleleft\}$$

Note that the reversal of the arguments only matters for the non-commutative connective $\triangleleft$, and $(\!|Q^\dagger|\!)$ is the same as $(\!|Q|\!)$, except that all directed R-edges have the opposite direction. Thus, conjugacy preserves provability both in pomset logic (reversing the direction of all cycles in the correctness criterion) and in system $\mathsf{BV\hat{u}}$ (the inference rules are closed under conjugacy, with $\hat{\mathsf{q}}^{\mathsf{L}}_3{\downarrow}$ and $\hat{\mathsf{q}}^{\mathsf{R}}_3{\downarrow}$ being swapped).

We will now go through all the rules of $\mathsf{BV\hat{u}}$ and check all possible applications. Using a similar argument as in the proof of Lemma 4.12, we will see that in each case there is a cycle in the resulting premise.

- $\mathsf{q}_4{\downarrow}\dfrac{[A \parr C] \triangleleft [B \parr D]}{\langle A \triangleleft B\rangle \parr \langle C \triangleleft D\rangle}$: Because of the action of the automorphisms α/β, we can without loss of generality assume that $A = a^\perp$ and $B = h^\perp$. There are three subcases:
  - $C = e^\perp$ and $D = b^\perp$. We get the cycle $(e \otimes h) \parr \langle e^\perp \triangleleft h^\perp\rangle$ in the premise of the $\mathsf{q}_4{\downarrow}$-application.
  - $C = g^\perp$ and $D = d^\perp$. We get the cycle $(a \otimes d) \parr \langle a^\perp \triangleleft d^\perp\rangle$ in the premise of the $\mathsf{q}_4{\downarrow}$-application.
  - $C = c^\perp$ and $D = f^\perp$. We get the cycle $(b \otimes c) \parr (e \otimes h) \parr \langle c^\perp \triangleleft h^\perp\rangle \parr \langle e^\perp \triangleleft b^\perp\rangle$ in the premise of the $\mathsf{q}_4{\downarrow}$-application.
- $\hat{\mathsf{q}}^{\mathsf{L}}_3{\downarrow}\dfrac{[A \parr C] \triangleleft B}{\langle A \triangleleft B\rangle \parr C}$: As before, because of the symmetries of Q, we only need to consider the case where $A = a^\perp$ and $B = h^\perp$. There are now five subcases of how to match C:
  - $C = \langle a \triangleleft b\rangle \otimes \langle c \triangleleft d\rangle$. We get the cycle $(e \otimes h) \parr \langle b \triangleleft h^\perp\rangle \parr \langle e^\perp \triangleleft b^\perp\rangle$ in the premise of the $\hat{\mathsf{q}}^{\mathsf{L}}_3{\downarrow}$-application.
  - $C = \langle e \triangleleft f\rangle \otimes \langle g \triangleleft h\rangle$. We get the cycle $h \triangleleft h^\perp$ in the premise of the $\hat{\mathsf{q}}^{\mathsf{L}}_3{\downarrow}$-application.
  - $C = e^\perp \triangleleft b^\perp$. We get the cycle $(e \otimes h) \parr \langle e^\perp \triangleleft h^\perp\rangle$ in the premise of the $\hat{\mathsf{q}}^{\mathsf{L}}_3{\downarrow}$-application.

- $C = g^\perp \triangleleft d^\perp$. We get the cycle $(b \otimes d) \parr (e \otimes h) \parr \langle d^\perp \triangleleft h^\perp \rangle \parr \langle e^\perp \triangleleft b^\perp \rangle$ in the premise of the $\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}$-application.
- $C = c^\perp \triangleleft f^\perp$. We get the cycle $(f \otimes h) \parr \langle f^\perp \triangleleft h^\perp \rangle$ in the premise of the $\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}$-application.

- $\hat{\mathsf{q}}_3^{\mathsf{R}}{\downarrow} \dfrac{A \triangleleft [B \parr C]}{\langle A \triangleleft B \rangle \parr C}$: Similar to $\hat{\mathsf{q}}_3^{\mathsf{L}}{\downarrow}$, by conjugacy.

- $\hat{\mathsf{q}}_2{\downarrow} \dfrac{A \triangleleft B}{A \parr B}$: The possible values for the ordered pair (A, B) are all pairs of distinct formulas in the sequent Γ_Q in Equation (11). We first look at the case $A = \langle a \triangleleft b \rangle \otimes \langle c \triangleleft d \rangle$ and $B = \langle e \triangleleft f \rangle \otimes \langle g \triangleleft h \rangle$. Here we get the cycle $\langle d \triangleleft g \rangle \parr \langle g^\perp \triangleleft d^\perp \rangle$ in the premise. The case $A = \langle e \triangleleft f \rangle \otimes \langle g \triangleleft h \rangle$ and $B = \langle a \triangleleft b \rangle \otimes \langle c \triangleleft d \rangle$ is symmetric to the this one via the automorphism β. Otherwise, either A or B (or both) have the form $x^\perp \triangleleft y^\perp$. It suffices to treat all the cases $R = x^\perp \triangleleft y^\perp$. This is because conjugation exchanges the roles of A and B in the $\mathsf{q}_2{\downarrow}$-rule, and Q is equal to its own conjugate up to the variable renaming performed by γ. We may also without loss of generality assume that $A = a^\perp \triangleleft h^\perp$; as before, this relies on the transitive action of the automorphisms of Q on the $x^\perp \triangleleft y^\perp$ that it contains. There are now five cases for B:
  - $B = \langle a \triangleleft b \rangle \otimes \langle c \triangleleft d \rangle$. We get the cycle $a^\perp \triangleleft a$ in the premise.
  - $B = \langle e \triangleleft f \rangle \otimes \langle g \triangleleft h \rangle$. We get the cycle $h^\perp \triangleleft h$ in the premise.
  - $B = e^\perp \triangleleft b^\perp$. We get the cycle $(e \otimes h) \parr \langle h^\perp \triangleleft e^\perp \rangle$ in the premise.
  - $B = g^\perp \triangleleft d^\perp$. We get the cycle $(a \otimes d) \parr \langle a^\perp \triangleleft d^\perp \rangle$ in the premise.
  - $B = c^\perp \triangleleft f^\perp$. We get the cycle $(f \otimes h) \parr \langle h^\perp \triangleleft f^\perp \rangle$ in the premise.

- $\hat{\mathsf{s}}_3 \dfrac{[A \parr C] \otimes B}{(A \otimes B) \parr C}$: There are two possibilities to match $A \otimes B$: either with $\langle a \triangleleft b \rangle \otimes \langle c \triangleleft d \rangle$ or with $\langle e \triangleleft f \rangle \otimes \langle g \triangleleft h \rangle$. Due to the commutativity of $\otimes$, we have four possibilities to match A and B. Due to the symmetries discussed above, we only need to consider the case where $A = a \triangleleft b$ and $B = c \triangleleft d$. There are now five cases how to match C:
  - $C = \langle e \triangleleft f \rangle \otimes \langle g \triangleleft h \rangle$. We get the cycle $(f \otimes c) \parr \langle c^\perp \triangleleft f^\perp \rangle$ in the premise.
  - $C = a^\perp \triangleleft h^\perp$. We get the cycle $(h^\perp \otimes c) \parr \langle c^\perp \triangleleft f^\perp \rangle \parr (f \otimes h)$ in the premise.
  - $C = e^\perp \triangleleft b^\perp$. We get the cycle $(e^\perp \otimes d) \parr \langle g^\perp \triangleleft d^\perp \rangle \parr (e \otimes g)$ in the premise.
  - $C = g^\perp \triangleleft d^\perp$. We get the cycle $d^\perp \otimes d$ in the premise.
  - $C = c^\perp \triangleleft f^\perp$. We get the cycle $c^\perp \otimes c$ in the premise.

- $\hat{\mathsf{s}}_2 \dfrac{A \otimes B}{A \parr B}$: This case is already subsumed by the case for $\hat{\mathsf{q}}_2{\downarrow}$.

In this way, we have completed the proof of the following proposition.

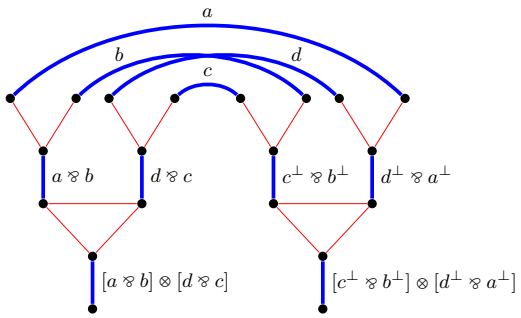
▶ **Proposition 5.2.** *The formula Q shown in Equation* (10) *is not provable in* BV.

▶ **Theorem 5.3.** *The theorems of* BV *form a proper subset of the theorems of pomset logic.*

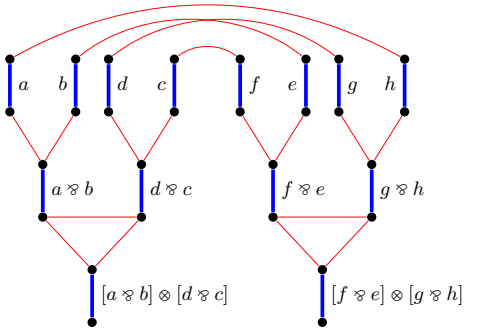
Proof. This follows immediately from Propositions 5.1 and 5.2. ◀

6 Conclusion

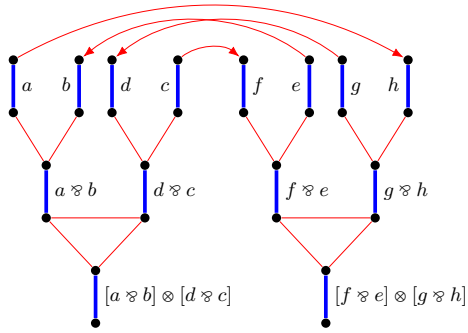
Let us end this paper by giving some historical perspective and some explanation how the formula Q has been found. The main reason that it took more than 20 year to find this (rather simple) formula was that everyone (including the second author) was looking into the wrong direction, trying to prove that BV and pomset logic are the same. This changed only after the first author (not being aware of the pomset logic vs. BV problem) observed



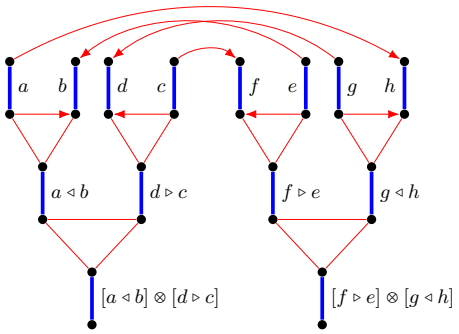
(a) A tree-like RB-prenet for a linear version of the medial rule of system SKS. Note that it does *not* satisfy the MLL_0 correctness criterion, and therefore also not the pomset criterion.



(b) A variation of the prenet on the left. The undirected R-edges on the top correspond to the addition of $a^\perp \otimes h^\perp, b^\perp \otimes e^\perp, d^\perp \otimes g^\perp, c^\perp \otimes f^\perp$. Note that the prenet is still *not* correct.



(c) The R-edges on top are now directed, corresponding to $a^\perp \triangleleft h^\perp, b^\perp \triangleright e^\perp, d^\perp \triangleright g^\perp, c^\perp \triangleleft f^\perp$. This modification validates the pomset logic correctness criterion, but the resulting sequent is not provable in BV.



(d) Adding more R-edges does preserve provability in pomset logic, but showing that the resulting sequent is not provable in BV is easier now, as every possible rule application breaks pomset correctness.

Figure 7 From the medial of SKS to our counterexample.

that checking pomset logic correctness is **coNP**-complete [18]. Since it had been observed before that BV is **NP**-complete [15], this immediately entailed that either **NP** = **coNP** or BV ≠ pomset logic.

Unfolding and dissecting the proof of **coNP**-completeness of pomset logic correctness led to a relation to classical logic provability. The details of this are subject of ongoing research and would go beyond the scope of this paper. But the outcome let us to the study of linear inferences [5, 6] which are a special case of balanced tautologies [27]. We were looking at linear inferences that are tautologies in classical logic but not provable in linear logic. The simplest such inference is $(A \wedge D) \vee (B \wedge C) \Rightarrow [A \vee B] \wedge [D \vee C]$, which corresponds to the medial rule of system SKS [2], a formulation of classical logic in the calculus of structures. Its linear version $(A \otimes D) \parr (B \otimes C) \multimap [A \parr B] \otimes [D \parr C]$ is, of course, not a theorem of MLL_0. This can be immediately seen by inspecting the RB-prenet for the formula $(a \otimes d) \parr (b \otimes c) \multimap [a \parr b] \otimes [c \parr d]$, which is shown in Figure 7a, and which contains several (chordless) æ-cycles. Then, on the right of that "medial RB-prenet", in Figure 7b, we replace the B-edges corresponding to the atoms by a pair of B-edges connected by an (undirected) R-edge. This does not affect provability, as no æ-cycles are added or removed. Then, in Figure 7c, we give these new R-edges a direction. By choosing the "right" direction, we can break all æ-cycles, which means the result becomes correct with respect to the pomset logic correctness criterion. But the resulting formula (or sequent) remains unprovable in

BV. To simplify the proof of non-provability in BV, we added further R-edges, as shown in Figure 7d, that do not break provability in pomset logic. The result is an intermediate step between the RB-prenets in Figure 4 and Figure 5.

The knowledge that BV and pomset logic are different, leads to four immediate open problems: (i) can we find a proof net correctness criterion for BV, (ii) can we find a deductive proof system for pomset logic that is independent from the prenets[9], (iii) which of the two logics is better, and (iv) are these two the only ones, or are there more logics having these three connectives and being conservative over MLL_0?

References

1 V. Michele Abrusci and Paul Ruet. Non-commutative logic I: the multiplicative fragment. *Annals of Pure and Applied Logic*, 101(1):29–64, 1999. doi:10.1016/S0168-0072(99)00014-7.

2 Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba, December 3-7, 2001, Proceedings*, volume 2250 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2001. doi:10.1007/3-540-45653-8_24.

3 Paola Bruscoli. A purely logical account of sequentiality in proof search. In Peter J. Stuckey, editor, *Logic Programming, 18th International Conference, ICLP 2002, Copenhagen, Denmark, July 29 - August 1, 2002, Proceedings*, volume 2401 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 2002. doi:10.1007/3-540-45619-8_21.

4 Denis Béchet, Philippe de Groote, and Christian Retoré. A complete axiomatisation for the inclusion of series-parallel partial orders. In Hubert Comon, editor, *Rewriting Techniques and Applications, 8th International Conference, RTA-97, Sitges, Spain, June 2-5, 1997, Proceedings*, volume 1232 of *Lecture Notes in Computer Science*, pages 230–240. Springer, 1997. doi:10.1007/3-540-62950-5_74.

5 Anupam Das and Alex A. Rice. New minimal linear inferences in boolean logic independent of switch and medial. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.14.

6 Anupam Das and Lutz Straßburger. On linear rewriting systems for boolean logic and some applications to proof theory. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:9)2016.

7 Arnaud Fleury and Christian Retoré. The mix rule. *Mathematical Structures in Computer Science*, 4(2):273–285, 1994. doi:10.1017/S0960129500000451.

8 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, January 1987. doi:10.1016/0304-3975(87)90045-4.

9 Jean-Yves Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 69–108. American Mathematical Society, Providence, RI, 1989. Proceedings of a Summer Research Conference held June 14–20, 1987. doi:10.1090/conm/092/1003197.

10 Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1), January 2007. According to the author, the published version contains errors introduced by the editorial processing; the authoritative version is arXiv:cs/9910023. doi:10.1145/1182613.1182614.

11 Alessio Guglielmi and Lutz Straßburger. Non-commutativity and MELL in the Calculus of Structures. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Laurent Fribourg, editors, *Computer Science Logic*, volume 2142, pages 54–68. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi:10.1007/3-540-44802-0_5.

[9] The sequent system proposed by Slavnov [25], uses labels for encoding the paths in the proof net.

12 Ross Horne and Alwen Tiu. Constructing weak simulations from linear implications for processes with private names. *Mathematical Structures in Computer Science*, 29(8):1275–1308, 2019. doi:10.1017/S0960129518000452.

13 Ross Horne, Alwen Tiu, Bogdan Aman, and Gabriel Ciobanu. De morgan dual nominal quantifiers modelling private names in non-commutative logic. *ACM Transactions on Computational Logic*, 20(4):22:1–22:44, 2019. doi:10.1145/3325821.

14 Ozan Kahramanoğulları. System BV without the Equalities for Unit. In *Computer and Information Sciences - ISCIS 2004*, volume 3280, pages 986–995. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-30182-0_99.

15 Ozan Kahramanoğulları. System BV is NP-complete. *Annals of Pure and Applied Logic*, 152(1):107–121, March 2008. doi:10.1016/j.apal.2007.11.005.

16 Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170, 1958. URL: http://www.jstor.org/stable/2310058.

17 Alain Lecomte and Christian Retoré. Pomset logic as an alternative categorial grammar. In *Formal Grammar*, pages 181–196. FoLLI, 1995.

18 Lê Thành Dũng Nguyên. Complexity of correctness for pomset logic proof nets, 2020. arXiv:1912.10606.

19 Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. doi:10.2307/421090.

20 Christian Retoré. *Réseaux et séquents ordonnés*. PhD thesis, Université Paris VII, February 1993. URL: https://tel.archives-ouvertes.fr/tel-00585634.

21 Christian Retoré. Pomset logic: A non-commutative extension of classical linear logic. In *Typed Lambda Calculi and Applications*, volume 1210, pages 300–318. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi:10.1007/3-540-62688-3_43.

22 Christian Retoré. Pomset Logic as a Calculus of Directed Cographs. Research Report 3714, INRIA, June 1999. URL: https://hal.inria.fr/inria-00072953.

23 Christian Retoré. Handsome proof-nets: perfect matchings and cographs. *Theoretical Computer Science*, 294(3):473–488, February 2003. doi:10.1016/S0304-3975(01)00175-X.

24 Christian Retoré. Pomset Logic: The other approach to noncommutativity in logic. In Claudia Casadio and Philip J. Scott, editors, *Joachim Lambek: the interplay of mathematics, logic, and linguistics*, Outstanding Contributions to Logic, chapter 9, pages 299–346. Springer, 2021. doi:10.1007/978-3-030-66545-6_9.

25 Sergey Slavnov. On noncommutative extensions of linear logic. *Logical Methods in Computer Science*, Volume 15, Issue 3, September 2019. doi:10.23638/LMCS-15(3:30)2019.

26 Lutz Straßburger. System NEL is undecidable. *Electronic Notes in Theoretical Computer Science*, 84:166–177, 2003. doi:10.1016/S1571-0661(04)80853-3.

27 Lutz Straßburger. Extension without cut. *Annals of Pure and Applied Logic*, 163(12):1995–2007, 2012. doi:10.1016/j.apal.2012.07.004.

28 Lutz Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD Thesis, Technische Universität Dresden, 2003. URL: https://www.lix.polytechnique.fr/~lutz/papers/dissvonlutz.pdf.

29 David N. Yetter. Quantales and (noncommutative) linear logic. *The Journal of Symbolic Logic*, 55(1):41–64, March 1990. doi:10.2307/2274953.

Revisiting Parameter Synthesis for One-Counter Automata

Guillermo A. Pérez
University of Antwerp, Flanders Make, Belgium

Ritam Raha
University of Antwerp, Belgium
LaBRI, University of Bordeaux, France

Abstract

We study the synthesis problem for one-counter automata with parameters. One-counter automata are obtained by extending classical finite-state automata with a counter whose value can range over non-negative integers and be tested for zero. The updates and tests applicable to the counter can further be made parametric by introducing a set of integer-valued variables called parameters. The synthesis problem for such automata asks whether there exists a valuation of the parameters such that all infinite runs of the automaton satisfy some ω-regular property. Lechner showed that (the complement of) the problem can be encoded in a restricted one-alternation fragment of Presburger arithmetic with divisibility. In this work (i) we argue that said fragment, called $\forall\exists_R\text{PAD}^+$, is unfortunately undecidable. Nevertheless, by a careful re-encoding of the problem into a decidable restriction of $\forall\exists_R\text{PAD}^+$, (ii) we prove that the synthesis problem is decidable in general and in **2NEXP** for several fixed ω-regular properties. Finally, (iii) we give polynomial-space algorithms for the special cases of the problem where parameters can only be used in counter tests.

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

Keywords and phrases Parametric one-counter automata, Reachability, Software Verification

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.33

Related Version *Full Version*: https://arxiv.org/abs/2005.01071

Funding This work was supported by the Belgian FWO "SAILor" project (G030020N).

Acknowledgements We thank all anonymous reviewers who carefully read earlier version of this work and helped us polish the paper. We also thank Michaël Cadilhac, Nathanaël Fijalkow, Philip Offtermatt, and Mikhail R. Starchak for useful feedback; Antonia Lechner and James Worrell for having brought the inconsistencies in [6, 21] to our attention; Radu Iosif and Marius Bozga for having pointed out precisely what part of the argument in both papers is incorrect.

1 Introduction

Our interest in one-counter automata (OCA) with parameters stems from their usefulness as models of the behaviour of programs whose control flow is determined by *counter variables*.

```
def funprint(x):
    i = 0
    i += x
    while i >= 0:
        if i == 0:
            print("Hello")
        if i == 1:
            print("world")
        if i >= 2:
            assert(False)
        i -= 1
    # end program
```

Indeed, the executions of such a program can be over-approximated by its control-flow graph (CFG) [1]. The CFG can be leveraged to get a *conservative response* to interesting questions about the program, such as: "is there a value of x such that the false assertion is avoided?" The CFG abstracts away all variables and their values (see Figure 1) and this introduces non-determinism. Hence, the question becomes: "is it the case that all paths from the initial vertex avoid the one labelled with 10?" In this particular example, the abstraction is too *coarse* and thus we obtain a false negative. In such cases, the abstraction of the program should be refined [9]. A natural refinement of the CFG in this context is obtained by tracking the value of i (cf. program graphs in [2]). The result is an OCA with parameters such that: For $x \in \{0, 1\}$ it has no run that reaches the state labelled with 10. This is an instance of a *safety (parameter) synthesis problem* for which the answer is positive.

Figure 1 On the left, the CFG with vertex labels corresponding to source code line numbers; on the right, the CFG extended by tracking the value of i.

In this work, we focus on the parameter synthesis problems for given OCA with parameters and do not consider the problem of obtaining such an OCA from a program (cf. [10]).

Counter automata [29] are a classical model that extend finite-state automata with integer-valued counters. These have been shown to be useful in modelling complex systems, such as programs with lists and XML query evaluation algorithms [5, 8]. Despite their usefulness as a modelling formalism, it is known that two counters suffice for counter automata to become Turing powerful. In particular, this means that most interesting questions about them are undecidable [29]. To circumvent this, several restrictions of the model have been studied in the literature, e.g. reversal-bounded counter automata [18] and automata with a single counter. In this work we focus on an extension of the latter: OCA with parametric updates and parametric tests.

An existential version of the synthesis problems for OCA with parameters was considered by Göller et al. [12] and Bollig et al. [4]. They ask whether there exist a valuation of the parameters and a run of the automaton which satisfies a given ω-regular property. This is in contrast to the present problem where we quantify runs universally. (This is required for the conservative-approximation use case described in the example above.) We note that, of those two works, only [12] considers OCA with parameters allowed in both counter updates and counter tests while [4] studies OCA with parametric tests only. In this paper, unless explicitly stated otherwise, we focus on OCA with parametric tests and updates like in [12]. Further note that the model we study has an asymmetric set of tests that can be applied to the counter: lower-bound tests, and equality tests (both parametric and non-parametric). The primary reason for this is that adding upper-bound tests results in a model for which even the decidability of the (arguably simpler) existential reachability synthesis problem is a long-standing open problem [7]. Namely, the resulting model corresponds to Ibarra's *simple programs* [17].

In both [12] and [4], the synthesis problems for OCA with parameters were stated as open. Later, Lechner [21] gave an encoding for the complement of the synthesis problems into a one-alternation fragment of Presburger arithmetic with divisibility (PAD). Her encoding relies

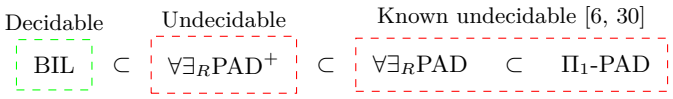


Figure 2 Syntactical fragments of PAD ordered w.r.t. their language (of sentences).

Table 1 Known and new complexity bounds for parameter synthesis problems.

	Lower bound	Upper bound
LTL	**PSPACE**-hard [32]	in **3NEXP** (Cor. 17)
Reachability	**coNP**-hard (Prop. 23)	in **2NEXP** (Thm. 10)
Safety, Büchi, coBüchi	$\mathbf{NP^{NP}}$-hard [21, 22]	

on work by Haase et al. [16], which shows how to compute a linear-arithmetic representation of the reachability relation of OCA (see [24] for an implementation). In the same work, Haase et al. show that the same can be achieved for OCA with parameters using the divisibility predicate. In [21], Lechner goes on to consider the complexity of (validity of sentences in) the language corresponding to the one-alternation fragment her encoding targets. An earlier paper [6] by Bozga and Iosif argues that the fragment is decidable and Lechner carefully repeats their argument while leveraging bounds on the bitsize of solutions of existential PAD formulas [23] to argue the complexity of the fragment is **co2NEXP**. For ω-regular properties given as a linear temporal logic (LTL) formula, her encoding is exponential in the formula and thus it follows that the LTL synthesis problem is decidable and in **3NEXP**.

Problems in the literature. *Presburger arithmetic* is the first-order theory of $\langle \mathbb{Z}, 0, 1, +, < \rangle$. *Presburger arithmetic with divisibility (PAD)* is the extension of PA obtained when we add a binary *divisibility predicate*. The resulting language is undecidable [30]. In fact, a single quantifier alternation already allows to encode general multiplication, thus becoming undecidable [27]. However, the purely existential (Σ_0) and purely universal (Π_0) fragments have been shown to be decidable [3, 26].

The target of Lechner's encoding is ∀∃$_R$PAD$^+$, a subset of all sentences in the Π_1-fragment of PAD. Such sentences look as follows: $\forall \boldsymbol{x} \exists \boldsymbol{y} \bigvee_{i \in I} \bigwedge_{j \in J_i} f_j(\boldsymbol{x}) \mid g_j(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_i(\boldsymbol{x}, \boldsymbol{y})$ where φ is a quantifier-free PAD formula without divisibility. Note that all divisibility constraints appear in positive form (hence the $^+$) and that, within divisibility constraints, the existentially-quantified variables y_i appear only on the right-hand side (hence the $\exists_{\mathrm{R}}$). In [6], the authors give a quantifier-elimination procedure for sentences in a further restricted fragment we call the Bozga-Iosif-Lechner fragment (BIL) that is based on "symbolically applying" the generalized Chinese remainder theorem (CRT) [20]. Their procedure does not eliminate all quantifiers but rather yields a sentence in the Π_0-fragment of PAD. (Decidability of the BIL language would then follow from the result of Lipshitz [26].) Then, they *briefly argue* how the algorithm generalizes to ∀∃$_R$PAD$^+$. There are two crucial problems in the argument from [6] that we have summarized here (and which were reproduced in Lechner's work): First, the quantifier-elimination procedure of Bozga and Iosif does not directly work for BIL. Indeed, not all BIL sentences satisfy the conditions required for the CRT to be applicable as used in their algorithm. Second, there is no way to generalize their algorithm to ∀∃$_R$PAD$^+$ since the language is undecidable. Interestingly, undecidability follows directly from other results in [6, 21]. In Lechner's thesis [22], the result from [6] was stated as being under review. Correspondingly, the decidability of the synthesis problems for OCA with parameters was only stated conditionally on ∀∃$_R$PAD$^+$ being decidable.

Our contribution. In Section 2, using developments from [6, 21], we argue that $\forall\exists_R\text{PAD}^+$ is undecidable (Theorem 2). Then, in the same section, we "fix" the definition of the BIL fragment by adding to it a necessary constraint so that the quantifier-elimination procedure from [6] works correctly. For completeness, and to clarify earlier mistakes in the literature, we recall Lechner's analysis of the algorithm and conclude, just as she did, that the complexity of BIL is in **co2NEXP** [21] (Theorem 4). After some preliminaries regarding OCA with parameters in Section 3, we re-establish decidability of various synthesis problems in Section 4 (Theorem 10 and Corollary 17, see Table 1 for a summary). To do so, we follow Lechner's original idea from [21] to encode them into $\forall\exists_R\text{PAD}^+$ sentences. However, to ensure we obtain a BIL sentence, several parts of her encoding have to be adapted. Finally, in Section 5 we make small modifications to the work of Bollig et al. [4] to give more efficient algorithms that are applicable when only tests have parameters (Theorem 18 and Corollary 19).

2 Presburger Arithmetic with divisibility

Presburger arithmetic (PA) is the first-order theory over $\langle\mathbb{Z}, 0, 1, +, <\rangle$ where $+$ and $<$ are the standard addition and ordering of integers. *Presburger arithmetic with divisibility (PAD)* is the extension of PA obtained when we add the binary divisibility predicate $\mid$, where for all $a, b \in \mathbb{Z}$ we have $a \mid b \iff \exists c \in \mathbb{Z} : b = ac$. Let X be a finite set of first-order variables. A *linear polynomial* over $\boldsymbol{x} = (x_1, \ldots, x_n) \in X^n$ is given by the syntax rule: $p(\boldsymbol{x}) ::= \sum_{1\leq i\leq n} a_i x_i + b$, where the a_i, b and the first-order variables from $\boldsymbol{x}$ range over $\mathbb{Z}$. In general, quantifier-free PAD formulas have the grammar: $\varphi ::= \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid f(\boldsymbol{x})\ P\ g(\boldsymbol{x})$, where P can be the order predicate $<$ or the divisibility predicate $\mid$, and f, g are linear polynomials. We define the standard Boolean abbreviation $\varphi_1 \vee \varphi_2 \iff \neg(\neg\varphi_1 \wedge \neg\varphi_2)$. Moreover we introduce the abbreviations $f(x) \leq g(x) \iff f(x) < g(x) + 1$ and $f(x) = g(x) \iff f(x) \leq g(x) \wedge g(x) \leq f(x)$.

The *size* $|\varphi|$ of a PAD formula φ is defined by structural induction over $|\varphi|$: For a linear polynomial $p(\boldsymbol{x})$ we define $|p(\boldsymbol{x})|$ as the number of symbols required to write it if the coefficients are given in binary. Then, we define $|\varphi_1 \wedge \varphi_2| \stackrel{\text{def}}{=} |\varphi_1| + |\varphi_2| + 1$, $|\neg\varphi| \stackrel{\text{def}}{=} |\exists x.\varphi| \stackrel{\text{def}}{=} |\varphi| + 1$, $|f(\boldsymbol{x})\ P\ g(\boldsymbol{x})| \stackrel{\text{def}}{=} |f(\boldsymbol{x})| + |g(\boldsymbol{x})| + 1$.

2.1 Allowing one restricted alternation

We define the language $\forall\exists_R\text{PAD}$ of all PAD sentences allowing a universal quantification over some variables, followed by an existential quantification over variables that may not appear on the left-hand side of divisibility constraints. Formally, $\forall\exists_R\text{PAD}$ is the set of all PAD sentences of the form: $\forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_m \varphi(\boldsymbol{x}, \boldsymbol{y})$ where φ is a quantifier-free PAD formula and all its divisibility constraints are of the form $f(\boldsymbol{x}) \mid g(\boldsymbol{x}, \boldsymbol{y})$.

Positive-divisibility fragment. We denote by $\forall\exists_R\text{PAD}^+$ the subset of $\forall\exists_R\text{PAD}$ sentences φ where the negation operator can only be applied to the order predicate $<$ and the only other Boolean operators allowed are conjunction and disjunction. In other words, $\forall\exists_R\text{PAD}^+$ is a restricted negation normal form in which divisibility predicates cannot be negated. Lechner showed in [21] that all $\forall\exists_R\text{PAD}$ sentences can be translated into $\forall\exists_R\text{PAD}^+$ sentences.

▶ **Proposition 1** (Lechner's trick [21]). *For all φ_1 in $\forall\exists_R PAD$ one can compute φ_2 in $\forall\exists_R PAD^+$ such that φ_1 is true if and only if φ_2 is true.*

2.2 Undecidability of both one-alternation fragments

We will now prove that the language $\forall\exists_R\text{PAD}^+$ is undecidable, that is, to determine whether a given sentence from $\forall\exists_R\text{PAD}^+$ is true is an undecidable problem.

▶ **Theorem 2.** *The language $\forall\exists_R PAD^+$ is undecidable.*

From Proposition 1 it follows that arguing $\forall\exists_R\text{PAD}$ is undecidable suffices to prove the theorem. The latter was proven in [6]. More precisely, they show the complementary language is undecidable. Their argument consists in defining the least-common-multiple predicate, the squaring predicate, and subsequently integer multiplication. Undecidability thus follows from the MRDP theorem [28] which states that satisfiability for such equations (i.e. Hilbert's 10th problem) is undecidable. Hence, Theorem 2 is a direct consequence of the following result.

▶ **Proposition 3** (From [6]). *The language $\forall\exists_R PAD$ is undecidable.*

2.3 The Bozga-Iosif-Lechner fragment

The Bozga-Iosif-Lechner (BIL) fragment is the set of all $\forall\exists_R\text{PAD}^+$ sentences of the form:

$$\forall x_1 \dots \forall x_n \exists y_1 \dots \exists y_m (\boldsymbol{x} < 0) \vee \bigvee_{i \in I} \bigwedge_{j \in J_i} (f_j(\boldsymbol{x}) \mid g_j(\boldsymbol{x}, \boldsymbol{y}) \wedge f_j(\boldsymbol{x}) > 0) \wedge \varphi_i(\boldsymbol{x}) \wedge \boldsymbol{y} \geq \mathbf{0}$$

where $I, J_i \subseteq \mathbb{N}$ are all finite index sets, the f_j and g_j are linear polynomials and the $\varphi_i(\boldsymbol{x})$ are quantifier-free PA formulas over the variables $\boldsymbol{x}$. Note that, compared to $\forall\exists_R\text{PAD}^+$, BIL sentences only constraint non-negative values of $\boldsymbol{x}$. (This technicality is necessary due to our second constraint below.) For readability, henceforth, we omit $(\boldsymbol{x} < 0)$ and just assume the $\boldsymbol{x}$ take non-negative integer values, i.e. from $\mathbb{N}$. Additionally, it introduces the following three important constraints:

1. The $\boldsymbol{y}$ variables may only appear on the right-hand side of divisibility constraints.
2. All divisibility constraints $f_j(\boldsymbol{x}) \mid g_j(\boldsymbol{x}, \boldsymbol{y})$ are conjoined with $f_j(\boldsymbol{x}) > 0$.
3. The $\boldsymbol{y}$ variables are only allowed to take non-negative values.

It should be clear that the first constraint is necessary to avoid undecidability. Indeed, if the $\boldsymbol{y}$ variables were allowed in the PA formulas $\varphi_i(\boldsymbol{x})$ then we could circumvent the restrictions of where they appear in divisibilities by using equality constraints. The second constraint is similar in spirit. Note that if $a = 0$ then $a \mid b$ holds if and only if $b = 0$ so if the left-hand side of divisibility constraints is allowed to be 0 then we can encode PA formulas on $\boldsymbol{x}$ and $\boldsymbol{y}$ as before. Also, the latter (which was missing in [6, 21]) will streamline the application of the generalized Chinese remainder theorem in the algorithm described in the sequel. While the third constraint is not required for decidability, it is convenient to include it for Section 4, where we encode instances of the synthesis problem into the BIL fragment.

In the rest of this section, we recall the decidability proof by Bozga and Iosif [6] and refine Lechner's analysis [21] to obtain the following complexity bound.

▶ **Theorem 4.** *The BIL-fragment language is decidable in* **co2NEXP**.

The idea of the proof is as follows: We start from a BIL sentence. First, we use the *generalized Chinese remainder theorem* (CRT, for short) to replace all of the existentially quantified variables in it with a single universally quantified variable. We thus obtain a sentence in $\forall\text{PAD}$ (i.e. the Π_0-fragment of PAD) and argue that the desired result follows from the bounds on the bitsize of satisfying assignments for existential PAD formulas [23].

▶ **Theorem 5** (Generalized Chinese remainder theorem [20]). *Let $m_i \in \mathbb{N}_{>0}$, $a_i, r_i \in \mathbb{Z}$ for $1 \leq i \leq n$. Then, there exists $x \in \mathbb{Z}$ such that $\bigwedge_{i=1}^{n} m_i \mid (a_i x - r_i)$ if and only if:*

$$\bigwedge_{1 \leq i,j \leq n} \gcd(a_i m_j, a_j m_i) \mid (a_i r_j - a_j r_i) \wedge \bigwedge_{i=1}^{n} \gcd(a_i, m_i) \mid r_i.$$

The solution for x is unique modulo $\operatorname{lcm}(m'_1, \ldots, m'_n)$*, where* $m'_i = {}^{m_i}/_{\gcd(a_i, m_i)}$.

From a BIL sentence, we apply the CRT to the rightmost existentially quantified variable and get a sentence with one less existentially quantified variable and with gcd-expressions. Observe that the second restriction we highlighted for the BIL fragment (the conjunction with $f_j(\boldsymbol{x}) > 0$) is necessary for the correct application of the CRT. We will later argue that we can remove the gcd expressions to obtain a sentence in ∀PAD.

▶ **Example 6.** Consider the sentence:

$$\forall x \exists y_1 \exists y_2 \bigvee_{i \in I} \bigwedge_{j \in J_i} (f_j(x) \mid g_j(x, \boldsymbol{y}) \wedge f_j(x) > 0) \wedge \varphi_i(x) \wedge \boldsymbol{y} \geq \mathbf{0}.$$

Let α_j denote the coefficient of y_2 in $g_j(x, \boldsymbol{y})$ and $r_j(x, y_1) \stackrel{\text{def}}{=} -(g_j(x, \boldsymbol{y}) - \alpha_j y_2)$. We can rewrite the above sentence as $\forall x \exists y_1 \bigvee_{i \in I} \psi_i(x, y_1) \wedge \varphi'_i(x) \wedge y_1 \geq 0$ where:

$$\psi_i(x, y_1) = \exists y_2 \bigwedge_{j \in J_i} (f_j(x) \mid (\alpha_j y_2 - r_j(x, y_1))) \wedge y_2 \geq 0, \text{ and}$$
$$\varphi'_i(x) = \varphi_i(x) \wedge \bigwedge_{j \in J_i} f_j(x) > 0.$$

Applying the CRT, $\psi_i(x, y_1)$ can equivalently be written as follows:

$$\bigwedge_{j,k \in J_i} \gcd(\alpha_k f_j(x), \alpha_j f_k(x)) \mid (\alpha_j r_k(x, y_1) - \alpha_k r_j(x, y_1)) \wedge \bigwedge_{j \in J_i} \gcd(\alpha_j, f_j(x)) \mid r_j(x, y_1).$$

Note that we have dropped the $y_2 \geq 0$ constraint without loss of generality since the CRT states that the set of solutions forms an arithmetic progression containing infinitely many positive (and negative) integers. This means the constraint will be trivially satisfied for any valuation of x and y_1 which satisfies $\psi_i(x, y_1) \wedge \varphi_i(x) \wedge y_1 \geq 0$ for some $i \in I$. Observe that y_1 only appears in polynomials on the right-hand side of divisibilities.

The process sketched in the example can be applied in general to BIL sentences sequentially starting from the rightmost quantified y_i. At each step, the size of the formula is at most squared. In what follows, it will be convenient to deal with a single polyadic gcd instead of nested binary ones. Thus, using associativity of gcd and pushing coefficients inwards – i.e. using the equivalence $a \cdot \gcd(x, y) \equiv \gcd(ax, ay)$ for $a \in \mathbb{N}$ – we finally obtain a sentence:

$$\forall x_1 \ldots \forall x_n \bigvee_{i \in I} \bigwedge_{j \in L_i} (\gcd(\{f'_{j,k}(\boldsymbol{x})\}_{k=1}^{K_j}) \mid g'_j(\boldsymbol{x})) \wedge \varphi'_i(\boldsymbol{x}) \tag{1}$$

where $|L_i|$, $|K_j|$, and the coefficients may all be doubly-exponential in the number m of removed variables, due to iterated squaring.

Eliminating the gcd operator. In this next step, our goal is to obtain an ∀PAD sentence from Equation (1). Recall that ∀PAD "natively" allows for negated divisibility constraints. (That is, without having to encode them using Lechner's trick.) Hence, to remove expressions in terms of gcd from Equation (1), we can use the following identity:

$$\gcd(f_1(\boldsymbol{x}),\ldots,f_n(\boldsymbol{x})) \mid g(\boldsymbol{x}) \iff \forall d \left(\bigwedge_{i=1}^{n} d \mid f_i(\boldsymbol{x}) \right) \to d \mid g(\boldsymbol{x}).$$

This substitution results in a constant blowup of the size of the sentence. The above method gives us a sentence $\forall \boldsymbol{x} \forall d \psi(\boldsymbol{x}, d)$, where $\psi(\boldsymbol{x}, d)$ is a quantifier-free PAD formula. To summarize:

▶ **Lemma 7.** *For any BIL sentence $\varphi = \forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_m \bigvee_{i \in I} \varphi_i(\boldsymbol{x}, \boldsymbol{y})$ we can construct an ∀PAD sentence $\psi = \forall x_1 \ldots \forall x_n \forall d \bigvee_{i \in I} \psi_i(\boldsymbol{x}, d)$ such that: φ is true if and only if ψ is true and for all $i \in I$, $|\psi_i| \leq |\varphi_i|^{2^m}$. The construction is realizable in time $\mathcal{O}(|\varphi|^{2^m})$.*

To prove Theorem 4, the following small-model results for purely existential PAD formulas and BIL will be useful.

▶ **Theorem 8** ([23, Theorem 14])**.** *Let $\varphi(x_1, \ldots, x_n)$ be a ∃PAD formula. If φ has a solution then it has a solution $(a_1, \ldots, a_n) \in \mathbb{Z}^n$ with the bitsize of each a_i bounded by $|\varphi|^{\text{poly}(n)}$.*

▶ **Corollary 9.** *Let $\forall x_1 \ldots \forall x_n \varphi(x_1, \ldots, x_n)$ be a BIL sentence. If $\neg\varphi$ has a solution then it has a solution $(a_1, \ldots, a_n) \in \mathbb{Z}^n$ with the bitsize of each a_i bounded by $|\varphi|^{2^m \text{poly}(n+1)}$.*

Proof. Using Lemma 7, we translate the BIL sentence to $\forall x_1 \ldots \forall x_n \forall d \psi(\boldsymbol{x}, d)$, where the latter is an ∀PAD sentence. Then, using Theorem 8, we get that the ∃PAD formula $\neg\psi(\boldsymbol{x}, d)$ admits a solution if and only if it has one with bitsize bounded by $|\psi|^{\text{poly}(n+1)}$. Now, from Lemma 7 we have that $|\psi|$ is bounded by $|\varphi|^{2^m}$. Hence, we get that the bitsize of a solution is bounded by: $|\varphi|^{2^m \text{poly}(n+1)}$. ◀

We are now ready to prove the theorem.

Proof of Theorem 4. As in the proof of Corollary 9, we translate the BIL sentence to $\forall x_1 \ldots \forall x_n \forall d \psi(\boldsymbol{x}, d)$. Note that our algorithm thus far runs in time: $\mathcal{O}\left(|\varphi|^{2^m}\right)$. By Corollary 9, if $\neg\psi(\boldsymbol{x}, d)$ has a solution then it has one encodable in binary using a doubly exponential amount of bits with respect to the size of the input BIL sentence. The naive guess-and-check decision procedure applied to $\neg\psi(\boldsymbol{x}, d)$ gives us a **co2NEXP** algorithm for BIL sentences. Indeed, after computing $\psi(\boldsymbol{x}, d)$ and guessing a valuation, checking it satisfies $\neg\psi$ takes polynomial time in the bitsize of the valuation and $|\psi|$, hence doubly exponential time in $|\varphi|$. ◀

3 Succinct One-Counter Automata with Parameters

We now define OCA with parameters and recall some basic properties. The concepts and observations we introduce here are largely taken from [16] and the exposition in [22].

A *succinct parametric one-counter automaton (SOCAP)* is a tuple $\mathcal{A} = (Q, T, \delta, X)$, where Q is a finite set of states, X is a finite set of parameters, $T \subseteq Q \times Q$ is a finite set of transitions and $\delta : T \to Op$ is a function that associates an operation to every transition. The set $Op = CU \uplus PU \uplus ZT \uplus PT$ is the union of: *Constant Updates* $CU \stackrel{\text{def}}{=} \{+a : a \in \mathbb{Z}\}$, *Parametric Updates* $PU \stackrel{\text{def}}{=} \{Sx : S \in \{+1, -1\}, x \in X\}$, *Zero Tests* $ZT \stackrel{\text{def}}{=} \{= 0\}$, and

Parametric Tests $PT \stackrel{\text{def}}{=} \{= x, \geq x : x \in X\}$. We denote by "$= 0$" or "$= x$" an *equality test* between the value of the counter and zero or the value of x respectively; by "$\geq x$", a *lower-bound test* between the values of the counter and x. A *valuation* $V : X \to \mathbb{N}$ assigns to every parameter a natural number. We assume CU are encoded in binary, hence the S in SOCAP. We omit "parametric" if $X = \emptyset$ and often write $q \xrightarrow{op} q'$ to denote $\delta(q, q') = op$.

A *configuration* is a pair (q, c) where $q \in Q$ and $c \in \mathbb{N}$ is the *counter value*. Given a valuation $V : X \to \mathbb{N}$ and a configuration (q_0, c_0), a *V-run from* (q_0, c_0) is a sequence $\rho = (q_0, c_0)(q_1, c_1)\dots$ such that for all $i \geq 0$ the following hold: $q_i \xrightarrow{op_{i+1}} q_{i+1}$; $c_i = 0$, $c_i = V(x)$, and $c_i \geq V(x)$, if $\delta(q_i, q_{i+1})$ is "$= 0$", "$= x$", and "$\geq x$", respectively; and c_{i+1} is obtained from c_i based on the counter operations. That is, c_{i+1} is c_i if $\delta(q_i, q_{i+1}) \in (ZT \cup PT)$; $c_i + a$ if $\delta(q_i, q_{i+1}) = +a$; $c_i + S \cdot V(x)$ if $\delta(q_i, q_{i+1}) = Sx$. We say ρ *reaches* a state $q_f \in Q$ if there exists $j \in \mathbb{N}$, such that $q_j = q_f$. Also, ρ reaches or visits a set of states $F \subseteq Q$ iff ρ reaches a state $q_f \in F$. If V is clear from the context we just write run instead of V-run.

The *underlying (directed) graph* of $\mathcal{A}$ is $G_{\mathcal{A}} = (Q, T)$. A V-run $\rho = (q_0, c_0)(q_1, c_1)\dots$ in $\mathcal{A}$ *induces* a path $\pi = q_0 q_1 \dots$ in $G_{\mathcal{A}}$. We assign weights to $G_{\mathcal{A}}$ as follows: For $t \in T$, weight(t) is 0 if $\delta(t) \in ZT \cup PT$; a if $\delta(t) = +a$; and $S \cdot V(x)$ if $\delta(t) = Sx$. We extend the weight function to finite paths in the natural way. Namely, we set $\text{weight}(q_0 \dots q_n) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} \text{weight}(q_i, q_{i+1})$.

Synthesis problems. *The synthesis problem* asks, given a SOCAP $\mathcal{A}$, a state q and an ω-regular property p, whether there exists a valuation V such that all infinite V-runs from $(q, 0)$ satisfy p. We focus on the following classes of ω-regular properties. Given a set of target states $F \subseteq Q$ and an infinite run $\rho = (q_0, c_0)(q_1, c_1)\dots$ we say ρ satisfies:

- the *reachability* condition if $q_i \in F$ for some $i \in \mathbb{N}$;
- the *Büchi* condition if $q_i \in F$ for infinitely many $i \in \mathbb{N}$;
- the *coBüchi* condition if $q_i \in F$ for finitely many $i \in \mathbb{N}$ only;
- the *safety* condition if $q_i \notin F$ for all $i \in \mathbb{N}$;
- the *linear temporal logic* (LTL) formula φ over a set of atomic propositions P – and with respect to a labelling function $f : Q \to 2^P$ – if $f(q_0)f(q_1)\dots \models \varphi$.[1]

We will decompose the synthesis problems into reachability sub-problems. It will thus be useful to recall the following connection between reachability (witnesses) and graph flows.

Flows. For a directed graph $G = (V, E)$, we denote the set of immediate successors of $v \in V$ by $vE \coloneqq \{w \in V \mid (v, w) \in E\}$ and the immediate predecessors of v by Ev, defined analogously. An *s–t flow* is a mapping $f : E \to \mathbb{N}$ that satisfies flow conservation: $\forall v \in V \setminus \{s, t\} : \sum_{u \in Ev} f(u, v) = \sum_{u \in vE} f(v, u)$. That is, the total incoming flow equals the total outgoing flow for all but the source and the target vertices. We then define the *value* of a flow f as: $|f| \stackrel{\text{def}}{=} \sum_{v \in sE} f(s, v) - \sum_{u \in Es} f(u, s)$. We denote by *support*$(f)$ the set $\{e \in E \mid f(e) > 0\}$ of edges with non-zero flow. A cycle in a flow f is a cycle in the sub-graph induced by support(f). For weighted graphs, we define $\text{weight}(f) \stackrel{\text{def}}{=} \sum_{e \in E} f(e)\text{weight}(e)$.

Path flows. Consider a path $\pi = v_0 v_1 \dots$ in G. We denote by f_π its Parikh image, i.e. f_π maps each edge e to the number of times e occurs in π. A flow f is called a *path flow* if there exists a path π such that $f = f_\pi$. Finally, we observe that an s–t path flow f in G induces a t–s path flow f' with $f'(u, v) = f(v, u)$, for all $(u, v) \in E$, in the skew transpose of G.

[1] See, e.g., [2] for the classical semantics of LTL.

4 Encoding Synthesis Problems into the BIL Fragment

In this section, we prove that all our synthesis problems are decidable. More precisely, we establish the following complexity upper bounds.

▶ **Theorem 10.** *The reachability, Büchi, coBüchi, and safety synthesis problems for succinct one-counter automata with parameters are all decidable in* **2NEXP**.

The idea is as follows: we focus on the coBüchi synthesis problem and reduce its complement to the truth value of a BIL sentence. To do so, we follow Lechner's encoding of the complement of the Büchi synthesis problem into $\forall\exists_R\text{PAD}^+$ [21]. The encoding heavily relies on an encoding for (existential) reachability from [16]. We take extra care to obtain a BIL sentence instead of an $\forall\exists_R\text{PAD}^+$ one as Lechner originally does.

It can be shown that the other synthesis problems reduce to the coBüchi one in polynomial time. The corresponding bounds thus follow from the one for coBüchi synthesis. The proof of the following lemma is given in the long version of the paper.

▶ **Lemma 11.** *The reachability, safety and the Büchi synthesis problems can be reduced to the coBüchi synthesis problem in polynomial time.*

Now the cornerstone of our reduction from the complement of the coBüchi synthesis problem to the truth value of a BIL sentence is an encoding of *reachability certificates* into $\forall\exists_R\text{PAD}$ formulas which are "almost" in BIL. In the following subsections we will focus on a SOCAP $\mathcal{A} = (Q, T, \delta, X)$ with $X = \{x_1, \ldots, x_n\}$ and often write $\boldsymbol{x}$ for $(x_1, \ldots, x_n)$. We will prove that the existence of a V-run from (q, c) to (q', c') can be reduced to the satisfiability problem for such a formula.

▶ **Proposition 12.** *Given states q, q', one can construct in deterministic exponential time in $|\mathcal{A}|$ a PAD formula: $\varphi^{(q,q')}_{\text{reach}}(\boldsymbol{x}, a, b) = \exists\boldsymbol{y} \bigvee_{i\in I} \varphi_i(\boldsymbol{x}, \boldsymbol{y}) \wedge \psi_i(\boldsymbol{y}, a, b) \wedge \boldsymbol{y} \geq \boldsymbol{0}$ such that $\forall\boldsymbol{x}\exists\boldsymbol{y} \bigvee_{i\in I} \varphi_i(\boldsymbol{x}, \boldsymbol{y}) \wedge \boldsymbol{y} \geq 0$ is a BIL sentence, the $\psi_i(\boldsymbol{y}, a, b)$ are quantifier-free PA formulas, and additionally:*

- *a valuation V of $X \cup \{a, b\}$ satisfies $\varphi^{(q,q')}_{\text{reach}}$ iff there is a V-run from $(q, V(a))$ to $(q', V(b))$;*
- *the bitsize of constants in $\varphi^{(q,q')}_{\text{reach}}$ is of polynomial size in $|\mathcal{A}|$;*
- *$|\varphi^{(q,q')}_{\text{reach}}|$ is at most exponential with respect to $|\mathcal{A}|$; and*
- *the number of $\boldsymbol{y}$ variables is polynomial with respect to $|\mathcal{A}|$.*

Below, we make use of this proposition to prove Theorem 10. Then, we prove some auxiliary results in Section 4.1 and, in Section 4.2, we present a sketch of our proof of Proposition 12.

We will argue that $\forall\boldsymbol{x}\exists a\exists b\varphi^{(q,q')}_{\text{reach}}(\boldsymbol{x}, a, b)$ can be transformed into an equivalent BIL sentence. Note that for this to be the case it suffices to remove the $\psi_i(\boldsymbol{y}, a, b)$ subformulas. Intuitively, since these are quantifier-free PA formulas, their set of satisfying valuations is *semi-linear* (see, for instance, [15]). Our intention is to remove the $\psi_i(\boldsymbol{y}, a, b)$ and replace the occurrences of $\boldsymbol{y}, a, b$ in the rest of $\varphi^{(q,q')}_{\text{reach}}(\boldsymbol{x}, a, b)$ with linear polynomials "generating" their set of solutions. This is formalized below.

Affine change of variables. Let $\boldsymbol{A} \in \mathbb{Z}^{m\times n}$ be an integer matrix of size $m \times n$ of rank r, and $\boldsymbol{b} \in \mathbb{Z}^m$. Let $\boldsymbol{C} \in \mathbb{Z}^{p\times n}$ be an integer matrix of size $p \times n$ such that $\left(\begin{smallmatrix}\boldsymbol{A}\\ \boldsymbol{C}\end{smallmatrix}\right)$ has rank s, and $\boldsymbol{d} \in \mathbb{Z}^p$. We write μ for the maximum absolute value of an $(s-1)\times(s-1)$ or $s \times s$ sub-determinant of the matrix $\left(\begin{smallmatrix}\boldsymbol{A} & \boldsymbol{b}\\ \boldsymbol{C} & \boldsymbol{d}\end{smallmatrix}\right)$ that incorporates at least r rows from $\begin{pmatrix}\boldsymbol{A} & \boldsymbol{b}\end{pmatrix}$.

▶ **Theorem 13** (From [33]). *Given integer matrices $\boldsymbol{A} \in \mathbb{Z}^{m\times n}$ and $\boldsymbol{C} \in \mathbb{Z}^{p\times n}$, integer vectors $\boldsymbol{b} \in \mathbb{Z}^m$ and $\boldsymbol{d} \in \mathbb{Z}^p$, and μ defined as above, there exists a finite set I, a collection of $n\times(n-r)$ matrices $\boldsymbol{E}^{(i)}$, and $n \times 1$ vectors $\boldsymbol{u}^{(i)}$, indexed by $i \in I$, all with integer entries bounded by $(n+1)\mu$ such that: $\{\boldsymbol{x} \in \mathbb{Z}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \wedge \boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d}\} = \bigcup_{i\in I}\{\boldsymbol{E}^{(i)}\boldsymbol{y} + \boldsymbol{u}^{(i)} : \boldsymbol{y} \in \mathbb{Z}^{n-r}, \boldsymbol{y} \geq \boldsymbol{0}\}$.*

We are now ready to prove Theorem 10.

Proof of Theorem 10. We will first prove that the complement of the coBüchi synthesis problem can be encoded into a BIL sentence. Recall that the complement of the coBüchi synthesis problem asks: given a SOCAP $\mathcal{A}$ with parameters X, for all valuations does there exist an infinite run from a given configuration $(q, 0)$, that visits the target set F infinitely many times. Without loss of generality, we assume that the automaton has no parametric tests as they can be simulated using parametric updates and zero tests.

The idea is to check if there exists a reachable "pumpable cycle" containing one of the target states. Formally, given the starting configuration $(q, 0)$, we want to check if we can reach a configuration (q_f, k), where $q_f \in F$ and $k \geq 0$ and then we want to reach q_f again via a pumpable cycle. This means that starting from (q_f, k) we reach the configuration (q_f, k) again or we reach a configuration (q_f, k') with $k' \geq k$ without using zero-test transitions. Note that reachability while avoiding zero tests is the same as reachability in the sub-automaton obtained after deleting all the zero-test transitions. We write $\varphi_{\text{reach}-\text{nt}}$ for the φ_{reach} formula constructed for that sub-automaton as per Proposition 12. The above constraints can be encoded as a formula $\varphi_{\text{Büchi}}(\boldsymbol{x}) = \exists k \exists k' \bigvee_{q_f \in F} \zeta(\boldsymbol{x}, k, k')$ where the subformula ζ is: $(k \leq k') \wedge \varphi_{\text{reach}}^{(q,q_f)}(\boldsymbol{x}, 0, k) \wedge \left(\varphi_{\text{reach}-\text{nt}}^{(q_f,q_f)}(\boldsymbol{x}, k, k') \vee \varphi_{\text{reach}}^{(q_f,q_f)}(\boldsymbol{x}, k, k)\right)$. Finally, the formula $\varphi_{\text{Büchi}}(\boldsymbol{x})$ will look as follows:

$$\exists \boldsymbol{y} \exists k \exists k' \bigvee_{i \in I} \bigwedge_{j \in J_i} (f_j(\boldsymbol{x}) \mid g_j(\boldsymbol{x}, \boldsymbol{y})) \wedge \varphi_i(\boldsymbol{x}) \wedge \psi_i(\boldsymbol{y}, k, k') \wedge \boldsymbol{y} \geq \boldsymbol{0}$$

where, by Proposition 12, the $\varphi_i(\boldsymbol{x})$ are quantifier-free PA formulas over $\boldsymbol{x}$ constructed by grouping all the quantifier-free PA formulas over $\boldsymbol{x}$. Similarly, we can construct $\psi_i(\boldsymbol{y}, k, k')$ by grouping all the quantifier free formulas over $\boldsymbol{y}, k$ and k'. Now, we use the affine change of variables to remove the formulas $\psi_i(\boldsymbol{y}, k, k')$. Technically, the free variables from the subformulas ψ_i will be replaced in all other subformulas by linear polynomials on newly introduced variables $\boldsymbol{z}$. Hence, the final formula $\varphi_{\text{Büchi}}(\boldsymbol{x})$ becomes:

$$\exists \boldsymbol{z} \bigvee_{i \in I'} \bigwedge_{j \in J_i} (f_j(\boldsymbol{x}) \mid g_j(\boldsymbol{x}, \boldsymbol{z})) \wedge \varphi_i(\boldsymbol{x}) \wedge \boldsymbol{z} \geq \boldsymbol{0}.$$

Note that, after using the affine change of variables, the number of $\boldsymbol{z}$ variables is bounded by the number of old existentially quantified variables $(\boldsymbol{y}, k, k')$. However, we have introduced exponentially many new disjuncts.[2]

By construction, for a valuation V there is an infinite V-run in $\mathcal{A}$ from $(q, 0)$ that visits the target states infinitely often iff $\varphi_{\text{Büchi}}(V(\boldsymbol{x}))$ is true. Hence, $\forall \boldsymbol{x}(\boldsymbol{x} < 0 \vee \varphi_{\text{Büchi}}(\boldsymbol{x}))$ precisely encodes the complement of the coBüchi synthesis problem. Also, note that it is a BIL sentence since the subformulas (and in particular the divisibility constraints) come from our usage of Proposition 12. Now, the number of $\boldsymbol{z}$ variables, say m, is bounded by the number of $\boldsymbol{y}$ variables before the affine change of variables which is polynomial with respect to $|\mathcal{A}|$ from Proposition 12. Also, the bitsize of the constants in $\varphi_{\text{Büchi}}$ is polynomial in $|\mathcal{A}|$ though the size of the formula is exponential in $|\mathcal{A}|$. Now, using Lemma 7, we construct an $\forall$PAD sentence $\forall \boldsymbol{x} \forall d \psi(\boldsymbol{x}, d)$ from $\forall \boldsymbol{x}(\boldsymbol{x} < 0 \vee \varphi_{\text{Büchi}}(\boldsymbol{x}))$. By Corollary 9, $\neg\psi$ admits a solution of bitsize bounded by: $\exp(\ln(|\varphi_{\text{Büchi}}|) 2^m \text{poly}(n+1)) = \exp(|\mathcal{A}| \cdot 2^{\text{poly}(|\mathcal{A}|)} \text{poly}(n+1))$, which is doubly exponential in the size of $|\mathcal{A}|$. As in the proof of Theorem 4, a guess-and-check algorithm for $\neg\psi$ gives us the desired **2NEXP** complexity result for the coBüchi synthesis problem. By Lemma 11, the other synthesis problems have the same complexity. ◀

[2] Indeed, because of the bounds on the entries of the matrices and vectors, the cardinality of the set I is exponentially bounded.

In the sequel we sketch our proof of Proposition 12.

4.1 Reachability certificates

We presently recall the notion of reachability certificates from [16]. Fix a SOCAP $\mathcal{A}$ and a valuation V. A flow f in $G_{\mathcal{A}}$ is a *reachability certificate* for two configurations $(q,c),(q',c')$ in $\mathcal{A}$ if there is a V-run from (q,c) to (q',c') that induces a path π such that $f = f_\pi$ and one of the following holds: *(type 1)* f has no positive-weight cycles, *(type 2)* f has no negative-weight cycles, or *(type 3)* f has a positive-weight cycle that can be taken from (q,c) and a negative-weight cycle that can be taken to (q',c').

In the sequel, we will encode the conditions from the following result into a PAD formula so as to accommodate parameters. Intuitively, the proposition states that there is a run from (q,c) to (q',c') if and only if there is one of a special form: a decreasing prefix (type 1), a positive cycle leading to a plateau followed by a negative cycle (type 3), and an increasing suffix (type 2). Each one of the three sub-runs could in fact be an empty run.

▶ **Proposition 14** ([13, Lemma 4.1.14]). *If (q',c') is reachable from (q,c) in a SOCAP with $X = \emptyset$ and without zero tests then there is a run $\rho = \rho_1\rho_2\rho_3$ from (q,c) to (q',c'), where ρ_1, ρ_2, ρ_3, each have a polynomial-size reachability certificate of type 1, 3 and 2, respectively.*

Encoding the certificates. Now, we recall the encoding for the reachability certificates proposed by Lechner [21, 22]. Then, we highlight the changes necessary to obtain the required type of formula. We begin with type-1 and type-3 certificates.

▶ **Lemma 15** (From [22, Lem. 33 and Prop. 36]). *Suppose $\mathcal{A}$ has no zero tests and let $t \in \{1,2,3\}$. Given states q,q', one can construct in deterministic exponential time the existential PAD formula $\Phi_t^{(q,q')}(\boldsymbol{x},a,b)$. Moreover, a valuation V of $X \cup \{a,b\}$ satisfies $\Phi_t^{(q,q')}(\boldsymbol{x},a,b)$ iff there is a V-run from $(q,V(a))$ to $(q',V(b))$ that induces a path π with f_π a type-t reachability certificate.*

The formulas $\Phi_t^{(q,q')}$ from the result above look as follows:

$$\bigvee_{i\in I} \exists \boldsymbol{z} \bigwedge_{j\in J_i} m_j(\boldsymbol{x}) \mid z_j \wedge (m_j(\boldsymbol{x}) > 0 \leftrightarrow z_j > 0) \wedge \varphi_i(\boldsymbol{x}) \wedge \psi_i(\boldsymbol{z},a,b) \wedge \boldsymbol{z} \geq \mathbf{0}$$

where $|I|$ and the size of each disjunct are exponential.[3] Further, all the φ_i and ψ_i are quantifier-free PA formulas and the $m_j(\boldsymbol{x})$ are all either x, $-x$, or $n \in \mathbb{N}_{>0}$.

We observe that the constraint $(m_j(\boldsymbol{x}) > 0 \leftrightarrow z_j > 0)$ regarding when the variables can be 0, can be pushed into a further disjunction over which subset of X is set to 0. In one case the corresponding $m_j(\boldsymbol{x})$'s and z_j's are replaced by 0, in the remaining case we add to φ_i and ψ_i the constraints $z_j > 0$ and $m_j(\boldsymbol{x}) > 0$ respectively. We thus obtain formulas $\Psi_t^{(q,q')}(\boldsymbol{x},a,b)$ with the following properties.

▶ **Lemma 16.** *Suppose $\mathcal{A}$ has no zero tests and let $t \in \{1,2,3\}$. Given states q,q', one can construct in deterministic exponential time a PAD formula $\Psi_t^{(q,q')}(\boldsymbol{x},a,b) = \exists \boldsymbol{y} \bigvee_{i\in I} \varphi_i(\boldsymbol{x},\boldsymbol{y}) \wedge \psi_i(\boldsymbol{y},a,b) \wedge \boldsymbol{y} \geq \mathbf{0}$ s.t. $\forall \boldsymbol{x} \exists \boldsymbol{y} \bigvee_{i\in I} \varphi_i(\boldsymbol{x},\boldsymbol{y}) \wedge \boldsymbol{y} \geq \mathbf{0}$ is a BIL sentence, the $\psi_i(\boldsymbol{y},a,b)$ are quantifier-free PA formulas, and additionally:*

[3] Lechner [21] actually employs a symbolic encoding of the Bellman-Ford algorithm to get polynomial disjuncts in her formula. However, a naïve encoding – while exponential – yields the formula we present here and streamlines its eventual transformation to BIL.

- *a valuation V of $X \cup \{a, b\}$ satisfies $\Psi_t^{(q,q')}$ iff there is a V-run from $(q, V(a))$ to $(q', V(b))$ that induces a path π such that f_π is a type-t reachability certificate,*
- *the bitsize of constants in $\Psi_t^{(q,q')}$ is of polynomial size in $|\mathcal{A}|$,*
- *$|\Psi_t^{(q,q')}|$ is at most exponential with respect to $|\mathcal{A}|$, and*
- *the number of $\boldsymbol{y}$ variables is polynomial with respect to $|\mathcal{A}|$.*

4.2 Putting everything together

In this section, we combine the results from the previous subsection to construct φ_{reach} for Proposition 12. The construction, in full detail, and a formal proof that φ_{reach} enjoys the claimed properties are given in the long version of this paper. First, using Proposition 14 and the lemmas above, we define a formula $\varphi_{\text{reach}-\text{nt}}^{(q,q')}(\boldsymbol{x}, a, b)$ that is satisfied by a valuation V of $X \cup \{a, b\}$ iff there is a V-run from $(q, V(a))$ to $(q', V(b))$ without any zero-test transitions. To do so, we use formulas for the sub-automaton obtained by removing from $\mathcal{A}$ all zero-test transitions. Then, the formula $\varphi_{\text{reach}}^{(q,q')}(\boldsymbol{x}, a, b)$ expressing general reachability can be defined by taking a disjunction over all orderings on the zero tests. In other words, for each enumeration of zero-test transitions we take the conjunction of the intermediate $\varphi_{\text{reach}-\text{nt}}$ formulas as well as $\varphi_{\text{reach}-\text{nt}}$ formulas from the initial configuration and to the final one.

Recall that for any LTL formula φ we can construct a *universal coBüchi automaton* of exponential size in $|\varphi|$ [2, 19]. (A universal coBüchi automaton accepts a word w if all of its infinite runs on w visit F only finitely often. Technically, one can construct such an automaton for φ by constructing a Büchi automaton for $\neg\varphi$ and "syntactically complementing" its acceptance condition.) By considering the product of this universal coBüchi automaton and the given SOCAP, the LTL synthesis problem reduces to coBüchi synthesis.

▶ **Corollary 17.** *The LTL synthesis problem for succinct one-counter automata with parameters is decidable in* **3NEXP**.

5 One-Counter Automata with Parametric Tests

In this section, we introduce a subclass of SOCAP where only the tests are parametric. The updates are non-parametric and assumed to be given in unary. Formally, *OCA with parametric tests (OCAPT)* allow for constant updates of the form $\{+a : a \in \{-1, 0, 1\}\}$ and zero and parametric tests. However, $PU = \emptyset$.

We consider the synthesis problems for OCAPT. Our main result in this section are better complexity upper bounds than for general SOCAP. Lemma 11 states that all the synthesis problems reduce to the coBüchi synthesis problem for SOCAP. Importantly, in the construction used to prove Lemma 11, we do not introduce parametric updates. Hence, the reduction also holds for OCAPT. This allows us to focus on the coBüchi synthesis problem – the upper bounds for the other synthesis problems follow.

▶ **Theorem 18.** *The coBüchi, Büchi and safety synthesis problems for OCAPT are in* **PSPACE***; the reachability synthesis problem, in* $\mathbf{NP}^{\mathbf{coNP}} = \mathbf{NP}^{\mathbf{NP}}$.

To prove the theorem, we follow an idea from [4] to encode parameter valuations of OCAPT into words accepted by an alternating two-way automaton. Below, we give the proof of the theorem assuming some auxiliary results that will be established in the following subsections.

Proof. In Proposition 21, we reduce the coBüchi synthesis problem to the non-emptiness problem for alternating two-way automata. Hence, we get the **PSPACE** upper bound. Since the Büchi and the safety synthesis problems reduce to the coBüchi one (using Lemma 11) in polynomial time, these are also in **PSPACE**.

Next, we improve the complexity upper bound for the reachability synthesis problem from **PSPACE** to $\mathbf{NP^{NP}}$. In Lemma 22 we will prove that if there is a valuation V of the parameters such that all infinite V-runs reach F then we can assume that V assigns to each $x \in X$ a value at most exponential. Hence, we can guess their binary encoding and store it using a polynomial number of bits. Once we have guessed V and replaced all the x_i by $V(x_i)$, we obtain a non-parametric one counter automata $\mathcal{A}'$ with $X = \emptyset$ and we ask whether all infinite runs reach F. We will see in Proposition 23 that this problem is in **coNP**. The claimed complexity upper bound for the reachability synthesis problem follows. ◀

Using a similar idea to Corollary 17, we reduce the LTL synthesis problem to the coBüchi one and we obtain the following.

▶ **Corollary 19.** *The LTL synthesis problem for OCAPT is in* **EXPSPACE**.

5.1 Alternating two-way automata

Given a finite set Y, we denote by $\mathbb{B}^+(Y)$ the set of positive Boolean formulas over Y, including true and false. A subset $Y' \subseteq Y$ satisfies $\beta \in \mathbb{B}^+(Y)$, written $Y' \models \beta$, if β is evaluated to true when substituting true for every element in Y', and false for every element in $Y \setminus Y'$. In particular, we have $\emptyset \models \text{true}$.

We can now define an *alternating two-way automaton* (A2A, for short) as a tuple $\mathcal{T} = (S, \Sigma, s_{in}, \Delta, S_f)$, where S is a finite set of states, Σ is a finite alphabet, $s_{in} \in S$ is the initial state, $S_f \subseteq S$ is the set of accepting states, and $\Delta \subseteq S \times (\Sigma \cup \{\text{first?}\}) \times \mathbb{B}^+(S \times \{+1, 0, -1\})$ is the finite transition relation. The $+1$ intuitively means that the head moves to the right; -1, that the head moves to the left; 0, that it stays at the current position. Furthermore, transitions are labelled by Boolean formulas over successors which determine whether the current run branches off in a non-deterministic or a universal fashion.

A *run (tree)* γ of $\mathcal{T}$ on an infinite word $w = a_0 a_1 \cdots \in \Sigma^w$ from $n \in \mathbb{N}$ is a (possibly infinite) rooted tree whose vertices are labelled with elements in $S \times \mathbb{N}$ and such that it satisfies the following properties. The root of γ is labelled by (s_{in}, n). Moreover, for every vertex labelled by (s, m) with $k \in \mathbb{N}$ children labelled by $(s_1, n_1), \ldots, (s_k, n_k)$, there is a transition $(s, \sigma, \beta) \in \Delta$ such that, the set $\{(s_1, n_1 - m), \ldots, (s_k, n_k - m)\} \subseteq S \times \{+1, 0, -1\}$ satisfies β. Further $\sigma = \text{first?}$ implies $m = 0$, and $\sigma \in \Sigma$ implies $a_m = \sigma$.

A run is *accepting* if all of its infinite branches contain infinitely many labels from $S_f \times \mathbb{N}$. The *language of* $\mathcal{T}$ is $L(\mathcal{T}) \stackrel{\text{def}}{=} \{w \in \Sigma^\omega \mid \exists \text{ an accepting run of } \mathcal{T} \text{ on } w \text{ from } 0\}$. The *non-emptiness problem for A2As* asks, given an A2A $\mathcal{T}$ and $n \in \mathbb{N}$, whether $L(\mathcal{T}) \neq \emptyset$.

▶ **Proposition 20** (From [31])**.** *Language emptiness for A2As is in* **PSPACE**.

In what follows, from a given OCAPT $\mathcal{A}$ we will build an A2A $\mathcal{T}$ such that $\mathcal{T}$ accepts precisely those words which correspond to a valuation V of X under which all infinite runs satisfy the coBüchi condition. Hence, the corresponding synthesis problem for $\mathcal{A}$ reduces to checking non-emptiness of $\mathcal{T}$.

5.2 Transformation to alternating two-way automata

Following [4], we encode a valuation $V : X \to \mathbb{N}$ as an infinite *parameter word* $w = a_0 a_1 a_2 \ldots$ over the alphabet $\Sigma = X \cup \{\square\}$ such that $a_0 = \square$ and, for every $x \in X$, there is exactly one position $i \in \mathbb{N}$ such that $a_i = x$. We write $w(i)$ to denote its prefix $a_0 a_1 \ldots a_i$ up to the letter a_i. By $|w(i)|_\square$, we denote the number of occurrences of $\square$ in $a_1 \ldots a_i$. (Note that we ignore a_0.) Then, a parameter word w determines a valuation $V_w : x \mapsto |w(i)|_\square$ where $a_i = x$. We denote the set of all parameter words over X by W_X.

From a given OCAPT $\mathcal{A} = (Q, T, \delta, X)$, a starting configuration $(q_0, 0)$ and a set of target states F, we will now construct an A2A $\mathcal{T} = (S, \Sigma, s_{in}, \Delta, S_f)$ that accepts words $w \in W_X$ such that, under the valuation $V = V_w$, all infinite runs from $(q_0, 0)$ visit F only finitely many times.

▶ **Proposition 21.** *For all OCAPT $\mathcal{A}$ there is an A2A $\mathcal{T}$ with $|\mathcal{T}| = |\mathcal{A}|^{\mathcal{O}(1)}$ and $w \in L(\mathcal{T})$ if and only if all infinite V_w-runs of $\mathcal{A}$ starting from $(q_0, 0)$ visit F only finitely many times.*

The construction is based on the A2A built in [4], although we make more extensive use of the alternating semantics of the automaton. To capture the coBüchi condition, we simulate a *safety copy* with the target states as "non-accepting sink" (states having a self-loop and no other outgoing transitions) inside $\mathcal{T}$. Simulated accepting runs of $\mathcal{A}$ can "choose" to enter said safety copy once they are sure to never visit F again. Hence, for every state q in $\mathcal{A}$, we have two copies of the state in $\mathcal{T}$: $q' \in S$ representing q normally and $q'' \in S$ representing q from the safety copy. Now the idea is to encode runs of $\mathcal{A}$ as branches of run trees of $\mathcal{T}$ on parameter words w by letting sub-trees t whose root is labelled with (q', i) or (q'', i) correspond to the configuration $(q, |w(i)|_\square)$ of $\mathcal{A}$. If t is accepting, it will serve as a witness that all infinite runs of $\mathcal{A}$ from $(q, |w(i)|_\square)$ satisfy the coBüchi condition.

We present the overview of the construction below with some intuitions. A detailed proof of Proposition 21 is given in the long version of the paper.

- The constructed A2A $\mathcal{T}$ for the given $\mathcal{A}$ is such that for every $q \in Q$, there are two copies $q', q'' \in S$ as mentioned earlier. We also introduce new states in $\mathcal{T}$ as required.
- The A2A includes a sub-A2A that verifies that the input word is a valid parameter word. For every x_i, a branch checks that it appears precisely once in the parameter word.
- From a run sub-tree whose root is labelled with (q', i) or (q'', i), the A2A verifies that all runs of $\mathcal{A}$ from $(q, |w(i)|_\square)$ visit F only finitely many times. To do this, for all transitions δ of the form $q \xrightarrow{op} r$ in $\mathcal{A}$, we create a sub-A2A $\mathcal{T}^{\delta}_{\text{sub}}$ using copies of sub-A2As. For each such transition, one of two cases should hold: either the transition cannot be simulated (because of a zero test or a decrement from zero), or the transition can indeed be simulated. For the former, we add a *violation branch* to check that it is indeed the case; for the latter, a *validation branch* checks the transition can be simulated and a *simulation branch* reaches the next vertex with the updated counter value. Now if the root vertex is of the form (q', i) then the *simulation branch* could reach a vertex labelled with r' or with r'' – with the idea being that $\mathcal{T}$ can choose to move to the safety copy or to stay in the "normal" copy of $\mathcal{A}$. If the root vertex is of the form (q'', i), the simulation branch can only reach the vertex labelled with r'' with the updated counter value.
- We obtain the global A2A $\mathcal{T}$ by connecting sub-A2As. To ensure that all runs of $\mathcal{A}$ are simulated, we have the global transition relation Δ be a conjunction of that of the sub-A2As which start at the same state $q \in \{p', p''\}$ for some $p \in Q$. For instance, let $\delta_1 = (q, op_1, q_1)$ and $\delta_2 = (q, op_2, q_2)$ be transitions of $\mathcal{A}$. The constructed sub-A2As $\mathcal{T}^{\delta_1}_{\text{sub}}, \mathcal{T}^{\delta_2}_{\text{sub}}$ will contain transitions $(q, \square, \beta_1) \in \Delta_1, (q, \square, \beta_2) \in \Delta_2$ respectively. In $\mathcal{T}$, we instead have $(q, \square, \beta_1 \wedge \beta_2) \in \Delta$.
- Finally, the accepting states are chosen as follows: For every $q \in Q \setminus F$, we set q'' as accepting in $\mathcal{T}$. The idea is that if a run in $\mathcal{A}$ satisfies the coBüchi condition then, after some point, it stops visiting target states. In $\mathcal{T}$, the simulated run can choose to move to the safety copy at that point and loop inside it forever thus becoming an accepting branch. On the other hand, if a run does not satisfy the condition, its simulated version cannot stay within the safety copy. (Rather, it will reach the non-accepting sink states.) Also, the violation and the validation branches ensure that the operations along the runs

have been simulated properly inside $\mathcal{T}$. It follows that $\mathcal{T}$ accepts precisely those words whose run-tree contains a simulation branch where states from F have been visited only finitely many times.

5.3 An upper bound for reachability synthesis of OCAPT

Following [4], we now sketch a guess-and-check procedure using the fact that Proposition 21 implies a sufficient bound on valuations satisfying the reachability synthesis problem. Recall that, the reachability synthesis problem asks whether all infinite runs reach a target state.

▶ **Lemma 22** (Adapted from [4, Lemma 3.5]). *If there is a valuation V such that all infinite V-runs of $\mathcal{A}$ reach F, there is a valuation V' such that $V'(x) = \exp(|\mathcal{A}|^{\mathcal{O}(1)})$ for all $x \in X$ and all infinite V'-runs of $\mathcal{A}$ reach F.*

It remains to give an algorithm to verify that in the resulting non-parametric OCA (after substituting parameters with their values), all infinite runs from $(q_0, 0)$ reach F.

▶ **Proposition 23.** *Checking whether all infinite runs from $(q_0, 0)$ reach a target state in a non-parametric one-counter automata is* **coNP***-complete.*

Before proving the claim above, we first recall a useful lemma from [21].

A *path* $\pi = q_0 q_1 \dots q_n$ in $G_{\mathcal{A}}$ is a *cycle* if $q_0 = q_n$. We say the cycle is *simple* if no state (besides q_0) is repeated. A cycle *starts from a zero test* if $\delta(q_0, q_1)$ is "$= 0$". A *zero-test-free cycle* is a cycle where no $\delta(q_i, q_{i+1})$ is a zero test. We define a *pumpable cycle* as being a zero-test-free cycle such that for all runs $\rho = (q_0, c_0) \dots (q_n, c_n)$ lifted from π we have $c_n \geq c_0$, i.e., the effect of the cycle is non-negative.

▶ **Lemma 24** (From [21]). *Let $\mathcal{A}$ be a SOCA with an infinite run that does not reach F. Then, there is an infinite run of $\mathcal{A}$ which does not reach F such that it induces a path $\pi_0 \cdot \pi_1^\omega$, where π_1 either starts from a zero test or it is a simple pumpable cycle.*

Sketch of proof of Proposition 23. We want to check whether all infinite runs starting from $(q_0, 0)$ reach F. Lemma 24 shows two conditions, one of which must hold if there is an infinite run that does not reach F. Note that both conditions are in fact reachability properties: a path to a cycle that starts from a zero test or to a simple pumpable cycle.

For the first condition, making the reachability-query instances concrete requires configuration a $(q, 0)$ and a state q' such that $\delta(q, q')$ is a zero test. Both can be guessed and stored in polynomial time and space. For the other condition, we can assume that π_0 does not have any simple pumpable cycle. It follows that every cycle in π_0 has a zero test or has a negative effect. Let $W_{\max}$ be the sum of all the positive updates in $\mathcal{A}$. Note that the counter value cannot exceed $W_{\max}$ along any run lifted from π_0 starting from $(q_0, 0)$. Further, since π_1 is a simple cycle the same holds for $2W_{\max}$ for runs lifted from $\pi_0\pi_1$. Hence, we can guess and store in polynomial time and space the two configurations (q, c) and (q, c') required to make the reachability-query instances concrete.

Since the reachability problem for non-parametric SOCAP is in **NP** [16], we can guess which condition will hold and guess the polynomial-time verifiable certificates. This implies the problem is in **coNP**.

For the lower bound, one can easily give a reduction from the complement of the SubsetSum problem, which is **NP**-complete [11]. The idea is similar to reductions used in the literature to prove **NP**-hardness for reachability in SOCAP. In the long version of the paper, the reduction is given in full detail. ◀

6 Conclusion

We have clarified the decidability status of synthesis problems for OCA with parameters and shown that, for several fixed ω-regular properties, they are in **2NEXP**. If the parameters only appear on tests, then we further showed that those synthesis problems are in **PSPACE**. Whether our new upper bounds are tight remains an open problem: neither our **coNP**-hardness result for the reachability synthesis problem nor the **PSPACE** and $\mathbf{NP}^{\mathbf{NP}}$ hardness results known [32, 21, 22] for other synthesis problems (see Table 1) match them.

We believe the BIL fragment will find uses beyond the synthesis problems for OCA with parameters: e.g. it might imply decidability of the software-verification problems that motivated the study of $\forall\exists_R\mathrm{PAD}^+$ in [6], or larger classes of quadratic string equations than the ones solvable by reduction to $\exists$PAD [25]. While we have shown BIL is decidable in **2NEXP**, the best known lower bound is the trivial **coNP**-hardness that follows from encoding the complement of the SubsetSum problem. (Note that BIL does not syntactically include the Π_1-fragment of PA so it does not inherit hardness from the results in [14].) Additionally, it would be interesting to reduce validity of BIL sentences to a synthesis problem. Following [16], one can easily establish a reduction to this effect for sentences of the form: $\forall\boldsymbol{x}\exists\boldsymbol{y}\bigvee_{i\in I} f_i(\boldsymbol{x}) \mid g(\boldsymbol{x},\boldsymbol{y}) \wedge f_i(\boldsymbol{x}) > 0 \wedge \varphi_i(\boldsymbol{x}) \wedge \boldsymbol{y} \geq \mathbf{0}$ but full BIL still evades us.

References

1 Frances E Allen. Control flow analysis. *ACM Sigplan Notices*, 5(7):1–19, 1970.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

3 A. P. Beltyukov. Decidability of the universal theory of natural numbers with addition and divisibility. *Journal of Soviet Mathematics*, 14(5):1436–1444, November 1980. doi:10.1007/BF01693974.

4 Benedikt Bollig, Karin Quaas, and Arnaud Sangnier. The complexity of flat freeze LTL. *Logical Methods in Computer Science*, 15(3), 2019. doi:10.23638/LMCS-15(3:33)2019.

5 Ahmed Bouajjani, Marius Bozga, Peter Habermehl, Radu Iosif, Pierre Moro, and Tomás Vojnar. Programs with lists are counter automata. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 517–531, 2006. doi:10.1007/11817963_47.

6 Marius Bozga and Radu Iosif. On decidability within the arithmetic of addition and divisibility. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3441 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2005. doi:10.1007/978-3-540-31982-5_27.

7 Daniel Bundala and Joël Ouaknine. On parametric timed automata and one-counter machines. *Inf. Comput.*, 253:272–303, 2017. doi:10.1016/j.ic.2016.07.011.

8 Cristiana Chitic and Daniela Rosu. On validation of XML streams using finite state machines. In *Proceedings of the Seventh International Workshop on the Web and Databases, WebDB 2004, June 17-18, 2004, Maison de la Chimie, Paris, France, Colocated with ACM SIGMOD/PODS 2004*, pages 85–90, 2004. doi:10.1145/1017074.1017096.

9 Patrick Cousot and Radhia Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation. In Javier Esparza, Bernd Spanfelner, and Orna Grumberg, editors, *Logics and Languages for Reliability and Security*, volume 25 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 1–29. IOS Press, 2010. doi:10.3233/978-1-60750-100-8-1.

10 Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proofs that count. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 151–164. ACM, 2014. doi:10.1145/2535838.2535885.

11 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

12 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. doi:10.1007/978-3-642-14162-1_48.

13 Christoph Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, 2012.

14 Christoph Haase. Subclasses of presburger arithmetic and the weak EXP hierarchy. 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 47:1–47:10. ACM, 2014. doi:10.1145/2603088.2603092.

15 Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. URL: https://dl.acm.org/citation.cfm?id=3242964.

16 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, pages 369–383, 2009. doi:10.1007/978-3-642-04081-8_25.

17 Oscar H. Ibarra, Tao Jiang, Nicholas Q. Trân, and Hui Wang. New decidability results concerning two-way counter machines. *SIAM J. Comput.*, 24(1):123–137, 1995. doi:10.1137/S0097539792240625.

18 Oscar H. Ibarra, Jianwen Su, Zhe Dang, Tevfik Bultan, and Richard A. Kemmerer. Counter machines and verification problems. *Theor. Comput. Sci.*, 289(1):165–189, 2002. doi:10.1016/S0304-3975(01)00268-7.

19 Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 531–542. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.66.

20 Mahler Kurt. On the Chinese remainder theorem. *Mathematische Nachrichten*, 18(1-6):120–122, 1958. doi:10.1002/mana.19580180112.

21 Antonia Lechner. Synthesis problems for one-counter automata. In Mikolaj Bojanczyk, Slawomir Lasota, and Igor Potapov, editors, *Reachability Problems - 9th International Workshop, RP 2015, Warsaw, Poland, September 21-23, 2015, Proceedings*, volume 9328 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2015. doi:10.1007/978-3-319-24537-9_9.

22 Antonia Lechner. *Extensions of Presburger arithmetic and model checking one-counter automata*. PhD thesis, University of Oxford, 2016.

23 Antonia Lechner, Joël Ouaknine, and James Worrell. On the complexity of linear arithmetic with divisibility. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 667–676, 2015. doi:10.1109/LICS.2015.67.

24 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. doi:10.1007/978-3-030-62822-2_6.

25 Anthony W. Lin and Rupak Majumdar. Quadratic word equations with length constraints, counter systems, and presburger arithmetic with divisibility. 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 352–369. Springer, 2018. doi:10.1007/978-3-030-01090-4_21.

26 Leonard Lipshitz. The diophantine problem for addition and divisibility. *Transactions of the American Mathematical Society*, pages 271–283, 1978.

27 Leonard Lipshitz. Some remarks on the diophantine problem for addition and divisibility. *Bull. Soc. Math. Belg. Sér. B*, 33(1):41–52, 1981.

28 Ju V Matijasevic. Enumerable sets are diophantine. In *Soviet Math. Dokl.*, volume 11, pages 354–358, 1970.

29 Marvin L. Minsky. Recursive unsolvability of post's problem of "tag" and other topics in theory of turing machines. *Annals of Mathematics*, 74(3):437–455, 1961. URL: `http://www.jstor.org/stable/1970290`.

30 Julia Robinson. Definability and decision problems in arithmetic. *The Journal of Symbolic Logic*, 14(2):98–114, 1949.

31 Olivier Serre. Parity games played on transition graphs of one-counter processes. In Luca Aceto and Anna Ingólfsdóttir, editors, *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*, volume 3921 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2006. `doi:10.1007/11690634_23`.

32 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. `doi:10.1145/3828.3837`.

33 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, 1978.

First-Order Logic with Connectivity Operators

Nicole Schirrmacher
University of Bremen, Germany

Sebastian Siebertz
University of Bremen, Germany

Alexandre Vigny
University of Bremen, Germany

Abstract

First-order logic (FO) can express many algorithmic problems on graphs, such as the independent set and dominating set problem parameterized by solution size. On the other hand, FO cannot express the very simple algorithmic question whether two vertices are connected. We enrich FO with connectivity predicates that are tailored to express algorithmic graph properties that are commonly studied in parameterized algorithmics. By adding the atomic predicates $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$ that hold true in a graph if there exists a path between (the valuations of) x and y after (the valuations of) $z_1, \ldots, z_k$ have been deleted, we obtain *separator logic* $\mathrm{FO} + \mathrm{conn}$. We show that separator logic can express many interesting problems such as the feedback vertex set problem and elimination distance problems to first-order definable classes. Denote by $\mathrm{FO} + \mathrm{conn}_k$ the fragment of separator logic that is restricted to connectivity predicates with at most $k + 2$ variables (that is, at most k deletions). We show that $\mathrm{FO} + \mathrm{conn}_{k+1}$ is strictly more expressive than $\mathrm{FO} + \mathrm{conn}_k$ for all $k \geq 0$. We then study the limitations of separator logic and prove that it cannot express planarity, and, in particular, not the disjoint paths problem. We obtain the stronger *disjoint-paths logic* $\mathrm{FO} + \mathrm{DP}$ by adding the atomic predicates $\text{disjoint-paths}_k[(x_1, y_1), \ldots, (x_k, y_k)]$ that evaluate to true if there are internally vertex-disjoint paths between (the valuations of) x_i and y_i for all $1 \leq i \leq k$. Disjoint-paths logic can express the disjoint paths problem, the problem of (topological) minor containment, the problem of hitting (topological) minors, and many more. Again we show that the fragments $\mathrm{FO} + \mathrm{DP}_k$ that use predicates for at most k disjoint paths form a strict hierarchy of expressiveness. Finally, we compare the expressive power of the new logics with that of transitive-closure logics and monadic second-order logic.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Mathematics of computing → Combinatorics

Keywords and phrases First-order logic, graph theory, connectivity

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.34

Funding This paper is a part of the ANR-DFG project *Unifying Theories for Multivariate Algorithms* (UTMA), which has received funding from the German Research Foundation (DFG) with grant agreement No 446200270.

Acknowledgements We thank Mikołaj Bojańczyk for fruitful discussions. He independently studied $\mathrm{FO} + \mathrm{conn}$ and suggested the name *separator logic* for this logic. We also thank Michał Pilipczuk and Szymon Toruńczyk for fruitful discussions.

1 Introduction

Logic provides a very elegant way of formally describing computational problems. Fagin's celebrated result in 1974 [11] established that existential second-order logic captures the complexity class NP. Fagin thereby provided a machine-independent characterization of

a complexity class and initiated the field of descriptive complexity theory. Many other complexity classes were later characterized by logics in this theory. Today it remains one of the major open problems whether there exists a logic capturing PTime.

In 1990 Courcelle proved that every graph property definable in monadic second-order logic (MSO) can be decided in linear time on graphs of bounded treewidth [7]. This theorem has a much more algorithmic (rather than a complexity-theoretic) flavor, in the sense that, from a logical description of a problem, it derives an algorithmic approach on how to solve it on certain graph classes. Grohe in his seminal survey coined the term *algorithmic meta-theorem* for such theorems that provide general conditions on a problem and on the input instances that, when satisfied, imply the existence of an efficient algorithm for the problem [17]. Courcelle's theorem for MSO was extended to graph classes with bounded cliquewidth [8] and it is known that these are essentially the most general graph classes on which efficient MSO model-checking [15, 21] is possible. MSO is a powerful logic that can express many important algorithmic properties on graphs. With quantification over edges, we can for example express the existence of a Hamiltonian path, the existence of a fixed minor or topological minor, the disjoint paths problem, and many deletion problems. For a property Π, the task in the Π-deletion problem is to find in a given graph G a minimum-size subset S of $V(G)$ such that the graph $G - S$ obtained from G by removing S has the property Π. Important examples of Π-deletion problems are the feedback vertex set problem, the odd cycle transversal problem, or the problem of hitting all minors or topological minors from a given list $\mathcal{F}$. Also, many elimination distance problems recently studied [5] in parameterized algorithmics can be expressed in MSO. However, as we have seen, this expressiveness comes at the price of algorithmic intractability already on very restricted graph classes. This cannot be a surprise as e.g. the Hamiltonian path problem is NP-complete already on planar graphs of maximum degree 3 [6].

First-order logic (FO) is much weaker than MSO and consequently, the model-checking problem can be solved efficiently on much more general graph classes. FO model-checking is fixed-parameter tractable on a subgraph-closed class $\mathscr{C}$ if and only if $\mathscr{C}$ is nowhere dense [18] and a recent breakthrough result showed that it is fixed-parameter tractable on a class $\mathscr{C}$ of ordered graphs if and only if $\mathscr{C}$ has bounded twin-width [3]. FO is weaker than MSO but it can still express many important problems such as the independent set problem and dominating set problem parameterized by solution size, the Steiner tree problem parameterized by the number of Steiner vertices, and many more problems. On the other hand, first-order logic cannot even express the algorithmically extremely simple problem of whether a graph is connected. Also, the other algorithmic problems mentioned before are not expressible in FO, even though some of them are fixed-parameter tractable on general graphs. For example, we can efficiently test for a fixed minor or topological minor and solve the disjoint paths problem [26]. Many Π-deletion problems are fixed-parameter tractable, see e.g. [9, 14, 25], as well as many elimination distance problems [1, 12].

The fact that first-order logic can only express local properties is classically addressed by adding transitive-closure or fixed-point operators, see e.g. [10, 16, 22]. Unfortunately, this again comes at the price of intractable model-checking for very restricted graph classes. For example, even the model-checking problem for the very restricted monadic transitive-closure logic TC^1 studied by Grohe [17], is AW[$\star$]-hard on planar graphs of maximum degree at most 3 [17, Theorem 7.3]. Also, these logics fall short of being able to express all of the above mentioned algorithmic graph problems studied in recent parameterized algorithmics.

This motivates our present work in which we enrich first-order logic with basic connectivity predicates. The extensions are tailored to express algorithmic graph properties that are studied in recent parameterized algorithmics. We can add the atomic predicate $\mathrm{conn}_0(x, y)$

that evaluates to true on a graph G if (the valuations of) x and y are connected in G. This predicate easily generalizes to directed graphs but for simplicity, we work with undirected graphs only. Of course, with this predicate we can express connectivity of graphs, however, it falls short of expressing other interesting properties, e.g. it cannot express that a graph is acyclic. We hence introduce more general predicates $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$, parameterized by a number k, that evaluate to true on a graph G if (the valuations of) x and y are connected in G once (the valuations of) $z_1, \ldots, z_k$ have been deleted. The interplay of these predicates with the usual nesting of first-order quantification makes the new logic FO + conn already quite powerful. For example, we can express simple properties such as 2-connectivity by $\forall z \forall x \forall y \big(x \neq z \wedge y \neq z \rightarrow \mathrm{conn}_1(x, y, z)\big)$. We can also express many deletion problems, such as the feedback vertex set problem, and the elimination distance to bounded degree, and more generally, elimination distance to any fixed first-order property.

We also point to the work of Mikołaj Bojańczyk [2], who independently introduced FO + conn and proposed the name *separator logic*. He studied a variant of star-free expressions for graphs and showed that these expressions exactly correspond to separator logic. We follow his suggestion and thank Mikołaj for the discussion on separator logic.

In Section 3 we study the expressive power of separator logic. We give examples on properties expressible with separator logic as well as proofs that certain properties, such as planarity and in particular the disjoint paths problem, are not expressible in separator logic. We show that $(k + 2)$-connectivity of a graph cannot be expressed with only conn_k predicates and conclude that the restricted use of these predicates induces a natural hierarchy of expressiveness.

Using the notion of *block decompositions* together with known model-checking results, one can show that model-checking for formulas using only conn_1 predicates is fixed-parameter tractable on nowhere dense classes of graphs. Hence, we can evaluate very simple connectivity queries in formulas without an increase in the complexity of the model-checking problem on subgraph-closed graph classes. On the other hand, when we allow conn_2 predicates, there are some simple graph classes that do not exclude a topological minor, and on which model-checking becomes AW[⋆]-hard. In this paper, we do not go into the details of model-checking, but in a companion paper [24], we prove that in fact model-checking for FO + conn is fixed-parameter tractable on graph classes that exclude a topological minor.

The fact that planarity and the disjoint paths problem cannot be expressed in separator logic motivates us to define an even stronger logic that can express these properties. The atomic predicate $\mathrm{disjoint\text{-}paths}_k[(x_1, y_1), \ldots, (x_k, y_k)]$ evaluates to true if and only if there are internally vertex-disjoint paths between (the valuations of) x_i and y_i for all $1 \leq i \leq k$. Connectivity of x and y can be tested by $\mathrm{disjoint\text{-}paths}_1[(x, y)]$. More generally, the so obtained *disjoint-paths logic* FO + DP strictly extends separator logic. With this more powerful logic, we can test if a graph contains a fixed minor or topological minor, and in particular, test for planarity. In combination with first-order quantification, we can also express many Π-deletion problems such as the problem of hitting all minors or topological minors from a given list $\mathcal{F}$. On the other hand, we cannot express the odd cycle transversal problem, as we cannot even express bipartiteness of a graph. We study the expressive power of FO + DP in Section 4. Among other results, we prove that again an increase in the number of disjoint paths in the predicates leads to an increase in expressive power.

Note that while it would be desirable to be able to express bipartiteness, which is equivalent to 2-colorability, it is not desirable to express general colorability problems, as we aim for logics that are tractable on planar graphs and beyond, while the 3-colorability problem is NP-complete on planar graphs. This example shows again that it is a delicate

balance between expressiveness and tractability and it will be a challenging and highly interesting problem in future work to find the right set of predicates to express even more algorithmic graph properties while at the same time having tractable model-checking. Until now the complexity of the model-checking problem for FO + DP has remained elusive and will be a very interesting problem in future work.

We conclude the paper in Section 5 with a comparison between the newly introduced logics and more established ones, like MSO and transitive-closure logics.

2 Preliminaries

Graphs. In this paper we deal with finite and simple undirected graphs. Let G be a graph. We write $V(G)$ for the vertex set of G and $E(G)$ for its edge set. For a set $X \subseteq V(G)$ we write $G[X]$ for the subgraph of G induced by X and $G - X$ for the subgraph induced by $V(G) \setminus X$. For a singleton set $\{v\}$ we write $G - v$ instead of $G - \{v\}$. A *path* P in G is a subgraph on distinct vertices $v_1, \ldots, v_t$ with $\{v_i, v_{i+1}\} \in E(P)$ for all $1 \leq i < t$ and a path P is said to *connect* its endpoints v_1 and v_t. Two paths are *internally vertex-disjoint* if and only if every vertex that appears in both paths is an end point of both paths. The graph G is *connected* if every two of its vertices are connected by a path. It is *k-connected* if G has more than k vertices and $G - X$ is connected for every subset $X \subseteq V(G)$ of size strictly smaller than k. A *cycle* C in G is a subgraph on distinct vertices $v_1, \ldots, v_t$, $t \geq 3$, with $\{v_t, v_1\} \in E(C)$ and $\{v_i, v_{i+1}\} \in E(C)$ for all $1 \leq i < t$. An acyclic graph is a *forest* and a connected acyclic graph is a *tree*.

A graph H is a *minor* of G, denoted $H \preccurlyeq G$, if for all $v \in V(H)$ there are pairwise vertex-disjoint connected subgraphs G_v of G such that whenever $\{u, v\} \in E(H)$, then there are $x \in V(G_u)$ and $y \in V(G_v)$ with $\{x, y\} \in E(G)$. The graph H is a *topological minor* of G, denoted $H \preccurlyeq^{top} G$, if for all $v \in V(H)$ there is a distinct vertex x_v in G and for all $\{u, v\} \in E(H)$ there are internally vertex-disjoint paths P_{uv} in G with endpoints x_u and x_v. A graph is *planar* if and only if it does not contain K_5, the complete graph on 5 vertices, and $K_{3,3}$, the complete bipartite graph with two partitions of size 3, as a minor.

Logic. In this work we deal with structures over purely relational *signatures*. A (purely relational) signature is a collection of relation symbols, each with an associated arity. Let σ be a signature. A *σ-structure* $\mathfrak{A}$ consists of a non-empty set A, the universe of $\mathfrak{A}$, together with an interpretation of each k-ary relation symbol $R \in \sigma$ as a k-ary relation $R^{\mathfrak{A}} \subseteq A^k$. For a subset $X \subseteq A$ we write $\mathfrak{A}[X]$ for the substructure induced by X. A *partial isomorphism* between σ-structures $\mathfrak{A}$ and $\mathfrak{B}$ is an isomorphism between $\mathfrak{A}[X]$ and $\mathfrak{B}[Y]$ for some subset $X \subseteq A$ of the universe A of $\mathfrak{A}$ and some subset $Y \subseteq B$ of the universe B of $\mathfrak{B}$.

We assume an infinite supply VAR of variables. First-order formulas are built from the atomic formulas $x = y$, where x and y are variables, and $R(x_1, \ldots, x_k)$, where $R \in \sigma$ is a k-ary relation symbol and $x_1, \ldots, x_k$ are variables, by closing under the Boolean connectives $\neg$, $\wedge$ and $\vee$, and by existential and universal quantification $\exists x$ and $\forall x$. A variable x not in the scope of a quantifier is a *free variable*. A formula without free variables is a *sentence*. The *quantifier rank* $\mathrm{qr}(\varphi)$ of a formula φ is the maximum nesting depth of quantifiers in φ. We write $\mathrm{FO}_\sigma[q]$ for the set of all FO σ-formulas of quantifier rank at most q, or simply $\mathrm{FO}[q]$ if σ is clear from the context. A formula without quantifiers is called *quantifier-free*.

If $\mathfrak{A}$ is a σ-structure with universe A, then an *assignment* of the variables in $\mathfrak{A}$ is a mapping $\bar{a} : \textsc{Var} \to A$. We use the standard notation $(\mathfrak{A}, \bar{a}) \models \varphi(\bar{x})$ or $\mathfrak{A} \models \varphi(\bar{a})$ to indicate that φ is satisfied in $\mathfrak{A}$ when the free variables $\bar{x}$ of φ have been assigned by $\bar{a}$. We refer e.g. to the textbook [22] for more background on first-order logic.

3 Separator logic

In this section, we study the expressive power of separator logic FO + conn. Formally, we assume that σ is a signature that does not contain any of the relation symbols conn_k for all $k \geq 0$, and that it does contain a binary relation symbol E, representing an edge relation. We assume that E is always interpreted as an irreflexive and symmetric relation and connectivity will always refer to this relation. We let $\sigma + \text{conn} := \sigma \cup \{\text{conn}_k : k \geq 0\}$, where each conn_k is a $(k+2)$-ary relation symbol.

▶ **Definition 3.1.** *The formulas of* $(\text{FO} + \text{conn})[\sigma]$ *are the formulas of* $\text{FO}[\sigma + \text{conn}]$. *We usually simply write* $\text{FO} + \text{conn}$, *when* σ *is understood from the context.*

For a σ*-structure* $\mathfrak{A}$, *an assignment* $\bar{a}$ *and an* $\text{FO} + \text{conn}$ *formula* $\varphi(\bar{x})$, *we define the satisfaction relation* $(\mathfrak{A}, \bar{a}) \models \varphi(\bar{x})$ *as for first-order logic, where an atomic predicate* $\text{conn}_k(x, y, z_1, \ldots, z_k)$ *is evaluated as follows. Assume that the universe of* $\mathfrak{A}$ *is* A *and let* $G = (A, E^{\mathfrak{A}})$ *be the graph on vertex set* A *and edge set* $E^{\mathfrak{A}}$. *Then* $(\mathfrak{A}, \bar{a})$ *models* $\text{conn}_k(x, y, z_1, \ldots, z_k)$ *if and only if* $\bar{a}(x)$ *and* $\bar{a}(y)$ *are connected in* $G - \{\bar{a}(z_1), \ldots, \bar{a}(z_k)\}$.

Note in particular that if $\bar{a}(x) = \bar{a}(z_i)$ or $\bar{a}(y) = \bar{a}(z_i)$ for some $i \leq k$, then $(\mathfrak{A}, \bar{a}) \not\models \text{conn}_k(x, y, z_1, \ldots, z_k)$.

We write $\text{FO} + \text{conn}_k$ for the fragment of $\text{FO} + \text{conn}$ that uses only conn_ℓ predicates for $\ell \leq k$. The quantifier rank of an $\text{FO} + \text{conn}$ formula is defined as for plain first-order logic. For structures $\mathfrak{A}$ with universe A and $\bar{a} \in A^m$ and $\mathfrak{B}$ with universe B and $\bar{b} \in B^m$, we write $(\mathfrak{A}, \bar{a}) \equiv_{\text{conn}} (\mathfrak{B}, \bar{b})$ if $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$ satisfy the same $\text{FO} + \text{conn}$ formulas, that is, for all $\varphi(\bar{x})$ we have $\mathfrak{A} \models \varphi(\bar{a}) \Leftrightarrow \mathfrak{B} \models \varphi(\bar{b})$. Similarly, we write $(\mathfrak{A}, \bar{a}) \equiv_{\text{conn}_k} (\mathfrak{B}, \bar{b})$ and $(\mathfrak{A}, \bar{a}) \equiv_{\text{conn}_{k,q}} (\mathfrak{B}, \bar{b})$ if $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$ satisfy the same $\text{FO} + \text{conn}_k$ formulas and the same $\text{FO} + \text{conn}_k$ formulas of quantifier rank at most q, respectively.

3.1 Expressive power of separator logic

We now give examples of properties that are expressible with separator logic.

▶ **Example 3.2.** Connectivity is expressible in $\text{FO} + \text{conn}_0$ by the formula

$$\forall x \forall y \big(\text{conn}_0(x, y)\big).$$

More generally, for every non-negative integer k, $(k+1)$-connectivity can be expressed by the formula

$$\forall x \forall y \forall z_1 \ldots \forall z_k \Big(\bigwedge_{1 \leq i \leq k} (x \neq z_i \wedge y \neq z_i) \to \text{conn}_k(x, y, z_1, \ldots, z_k) \Big).$$

▶ **Example 3.3.** We can express that there exists a cycle by

$$\exists x \exists y \big(E(x,y) \wedge \exists z \big(\mathrm{conn}_1(z,x,y) \wedge \mathrm{conn}_1(z,y,x) \big) \big),$$

hence, that a graph is acyclic by the negation of that formula. We write $\psi_{acyclic}$ for that formula. We can express that a graph is a tree by stating that it is connected and acyclic.

We can conveniently express deletion problems by relativizing formulas as follows. For a formula φ that does not contain z as a free variable write $\mathrm{del}(z)[\varphi]$ for the formula obtained from φ by recursively replacing every subformula $\exists x \psi$ by $\exists x(x \neq z \wedge \psi)$, every subformula $\forall x \psi$ by $\forall x(x \neq z \to \psi)$ and every atomic formula $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$ by $\mathrm{conn}_{k+1}(x, y, z_1, \ldots, z_k, z)$. Then $(\mathfrak{A}, \bar{a}) \models \mathrm{del}(z)[\varphi]$ if and only if $(\mathfrak{A} - \bar{a}(z), \bar{a}) \models \varphi$, where $\mathfrak{A} - \bar{a}(z)$ denotes the substructure induced on the universe of $\mathfrak{A}$ without $\bar{a}(z)$.

▶ **Example 3.4.** We can state the existence of a feedback vertex set of size k by

$$\exists z_1 \mathrm{del}(z_1)[\cdots[\exists z_k \mathrm{del}(z_k)[\psi_{acyclic}] \ldots].$$

We can of course use the same principle to express any Π-deletion problem that is FO + conn expressible.

We can also, much more generally, express many elimination distance problems.

▶ **Example 3.5.** The *elimination distance* to a class $\mathscr{C}$ of graphs measures the number of recursive deletions of vertices needed for a graph G to become a member of $\mathscr{C}$. More precisely, a graph G has elimination distance 0 to $\mathscr{C}$ if $G \in \mathscr{C}$, and otherwise elimination distance at most $k+1$ if in every connected component of G we can delete a vertex such that the resulting graph has elimination distance at most k to $\mathscr{C}$. Elimination distance was introduced by Bulian and Dawar [5] in their study of the parameterized complexity of the graph isomorphism problem and has recently obtained much attention in the literature, see e.g. [1, 4, 13, 19, 20, 23].

Again, we define auxiliary notation. We write $\mathrm{comp}(x)$ for the connected component of (the valuation of) x. For a formula φ we write $\varphi^{[\mathrm{comp}(x)]}$ for the formula obtained from φ by recursively replacing all subformulas $\exists y \psi$ by $\exists y(\mathrm{conn}_0(x,y) \wedge \psi)$ and all subformulas $\forall y \psi$ by $\forall y(\mathrm{conn}_0(x,y) \to \psi)$. Then $(\mathfrak{A}, \bar{a}) \models \varphi^{[\mathrm{comp}(x)]}$ if and only if $(\mathfrak{A}[\mathrm{comp}(\bar{a}(x))], \bar{a}) \models \varphi$, where $\mathfrak{A}[\mathrm{comp}(\bar{a}(x))]$ denotes the substructure induced on the connected component of $\bar{a}(x)$.

Now assume $\mathscr{C}$ is a first-order definable class, say defined by a formula $\psi_{\mathscr{C}}$. Then elimination distance 0 to $\mathscr{C}$ is defined by $\mathrm{ed}_0 = \psi_{\mathscr{C}}$. If ed_k has been defined, then we can express elimination distance $k+1$ to $\mathscr{C}$ by the formula

$$\mathrm{ed}_{k+1} \coloneqq \mathrm{ed}_k \vee \forall x \big(\exists y \ \mathrm{del}(y)[\mathrm{ed}_k] \big)^{[\mathrm{comp}(x)]}.$$

Our final example concerns the expressive power of separator logic on finite words and finite trees. By the classical result of Büchi, a language on words is regular if and only if it is definable in MSO. Here, words are represented as finite structures over the vocabulary of the successor relation and unary predicates representing the letters of the alphabet. When considering first-order logic on strings, it makes a big difference whether one considers word structures over the successor relation or over its transitive closure, the order relation. Languages definable by FO over the order relation are exactly the star-free languages (see e.g. [22, Theorem 7.26]), while languages definable by FO over the successor relation are exactly the locally threshold testable languages [27, Theorem 4.8]. Similarly, MSO on trees can define exactly the tree regular languages (defined via tree automata,

see [22, Theorem 7.30]), while FO can only define a proper subclass of the regular tree languages when the ancestor-descendant or even only the parent-child relation is present. This background was also the motivation of Bojańczyk, who studied a variant of star-free expressions for graphs and showed that these expressions exactly correspond to separator logic [2]. In our example, we show that separator logic on rooted trees has exactly the same expressive power as first-order logic in the presence of the ancestor-descendant relation. Let us write FO[<] for the latter logic. On the other hand, we treat a rooted tree as a graph-theoretic tree with an additional unary predicate marking the root. In the degenerate case, we treat a word as a path, where one of the endpoints is marked by a unary predicate as the smallest vertex (the beginning of the word).

▶ **Example 3.6.** On rooted trees (and similarly on words) $\mathrm{FO} + \mathrm{conn}$ collapses to $\mathrm{FO} + \mathrm{conn}_1$ and has exactly the same expressive power as FO[<] over trees with the ancestor-descendant relation. We show first that $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$ can be expressed in FO[<]. For this, we need to ensure that x and y are not equal to any z_i and that no z_i lies on the unique path between x and y in the tree. We can define the vertices on the unique path between x and y by first defining the least common ancestor of x and y by the formula $\mathrm{lca}(x, y, z) = z \leq x \wedge z \leq y \wedge \neg\exists z'(z < z' \wedge z' \leq x \wedge z' \leq y)$. If z is the least common ancestor of x and y, it remains to state that none of the z_i lies either between x and z or between y and z, which is done by the formula $\exists z\big(\mathrm{lca}(x, y, z) \wedge \bigwedge_{1 \leq i \leq k} \neg(z \leq z_i \leq x \vee z \leq z_i \leq y)\big)$.

Conversely, we show that we can define with $\mathrm{FO} + \mathrm{conn}_1$ the ancestor-descendant relation in rooted trees. Assume the root is marked by the unary symbol R. Then $x < y$ is equivalent to $\exists r\big(R(r) \wedge \mathrm{conn}_1(x, r, y) \wedge \neg\mathrm{conn}_1(y, r, x)\big)$.

3.2 The limits of separator logic

We now study the limits of separator logic and show that planarity cannot be expressed in $\mathrm{FO} + \mathrm{conn}$. Slightly abusing notation let us also write $\mathrm{FO} + \mathrm{conn}_k$ for the properties that are expressible in $\mathrm{FO} + \mathrm{conn}_k$. We show that there is a strict hierarchy of expressiveness: $\mathrm{FO} + \mathrm{conn}_0 \subsetneq \mathrm{FO} + \mathrm{conn}_1 \subsetneq \mathrm{FO} + \mathrm{conn}_2 \subsetneq \ldots$ These results are based on an adaptation of the standard Ehrenfeucht-Fraïssé game (EF game), which is commonly used in the study of the expressive power of first-order logic.

Ehrenfeucht-Fraïssé Games. The Ehrenfeucht-Fraïssé game is played by two players called *Spoiler* and *Duplicator*. Given two structures $\mathfrak{A}$ and $\mathfrak{B}$, Spoiler's aim is to show that the structures can be distinguished by first-order logic (with formulas of a given quantifier rank), while Duplicator wants to prove the opposite. The q-round EF game proceeds in q rounds, where each round consists of the following two steps.

1. Spoiler picks an element $a \in \mathfrak{A}$ or an element $b \in \mathfrak{B}$.
2. Duplicator responds by picking an element of the other structure, that is, she picks a $b \in \mathfrak{B}$ if Spoiler chose $a \in \mathfrak{A}$, and she picks an $a \in \mathfrak{A}$ if Spoiler chose $b \in \mathfrak{B}$.

After q rounds, the game stops. Assume the players have chosen $\bar{a} = a_1, \ldots, a_q$ and $\bar{b} = b_1, \ldots, b_q$. Then Duplicator *wins* if the mapping $a_i \mapsto b_i$ for all $1 \leq i \leq q$ is a partial isomorphism of $\mathfrak{A}$ and $\mathfrak{B}$. We write for short $\bar{a} \mapsto \bar{b}$ for this mapping. Otherwise, Spoiler wins. We say that Duplicator *wins the q-round EF game* on $\mathfrak{A}$ and $\mathfrak{B}$ if she can force a win no matter how Spoiler plays. We then write $\mathfrak{A} \simeq_q \mathfrak{B}$.

▶ **Theorem 3.7** (Ehrenfeucht-Fraïssé, see e.g. [22, Theorem 3.18])**.** *Let $\mathfrak{A}$ and $\mathfrak{B}$ be two σ-structures where σ is purely relational. Then $\mathfrak{A} \equiv_q \mathfrak{B}$ if and only if $\mathfrak{A} \simeq_q \mathfrak{B}$.*

The EF game for FO naturally extends to separator logic. The *($\mathrm{conn}_{k,q}$)-game* is played just as the q-round EF game, but the winning condition is changed as follows. If in q rounds the players have chosen $\bar{a} = a_1, \ldots, a_q$ and $\bar{b} = b_1, \ldots, b_q$, then Duplicator wins if

1. the mapping $\bar{a} \mapsto \bar{b}$ is a partial isomorphism of $\mathfrak{A}$ and $\mathfrak{B}$, and
2. for every $\ell \leq k$ and every sequence $(i_1, \ldots, i_{\ell+2})$ of numbers in $\{1, \ldots, q\}$ we have

$$\mathfrak{A} \models \mathrm{conn}_\ell(a_{i_1}, \ldots, a_{i_{\ell+2}}) \iff \mathfrak{B} \models \mathrm{conn}_\ell(b_{i_1}, \ldots, b_{i_{\ell+2}}).$$

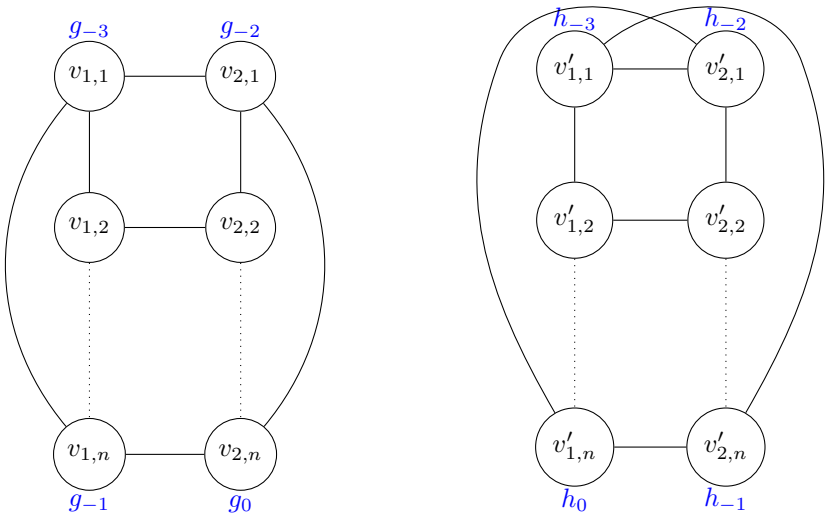
Otherwise, Spoiler wins. We say that Duplicator wins the $(\mathrm{conn}_{k,q})$-game on $\mathfrak{A}$ and $\mathfrak{B}$ if she can force a win no matter how Spoiler plays. We then write $\mathfrak{A} \simeq_{\mathrm{conn}_{k,q}} \mathfrak{B}$.

By following the lines of the proof of the classical Ehrenfeucht-Fraïssé Theorem we can prove the following theorem.

▶ **Theorem 3.8.** *Let $\mathfrak{A}$ and $\mathfrak{B}$ be two σ-structures where σ is purely rational (and contains a binary relation symbol E that is interpreted on both structures as an irreflexive and symmetric relation). Then $\mathfrak{A} \equiv_{\mathrm{conn}_{k,q}} \mathfrak{B}$ if and only if $\mathfrak{A} \simeq_{\mathrm{conn}_{k,q}} \mathfrak{B}$.*

The next theorem exemplifies the use of the $(\mathrm{conn}_{k,q})$-game.

▶ **Theorem 3.9.** *Planarity is not expressible in* $\mathrm{FO} + \mathrm{conn}$.



(a) G_q **(b)** H_q

Figure 1 Planarity is not expressible in $\mathrm{FO} + \mathrm{conn}$.

Proof. Assume planarity is expressible by a sentence φ of $\mathrm{FO} + \mathrm{conn}_k$ of quantifier rank q. Without loss of generality, we may assume that $k \leq q$, as otherwise, we have repetitions in the conn_k predicates that can be avoided by using conn_ℓ predicates for $\ell < k$. Let G_q and H_q be defined as shown in Figure 1, where $n = 2^{q+1}$. Then, G_q is planar but H_q embeds only in a surface of genus one (into the Möbius strip, which cannot be embedded into the plane). We show that $G_q \simeq_{\mathrm{conn}_{k,q}} H_q$, contradicting the assumption that φ must distinguish G_q and H_q. In fact, we prove an even stronger statement by giving Spoiler four free moves $g_{-3} = v_{1,1}$, $g_{-2} = v_{2,1}$, $g_{-1} = v_{1,n}$ and $g_0 = v_{2,n}$ in G_q and forcing Duplicator to respond with the vertices $h_{-3} = v'_{1,1}$, $h_{-2} = v'_{2,1}$, $h_{-1} = v'_{2,n}$ and $h_0 = v'_{1,n}$ in H_q. Note the twist in the last two vertices. These extra moves are helpful to define Duplicator's winning strategy.

We define the x-distance of two nodes $v_{i,j}$ and $v_{k,\ell}$ as $d_x(v_{i,j}, v_{k,\ell}) = |i-k|$ and the y-distance as $d_y(v_{i,j}, v_{k,\ell}) = |j-\ell|$. Note that the y-distance is not the distance in the graphs, e.g. $d_y(g_{-3}, g_{-1}) = 2^{q+1} - 1$, even though g_{-3} and g_{-1} are adjacent in G_q.

Assume now that the first i moves have been made in the game and the players have selected the vertices $\bar{g} = (g_{-3}, \dots, g_0, g_1, \dots, g_i)$ in G_q (where $g_1, \dots, g_i$ were freely chosen by the players), and $\bar{h} = (h_{-3}, \dots, h_0, h_1, \dots, h_i)$ in H_q (where $h_1, \dots, h_i$ were freely chosen by the players). We prove by induction that Duplicator can play in such a way that after round i of the $(\mathrm{conn}_{k,q})$-game the following conditions hold for all $-3 \leq j, \ell \leq i$:

1. if $g_j = v_{x,y}$, then $h_j = v'_{x',y}$, that is, corresponding pebbles are in the same row, and in particular $d_y(g_j, g_\ell) = d_y(h_j, h_\ell)$, and
2. if $d_y(g_j, g_\ell) \leq 2^{q-i}$, then $d_x(g_j, g_\ell) = d_x(h_j, h_\ell)$.

These conditions together with the first four extra moves imply that the mapping $\bar{g} \mapsto \bar{h}$ is a partial isomorphism of G_q and H_q. Let us show that also for every $0 \leq \ell \leq k$ and every sequence $(i_1, \dots, i_{\ell+2})$ of numbers in $\{-3, \dots, i\}$ we have $G_q \models \mathrm{conn}_\ell(g_{i_1}, \dots, g_{i_{\ell+2}})$ if and only if $H_q \models \mathrm{conn}_\ell(h_{i_1}, \dots, h_{i_{\ell+2}})$. Assume $G_q \models \mathrm{conn}_\ell(g_{i_1}, \dots, g_{i_{\ell+2}})$, that is, g_{i_1} and g_{i_2} are connected after the deletion of $g_{i_3}, \dots, g_{i_{\ell+2}}$, say by a path $P = v_{x_1,y_1} \dots v_{x_m,y_m}$, where $v_{x_1,y_1} = g_{i_1}$ and $v_{x_m,y_m} = g_{i_2}$. Then there are no $g_{i_{j_1}} = v_{x,y}$ and $g_{i_{j_2}} = v_{x',y'}$ (for $j_1, j_2 \geq 3$) with $y = y' = y_i$ and $x \neq x'$ for some $2 \leq i \leq m-1$ (this would block a row along which the path goes, which is not possible) and no $g_{i_{j_1}} = v_{x,y}$ and $g_{i_{j_2}} = v_{x',y'}$ (for $j_1, j_2 \geq 3$) with $y_i = y = y' - 1 = y_{i+1} - 1$ and $x \neq x'$ for some $2 \leq i \leq m-1$ (this would block a "diagonal" of which the path contains at least one vertex, which is not possible). By the first condition of the invariant there are no $h_{i_{j_1}} = v_{x,y}$ and $h_{i_{j_2}} = v_{x',y'}$ (for $j_1, j_2 \geq 3$) with $y = y' = y_i$ and $x \neq x'$ for some $2 \leq i \leq m-1$ and by the second condition of the invariant there are no $h_{i_{j_1}} = v_{x,y}$ and $h_{i_{j_2}} = v_{x',y'}$ (for $j_1, j_2 \geq 3$) with $y_i = y = y' - 1 = y_{i+1} - 1$ and $x \neq x'$ for some $2 \leq i \leq m-1$. Now, if $P' = v'_{x_1,y_1} \dots v'_{x_m,y_m}$ is not a path from h_{i_1} to h_{i_2} after the deletion of $h_{i_3}, \dots, g_{i_{\ell+2}}$, it is possible to reroute the path by switching the row appropriately, as the h_{i_j} never block a complete row or a diagonal, as shown above. The case $H_q \models \mathrm{conn}_\ell(h_{i_1}, \dots, h_{i_{\ell+2}})$ is symmetrical.

We now show that Duplicator can maintain this invariant throughout the game. For the initial configuration $i = 0$, the conditions are obviously fulfilled for $-3 \leq j, \ell \leq 0$. Corresponding pebbles are in the same row and note that $d_y(g_j, g_\ell) = 2^{q+1} - 1$, for $j \in \{-3, -2\}$ and $\ell \in \{-1, 0\}$ and analogously for h_j and h_ℓ.

For the induction step, suppose that the conditions are fulfilled so far and that Spoiler is making his $(i+1)$-move in G_q (the case of H_q is symmetrical). We may assume that Spoiler does not choose a vertex that was chosen before, say Spoiler picks $g_{i+1} = v_{\_,a}$. Duplicator must choose $h_{i+1} = v'_{\_,a}$ with the same y-coordinate. We have to make sure that she can choose the vertex with that y-coordinate satisfying the second condition. Let $g_j = v_{\_,b}$ and $g_\ell = v_{\_,c}$ with $-3 \leq j, \ell \leq i$ be such that $b \leq a \leq c$ and there is no other $g_k = v_{\_,d}$ with $b < d < c$. Intuitively, g_j is the lowest pebble that was placed above (or in the same row as) g_{i+1}, while g_k is the highest pebble that was placed below (or in the same row as) g_{i+1}. There are two cases:

1. $d_y(g_j, g_\ell) \leq 2^{q-i}$: Then by hypothesis, $d_x(h_j, h_\ell) = d_x(g_j, g_\ell)$ and $d_y(h_j, h_\ell) = d_x(g_j, g_\ell)$. Here, Duplicator chooses the unique $h_{i+1} = v'_{\_,a}$ such that $d_x(h_j, h_{i+1}) = d_x(g_j, g_{i+1})$, and we have $d_x(h_\ell, h_{i+1}) = d_x(g_\ell, g_{i+1})$.
2. $d_y(g_j, g_\ell) > 2^{q-i}$: Then $d_y(h_j, h_\ell) > 2^{q-i}$ and there are three possibilities:
   - $d_y(g_j, g_{i+1}) \leq 2^{q-(i+1)}$: Then $d_y(g_\ell, g_{i+1}) > 2^{q-(i+1)}$, and Duplicator chooses $h_{i+1} = v'_{\_,a}$ such that $d_x(h_j, h_{i+1}) = d_x(g_j, g_{i+1})$. Hence, $d_y(h_\ell, h_{i+1}) > 2^{q-(i+1)}$.

- $d_y(g_\ell, g_{i+1}) \leq 2^{q-(i+1)}$: Then $d_y(g_j, g_{i+1}) > 2^{q-(i+1)}$. Similarly to the previous case, Duplicator chooses $h_{i+1} = v'_{\_,a}$ such that $d_x(h_\ell, h_{i+1}) = d_x(g_\ell, g_{i+1})$. Consequently, $d_y(h_j, h_{i+1}) > 2^{q-(i+1)}$.
- $d_y(g_j, g_{i+1}) > 2^{q-(i+1)}$ and $d_y(g_\ell, g_{i+1}) > 2^{q-(i+1)}$: Here, Duplicator can choose $h_{i+1} = v'_{1,a}$ or $h_{i+1} = v'_{2,a}$ as she wants. We get that $d_y(h_j, h_{i+1}) \geq 2^{q-(i+1)}$ and $d_y(h_\ell, h_{i+1}) \geq 2^{q-(i+1)}$.

Thus, in all cases, the conditions are fulfilled and Duplicator wins the $(\mathrm{conn}_{k,q})$-game on G_q and H_q. Hence, planarity is not definable in $\mathrm{FO} + \mathrm{conn}$. ◀

As a graph is planar if and only if it excludes K_5 and $K_{3,3}$ as (topological) minors and we will show that this can be expressed using disjoint paths predicates, we conclude that the disjoint paths predicate cannot be expressed with $\mathrm{FO} + \mathrm{conn}$.

▶ **Corollary 3.10.** *The disjoint paths problem cannot be expressed in* $\mathrm{FO} + \mathrm{conn}$.

The proof of the next theorem is deferred to the next section, as it is a consequence of the fact that the even stronger logic $\mathrm{FO} + \mathrm{DP}$ cannot express bipartiteness (Theorem 4.7).

▶ **Theorem 3.11.** *Bipartiteness cannot be expressed in* $\mathrm{FO} + \mathrm{conn}$.

Finally, we show that the $\mathrm{FO}+\mathrm{conn}_k$ hierarchy is strict by proving that $(k+2)$-connectivity cannot be expressed by $\mathrm{FO}+\mathrm{conn}_k$. On the other hand, $(k+2)$-connectivity can be expressed by $\mathrm{FO} + \mathrm{conn}_{k+1}$ (Example 3.2).

▶ **Theorem 3.12.** *$(k+2)$-connectivity cannot be expressed by* $\mathrm{FO} + \mathrm{conn}_k$. *In particular, the* $\mathrm{FO} + \mathrm{conn}_k$ *hierarchy is strict, that is,* $\mathrm{FO} + \mathrm{conn}_0 \subsetneq \mathrm{FO} + \mathrm{conn}_1 \subsetneq \ldots$

Proof. Let k be an integer. For every integer q, we choose two graphs G_q and H_q such that:

- G_q is connected,
- H_q is not connected, and
- $G_q \simeq_q H_q$.

This is possible, as connectivity is not first-order definable and $\simeq_q$ has only finitely many equivalence classes.

Then, we define the graph G_q^k (resp. H_q^k) as the disjoint union of G_q (resp. H_q) and K_{k+1}, a clique of size $k+1$, and connect the vertices of the clique with all vertices of G_q (resp. H_q), that is, we add the additional edges such that $(x, y) \in E(G_q^k)$ (resp. $(x, y) \in E(H_q^k)$) if $x \in G_q$ (resp. $x \in H_q$) and $y \in K_{k+1}$. Obviously, G_q^k is $(k+2)$-connected (the deletion of any $k+1$ vertices cannot disconnect G_q^k), while H_q^k is not $(k+2)$-connected (the deletion of the copy of K_{k+1} disconnects H_q^k).

The same argument shows that every $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$ can be expressed by an atomic plain first-order formula: in both graphs (the valuations of) x and y are not connected after the deletion of (the valuations of) $z_1, \ldots, z_k$ if and only if x or y is equal to one of the z_i. Hence, to prove $G_q^k \simeq_{\mathrm{conn}_{k,q}} H_q^k$ it suffices to prove $G_q^k \simeq_q H_q^k$, and this finishes the proof.

▷ Claim 3.13. For all integers q, k we have $G_q^k \simeq_q H_q^k$.

Proof. The following is obviously a winning strategy for Duplicator in the q-round EF game on G_q^k and H_q^k. If Spoiler plays a pebble in the subgraph G_q or H_q, Duplicator can respond by a pebble in the subgraph H_q or G_q according to the winning strategy of Duplicator in the EF game on G_q and H_q. Otherwise, if Spoiler picks a pebble in the subgraph K_{k+1} of G_q^k or H_q^k, Duplicator can respond by a pebble in the subgraph K_{k+1} of the other graph H_q^k or G_q^k. ◁

This concludes the proof of Theorem 3.12. ◀

4 Disjoint-paths logic

In this section, we study the expressive power of disjoint-paths logic FO + DP. We again fix a signature σ that does not contain the symbol disjoint-paths$_k$ for any $k \geq 1$ and that does contain a binary (edge) relation symbol E. The disjoint paths predicates will always refer to this relation. We let $\sigma + \text{disjoint-paths} := \sigma \cup \{\text{disjoint-paths}_k : k \geq 1\}$, where each disjoint-paths$_k$ is a $2k$-ary relation symbol.

▶ **Definition 4.1.** *The formulas of* $(\text{FO} + \text{DP})[\sigma]$ *are the formulas of* $\text{FO}[\sigma + \text{disjoint-paths}]$. *We usually simply write* FO + DP*, when* σ *is understood from the context.*

For a σ*-structure* $\mathfrak{A}$*, an assignment* $\bar{a}$ *and an* FO + DP *formula* $\varphi(\bar{x})$*, we define the satisfaction relation* $(\mathfrak{A}, \bar{a}) \models \varphi(\bar{x})$ *as for first-order logic, where an atomic predicate* disjoint-paths$_k[(x_1, y_1), \dots (x_k, y_k)]$ *is evaluated as follows. Assume that the universe of* $\mathfrak{A}$ *is* A *and let* $G = (A, E^{\mathfrak{A}})$ *be the graph on vertex set* A *and edge set* $E^{\mathfrak{A}}$*. Then* $(\mathfrak{A}, \bar{a})$ *models* disjoint-paths$_k[(x_1, y_1), \dots, (x_k, y_k)]$ *if and only if in* G *there exist* k *internally vertex-disjoint paths* $P_1, \dots, P_k$*, where* P_i *connects* $\bar{a}(x_i)$ *and* $\bar{a}(y_i)$*.*

As previously mentioned, it is natural to consider these predicates for both undirected and directed graphs. We will, however, in this work only study the undirected case.

We write $\text{FO} + \text{DP}_k$ for the fragment of FO + DP that uses only disjoint-paths$_\ell$ predicates for $\ell \leq k$. The quantifier rank of an FO + DP formula is defined as for plain first-order logic. For structures $\mathfrak{A}$ with universe A and $\bar{a} \in A^m$ and $\mathfrak{B}$ with universe B and $\bar{b} \in B^m$, we write $(\mathfrak{A}, \bar{a}) \equiv_{\text{DP}} (\mathfrak{B}, \bar{b})$ if $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$ satisfy the same FO + DP formulas, that is, for all $\varphi(\bar{x})$ we have $\mathfrak{A} \models \varphi(\bar{a}) \Leftrightarrow \mathfrak{B} \models \varphi(\bar{b})$. Similarly, we write $(\mathfrak{A}, \bar{a}) \equiv_{\text{DP}_k} (\mathfrak{B}, \bar{b})$ and $(\mathfrak{A}, \bar{a}) \equiv_{\text{DP}_{k,q}} (\mathfrak{B}, \bar{b})$ if $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$ satisfy the same $\text{FO} + \text{DP}_k$ formulas and the same $\text{FO} + \text{DP}_k$ formulas of quantifier rank at most q, respectively.

4.1 Expressive power of disjoint-paths logic

We now study the expressive power of disjoint-paths logic.

▶ **Observation 4.2.** $\text{FO} + \text{conn} \subseteq \text{FO} + \text{DP}$ *because* $\text{conn}_k(x, y, z_1, \dots, z_k)$ *is equivalent to* $\text{disjoint-paths}_{k+1}[(x, y), (z_1, z_1), \dots, (z_k, z_k)] \wedge \bigwedge_{i \leq k} (z_i \neq x \wedge z_i \neq y)$.

Moreover, the inclusion is strict because planarity is not expressible in FO + conn as seen in Corollary 3.10. We show that planarity and in fact the property that a graph contains a fixed (topological) minor can be expressed in FO + DP.

▶ **Example 4.3.** For every fixed graph H, there is an FO + DP formula φ_H^{top} such that $G \models \varphi_H^{top}$ if and only if $H \preccurlyeq^{top} G$.

Let n, m, ℓ respectively be the number of vertices, edges, and isolated vertices in H. Let $x_1, \dots x_n$ be n variables. Let $e_1, \dots, e_m$ be the list of edges of H, and let v_{j_s} and v_{j_t} be the two endpoints of e_j. Finally, let $v_{i_1}, \dots, v_{i_\ell}$ be the isolated vertices of H. Then,

$$\varphi_H^{top} := \exists x_1, \dots x_n \Big(\bigwedge_{i \neq j} x_i \neq x_j \quad \wedge \\ \text{disjoint-paths}[(x_{e_{1_s}}, x_{e_{1_t}}), \dots (x_{e_{m_s}}, x_{e_{m_t}}), (x_{i_1}, x_{i_1}), \dots (x_{i_\ell}, x_{i_\ell})] \Big).$$

▶ **Example 4.4.** For every fixed graph H, there is an FO + DP formula φ_H such that $G \models \varphi_H$ if and only if $H \preccurlyeq G$. This is because, for every graph H, there exists a finite family of graphs $H_1, \dots, H_\ell$ such that $H \preccurlyeq G$ if and only if there is an $i \leq \ell$ such that $H_i \preccurlyeq^{top} G$.

This family can be obtained by considering all possibilities of replacing every branch set representing a vertex of H of degree $d \geq 3$ with a tree with at most d leaves and hardcoding their shapes by disjoint paths.

▶ **Example 4.5.** Planarity can be expressed in FO + DP. This is a corollary of the previous example, using the formula $\varphi_{planar} := \neg\varphi_{K_5} \wedge \neg\varphi_{K_{3,3}}$.

4.2 The limits of disjoint-paths logic

We now study the limits of disjoint-paths logic and show that bipartiteness cannot be expressed in FO + DP. We also show that the hierarchy on $(\mathrm{FO} + \mathrm{DP}_k)_{k\geq 1}$ is strict. These results are based again on an adaptation of the standard Ehrenfeucht-Fraïssé game.

The *($\mathrm{DP}_{k,q}$)-game* is played just as the q-round EF game, but the winning condition is changed as follows. If in q rounds the players have chosen $\bar{a} = a_1, \ldots, a_q$ and $\bar{b} = b_1, \ldots, b_q$, then Duplicator wins if

1. the mapping $\bar{a} \mapsto \bar{b}$ is a partial isomorphism of $\mathfrak{A}$ and $\mathfrak{B}$, and
2. for every $\ell \leq k$ and every sequence $(i_1, \ldots, i_{2\ell})$ of numbers in $\{1, \ldots, q\}$ we have

$$\begin{aligned} & \mathfrak{A} \models \text{disjoint-paths}[(a_{i_1}, a_{i_2}), \ldots, (a_{i_{2\ell-1}}, a_{i_{2\ell}})] \\ \iff \quad & \mathfrak{B} \models \text{disjoint-paths}[(b_{i_1}, b_{i_2}), \ldots, (b_{i_{2\ell-1}}, b_{i_{2\ell}})]. \end{aligned}$$

Otherwise, Spoiler wins. We say that Duplicator *wins the ($\mathrm{DP}_{k,q}$)-game* on $\mathfrak{A}$ and $\mathfrak{B}$ if she can force a win no matter how Spoiler plays. We then write $\mathfrak{A} \simeq_{\mathrm{DP}_{k,q}} \mathfrak{B}$.

By following the lines of the proof of the classical Ehrenfeucht-Fraïssé Theorem we can prove the following theorem.

▶ **Theorem 4.6.** *Let $\mathfrak{A}$ and $\mathfrak{B}$ be two σ-structures where σ is purely rational (and contains a binary relation symbol E that is interpreted on both structures as an irreflexive and symmetric relation). Then $\mathfrak{A} \equiv_{\mathrm{DP}_{k,q}} \mathfrak{B}$ if and only if $\mathfrak{A} \simeq_{\mathrm{DP}_{k,q}} \mathfrak{B}$.*

▶ **Theorem 4.7.** *Bipartiteness is not definable in* FO + DP*.*

Proof. Let q be an integer, and let G be a cycle graph with 2^q vertices and H a cycle graph with $2^q + 1$ vertices. Then, G is bipartite because it has an even number of vertices, and H is not bipartite because it has an odd number of vertices. We want to show that $G \simeq_{\mathrm{DP}_{k,q}} H$ by induction over q.

We define the distance $d(x, y)$ of two vertices x and y as the length of the shortest path between x and y.

Let $\bar{g} = (g_1, \ldots, g_i)$ be the first i moves in G and similarly $\bar{h} = (h_1, \ldots, h_i)$ the first i moves in H. We can prove by induction that Duplicator can play in such a way that after round i of the ($\mathrm{DP}_{k,q}$)-game the following conditions hold for all $j, \ell \leq i$:

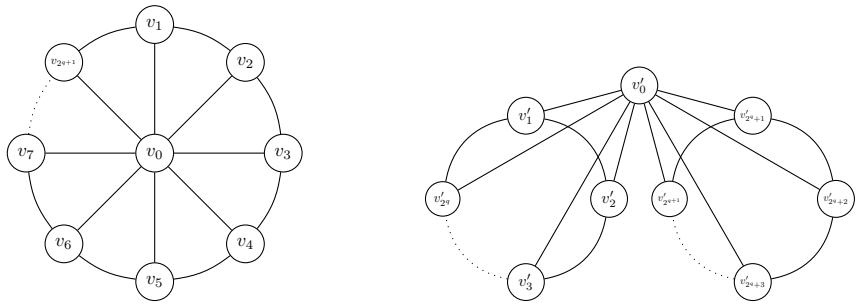
1. If $d(g_j, g_\ell) < 2^{q-i+1}$, then $d(g_j, g_\ell) = d(h_j, h_\ell)$.
2. If $d(g_j, g_\ell) \geq 2^{q-i+1}$, then $d(h_j, h_\ell) \geq 2^{q-i+1}$.
3. The pebbles are placed in G and H with the same "circular order".

By the first two conditions, the partial isomorphism $\bar{g} \mapsto \bar{h}$ can be ensured. Furthermore, the third condition implies that the second condition for Duplicator's win is also satisfied.

The base case $i = 1$ of the induction is trivial because $d(g_1, g_1) = d(h_1, h_1) = 0$.

For the induction step, suppose that $G \simeq_{\mathrm{DP}_{k,i}} H$ holds and Spoiler is making his $(i+1)$-st move in G. The case of H is equivalent.

If Spoiler picks g_j for some $j \leq i$, a pebble that was already played before, Duplicator can choose h_j, and the conditions are fulfilled by the induction hypothesis. Otherwise, Spoiler picks a pebble g_{i+1} that wasn't played before. Now we have to differentiate two cases:



(a) G_q **(b)** H_q

Figure 2 FO + DP hierarchy is strict.

1. There is only one other pebble that was already played, $g_j = g_1, j \leq i$. Then, we can find h_{i+1} such that $d(h_1, h_{i+1}) = d(g_1, g_{i+1})$.
2. g_{i+1} lies on the shortest path of g_j and g_ℓ with $j, \ell \leq i$ such that there is no other $g_n, n \leq i$ that lies on this path. Then, there are two possibilities:
   - $d(g_j, g_\ell) < 2^{q-i+1}$: Then $d(h_j, h_\ell) < 2^{q-i+1}$ and we can find h_{i+1} on the shortest path of h_j and h_ℓ such that $d(h_j, h_{i+1}) = d(g_j, g_{i+1})$ and $d(h_{i+1}, h_\ell) = d(g_{i+1}, g_\ell)$.
   - $d(g_j, g_\ell) \geq 2^{q-i+1}$: Then $d(h_j, h_\ell) \geq 2^{q-i+1}$ and there are three cases:
     a. $d(g_j, g_{i+1}) < 2^{q-i}$: Then $d(g_{i+1}, g_\ell) \geq 2^{q-i}$ and we can choose h_{i+1} on the shortest path of h_j and h_ℓ such that $d(h_j, h_{i+1}) = d(g_j, g_{i+1})$ and $d(h_{i+1}, h_\ell) \geq 2^{q-i}$.
     b. $d(g_{i+1}, g_\ell) < 2^{q-i}$: This case is similar to the previous one.
     c. $d(g_j, g_{i+1}) \geq 2^{q-i}$ and $d(g_{i+1}, g_\ell) \geq 2^{q-i}$: Since $d(h_j, h_\ell) \geq 2^{q-i+1}$, we can find h_{i+1} with $d(h_j, h_{i+1}) \geq 2^{q-i}$ and $d(h_{i+1}, h_\ell) \geq 2^{q-i}$ in the middle of the shortest path of h_j, and h_ℓ.

Thus, in all cases, the conditions are fulfilled. This completes the inductive proof. ◀

We now show that the hierarchy on $(\mathrm{FO} + \mathrm{DP}_k)_{k\geq 1}$ is strict.

▶ **Lemma 4.8.** *For all integers $k \geq 1$, $2k$-connectivity is not expressible in* $\mathrm{FO} + \mathrm{DP}_k$.

Proof. Let k be an integer. For every integer q, we define two graphs G_q and H_q such that:
- G_q is 2-connected,
- H_q is 1-connected but not 2-connected, and
- $G_q \simeq_q H_q$

For example, take G_q the cycle with 2^{q+1} many elements, together with an apex vertex, while H_q is the disjoint union of two cycles with 2^q many elements each, together with an apex vertex (see Figure 2).

We then define G_q^k (resp. H_q^k) as the lexicographical product of G_q (resp. H_q) with K_{2k}, the clique with $2k$ elements. More precisely, if $G_q = (V, E)$, where $V = \{1, \ldots, n\}$, then $G_q^k := (V', E')$ where:
- $V' := \{v_{1,1}, \ldots, v_{1,2k}, \ldots, v_{n,1}, \ldots, v_{n,2k}\}$
- $E' := \{\{v_{i,j}, v_{i',j'}\} \ : \ i = i' \vee (i, i') \in E\}$.

One can view G_q^k as $2k$ copies of G_q on top of each other. Vertices are replaced by $2k$-cliques, and edges are replaced by $(2k, 2k)$-bicliques. A direct consequence of the definition is the following equivalence.

▷ Claim 4.9. For all integers q, k, we have that $G_q^k \simeq_q H_q^k$.

Proof. Duplicator's strategy follows the one derived from $G_q \simeq_q H_q$. If Spoiler picks a vertex $v_{i,j} \in G_q^k$, then Duplicator can respond by choosing the vertex $v_{i',j} \in H_q^k$ where $v_{i'} \in H_q$ is Duplicator's respond to $v_i \in G_q$. ◁

We then show that over G_q^k and H_q^k, the predicate disjoint-paths$_k[\ ]$ is always true and therefore that, for these structures, $(\mathrm{FO} + \mathrm{DP}_k)[q]$ collapses to $\mathrm{FO}[q]$.

▷ Claim 4.10. For every integers q, k, for every k-tuples $\bar{a}, \bar{b}$, we have that G_q^k and H_q^k both model disjoint-paths$_k[(a_1, b_1), \ldots, (a_k, b_k)]$.

Proof. The proofs for G_q^k and H_q^k are identical, so we only do it for G_q^k. Remember that n is the number of vertices in G_q. The idea is that each of the k paths uses at most two "copies" of each vertex of G_q, hence $2k$ "copies" is enough for all paths to exists. For every $i \leq n$, let $B_i := \{v_{i,j} \ : \ j \leq 2k\}$, and $F_i := \{v_{i,j} \ : \ j \leq 2k \wedge v_{i,j} \notin \bar{a} \wedge v_{i,j} \notin \bar{b}\}$. We call B_i the set of vertices in *position* i, and F_i the *free vertices* in position i. We then compute each path, starting with (a_1, b_1).

Let i, j, i', j' such that $a_1 = v_{i,j}$ and $b_1 = v_{i',j'}$. If $i = i'$, then there is nothing to do as a_1 and b_1 are neighbors. Otherwise, note that for every $i'' \leq n$, $F_{i''} \neq \emptyset$, because there are only $2k - 2$ elements among $a_2, \ldots, a_k, b_2, \ldots, b_k$. Since G_q is a connected graph, there is a path from i to i'. For every inner node i'' of this path, we can select a vertex $v \in F_{i''}$. We can therefore create a path in G_q^k from a_1 to b_1 where all inner vertices are free vertices. We then remove these vertices from the sets of free vertices.

Let now $1 < \ell \leq k$, and let i, j, i', j' such that $a_\ell = v_{i,j}$ and $b_\ell = v_{i',j'}$. We assume that the first $\ell - 1$ paths have already been computed. Observe that here again, if $i = i'$ there is nothing to do. Otherwise, we again have that for every i'', $F_{i''}$ is not empty. This is because for every $s \leq k$, the path from a_s to b_s intersects $B_{i''}$ at most twice *(at most once for the inner vertices, and twice when the two endpoints are both in position i'')*. Therefore, we can select a path in G_q from i to i' and for each i'' in this path, pick a vertex $v \in F_{i''}$. ◁

With Claim 4.10, we can replace formulas of $(\mathrm{FO} + \mathrm{DP}_k)[q]$ by formulas of $\mathrm{FO}[q]$. Thanks to Claim 4.9, $G_q^k \simeq_q H_q^k$, we conclude that $G_q^k \simeq_{\mathrm{DP}_{k,q}} H_q^k$. So $\mathrm{FO} + \mathrm{DP}_k$ cannot express $2k$-connectivity. Note that this bound is tight for these structures i.e. $G_q^k \not\simeq_{\mathrm{DP}_{k+1,q}} H_q^k$. ◀

▶ **Lemma 4.11.** *The* $\mathrm{FO} + \mathrm{DP}_k$ *hierarchy is strict, that is,* $\mathrm{FO} + \mathrm{DP}_1 \subsetneq \mathrm{FO} + \mathrm{DP}_2 \subsetneq \ldots$

Proof. Consider the structures in the proof of Lemma 4.8, which are indistinguishable in $\mathrm{FO} + \mathrm{DP}_k$. The following sentence of $\mathrm{FO} + \mathrm{DP}_{k+1}$ distinguishes G_q^k and H_q^k:

$$\exists a_1 \ldots \exists b_{k+1}\ \neg\text{disjoint-paths}_{k+1}[(a_1, b_1), \ldots, (a_{k+1}, b_{k+1})]$$

In H_q^k, pick i such that $H_q \setminus i$ is not connected (i' and i'' two disconnected vertices). Then pick $a_j = v_{i,j}$ if $j \leq k$, $b_j = v_{i,k+j}$ if $j \leq k$, and finally $a_{k+1} = v_{i',1}$, $b_{k+1} = v_{i'',1}$. Intuitively, this means that the vertices $v_{i,j}$ are "blocked" for every $j \leq 2k$ by the first k paths and can therefore not be used for the $(k+1)$-st path such that this disjoint path does not exist.

G_q^k does not satisfy the formula because even if we "block" such a clique, there is still a disjoint path connecting every pair of vertices because G_q is 2-connected. ◀

5 Connection to other logics

In this section, we compare the expressive power of the separator logic and the disjoint-paths logic with monadic second-order logic and transitive-closure logic. Figure 3 depicts the connections between these logics.

5.1 Monadic second-order logic

Monadic second-order logic (MSO_1) allows quantification over sets of vertices in addition to the first-order quantifiers. It has a higher expressive power than first-order logic because for example connectivity is expressible in MSO_1 and every first-order formula can be expressed with the first-order quantifiers. Connectivity is expressible by

$$\forall R\Big(\big(\exists x R(x) \wedge \exists x \neg R(x)\big) \rightarrow \exists x \exists y\big(R(x) \wedge \neg R(y) \wedge E(x,y)\big)\Big)$$

By an extension of this formula, we can say that a given set S is connected:

$$\text{conn-set}(S) := \forall R\Big(\big(R \subseteq S \wedge \exists x\ R(x) \wedge \exists x\ (S(x) \wedge \neg R(x))\big) \\ \rightarrow \exists x \exists y\big(R(x) \wedge \neg R(y) \wedge S(y) \wedge E(x,y)\big)\Big)$$

Furthermore, we can express the connectivity operators in MSO_1. The connectivity operator $\mathrm{conn}_0(x,y)$ can be expressed by:

$$\mathrm{conn}_0(x,y) := \forall R\Big(R(x) \wedge \forall v \forall w\big((R(v) \wedge E(v,w)) \rightarrow R(w)\big) \rightarrow R(y)\Big)$$

and $\mathrm{conn}_k(x,y,z_1,\ldots,z_k)$ using $\text{conn-set}(S)$ by:

$$\mathrm{conn}_k(x,y,z_1,\ldots,z_k) := \exists S\ \big(\text{conn-set}(S) \wedge S(x) \wedge S(y) \wedge \bigwedge_{i \leq k} \neg S(z_i)\big).$$

We can express the disjoint paths predicates $\text{disjoint-paths}_k[(x_1,y_1),\ldots,(x_k,y_k)]$ by:

$$\exists S_1,\ldots,S_k\Big(\bigwedge_{i\leq k}\Big(S_i(x_i) \wedge S_i(y_i) \wedge \text{conn-set}(S_i)\Big) \\ \wedge \bigwedge_{i<j\leq k} \forall z\Big(\big(S_i(z) \wedge S_j(z)\big) \rightarrow \big((z = x_i \vee z = y_i) \wedge (z = x_j \vee z = y_j)\big)\Big)\Big)$$

Since the disjoint paths operators are expressible in MSO_1, $\mathrm{FO} + \mathrm{DP}$ is included in MSO_1. This inclusion is strict because it is well-known that bipartiteness is expressible in MSO_1:

$$\exists R_1 \exists R_2\Big(\forall x\big(R_1(x) \leftrightarrow \neg R_2(x)\big) \wedge \bigwedge_{i\leq 2} \forall x \forall y\big((R_i(x) \wedge R_i(y)) \rightarrow \neg E(x,y)\big)\Big)$$

but we showed in Theorem 4.7 that bipartiteness is not expressible in $\mathrm{FO} + \mathrm{DP}$.

5.2 Transitive-closure logic

Transitive-closure logic TC^i_j is the enrichment of first-order logic with the transitive-closure operator $[\mathrm{TC}_{\bar{x},\bar{y}}\varphi(\bar{x},\bar{y})]$ where $\bar{x}$ and $\bar{y}$ are tuples of length i and φ is a formula with at most j free variables other than $\bar{x}$ and $\bar{y}$.

Every $\mathrm{FO} + \mathrm{conn}_k$ formula can be expressed in TC^1_k because the conn_k operator can be expressed with the help of the transitive-closure operator:

$$\mathrm{conn}_k(x,y,z_1,\ldots,z_k) = [\mathrm{TC}_{v,w}E(v,w) \wedge v \neq z_1 \wedge \ldots \wedge v \neq z_k \wedge w \neq z_1 \wedge \ldots \wedge w \neq z_k](x,y)$$

In fact, TC^1_k is more expressible than $\mathrm{FO} + \mathrm{conn}_k$, as it can express bipartiteness [17, Example 7.2]. On the other hand, 2-connectivity can naturally be expressed in $\mathrm{FO} + \mathrm{conn}_1$, but presumably not in TC^1_0.

▶ **Conjecture 5.1.** *2-connectivity cannot be expressed in* TC^1_0.

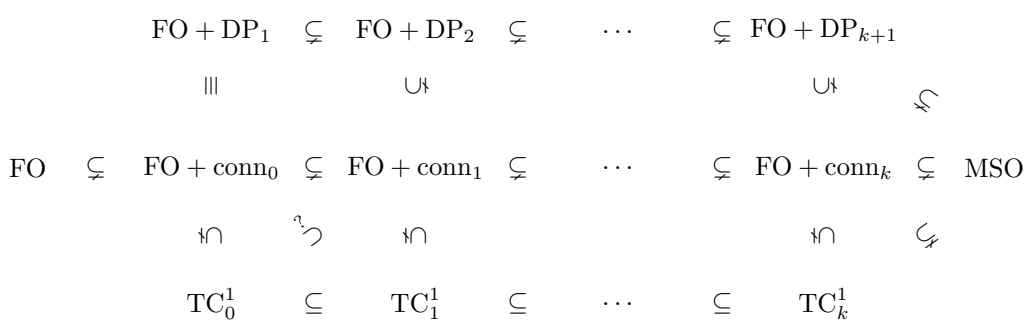


Figure 3 Connections between the logics.

6 Conclusion

We studied first-order logic enriched with connectivity predicates tailored to express algorithmic graph properties that are commonly studied in contemporary parameterized algorithmics. This yielded separator logic, which can query connectivity after the deletion of a bounded number of elements, and disjoint-paths logic, which can express the disjoint-paths problem. We demonstrated a rich expressiveness that arises from the interplay of these predicates with the nested quantification of first-order logic. We also studied the limits of expressiveness of these new logics.

In a companion paper, we studied the model-checking problem for separator logic and proved that it is fixed-parameter tractable parameterized by formula size on classes of graphs that exclude a fixed topological minor [24]. This yields a powerful algorithmic meta-theorem for separator logic. On the other hand, while the disjoint-paths problem is fixed-parameter tractable on general graphs [26], it is not clear that the model-checking problem for disjoint-paths logic is fixed-parameter tractable beyond graphs of bounded treewidth. This remains a challenging question for future work.

It will also be interesting to study other extensions of first-order logic that can express further interesting algorithmic graph problems, such as reachability with regular paths queries. This would, in the simplest case, allow to express bipartiteness and the odd cycle transversal problem. On the other hand, it is very likely that with general regular paths queries, we will get intractability beyond bounded treewidth graphs.

References

1 Akanksha Agrawal, Lawqueen Kanesh, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. An FPT algorithm for elimination distance to bounded degree graphs. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

2 Mikołaj Bojańczyk. Separator logic and star-free expressions for graphs. *arXiv preprint*, 2021. `arXiv:2107.13953`.

3 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *arXiv preprint*, 2021. `arXiv:2102.03117`.

4 Jannis Bulian. Parameterized complexity of distances to sparse graph classes. Technical report, University of Cambridge, Computer Laboratory, 2017.

5 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016.

6 Michael Buro. Simple amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs. In *International Conference on Computers and Games*, pages 250–261. Springer, 2000.

7 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

8 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Springer Science & Business Media, 2005.

11 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.

12 Fedor V. Fomin, Petr A. Golovach, Giannos Stamoulis, and Dimitrios M. Thilikos. An algorithmic meta-theorem for graph modification to planarity and FOL. In *28th Annual European Symposium on Algorithms, ESA 2020*, pages 51:1–51:17, 2020.

13 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Parameterized complexity of elimination distance to first-order logic properties. *arXiv preprint*, 2021. `arXiv:2104.02998`.

14 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Hitting topological minors is FPT. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1317–1326, 2020.

15 Robert Ganian, Petr Hliněný, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of MSO1 model-checking. *Journal of Computer and System Sciences*, 80(1):180–194, 2014.

16 Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and its applications*. Springer Science & Business Media, 2007.

17 Martin Grohe. Logic, graphs, and algorithms. *Logic and automata*, 2:357–422, 2008.

18 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM (JACM)*, 64(3):17, 2017.

19 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. In *STACS*, 2020.

20 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1757–1769, 2021.

21 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 189–198. IEEE, 2010.

22 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

23 Alexander Lindermayr, Sebastian Siebertz, and Alexandre Vigny. Elimination distance to bounded degree on planar graphs. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, pages 65:1–65:12, 2020.

24 Michał Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Toruńczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. *arXiv preprint*, 2021. `arXiv:2111.03725`.

25 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

26 Neil Robertson and P. D. Seymour. Graph minors. XIII. the disjoint paths problem. *J. Combin. Theory Ser. B*, 63:65–110, 1995.

27 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. Springer, 1997. `doi:10.1007/978-3-642-59126-6_7`.

Planar Realizability via Left and Right Applications

Haruka Tomita
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

We introduce a class of applicative structures called bi-BDI-algebras. Bi-BDI-algebras are generalizations of partial combinatory algebras and BCI-algebras, and feature two sorts of applications (left and right applications). Applying the categorical realizability construction to bi-BDI-algebras, we obtain monoidal bi-closed categories of assemblies (as well as of modest sets). We further investigate two kinds of comonadic applicative morphisms on bi-BDI-algebras as non-symmetric analogues of linear combinatory algebras, which induce models of exponential and exchange modalities on non-symmetric linear logics.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases Realizability, combinatory algebra, monoidal bi-closed category, exponential modality, exchange modality

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.35

Funding *Haruka Tomita*: This work was supported by JST ERATO Grant Number JPMJER1603, Japan.

Acknowledgements I would like to thank Masahito Hasegawa for a lot of helpful discussions and comments. I am also grateful to Naohiko Hoshino for useful advice. Thanks also to anonymous reviewers for their helpful feedback.

1 Introduction

Categorical realizability gives us a useful method to construct categorical models of various logics and programming languages from simple structures called applicative structures. The most well known is the case of partial combinatory algebras (PCAs), which are an important class of applicative structures. From a PCA $\mathcal{A}$, we can construct Cartesian closed categories (CCCs) $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ and a realizability topos $\mathbf{RT}(\mathcal{A})$ [11].

The structures of such categories obtained by realizability depend on the structures of applicative structures. Thus, assuming other conditions to applicative structures than being PCAs, we can obtain other categorical structures and use them to model other kinds of languages. A well known case is a class of applicative structure called BCI-algebras, which induces a symmetric monoidal closed structure on $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ [1, 2]. While PCAs correspond to the untyped lambda calculus by the combinatory completeness property, BCI-algebras correspond to the untyped linear lambda calculus.

These two cases for PCAs and BCI-algebras are useful to give various models based on CCCs and symmetric monoidal closed categories (SMCCs). On the other hand, the categorical realizability giving rise to non-symmetric categorical structures has not been investigated much. In our previous work [14], several classes of applicative structures that induce certain non-symmetric categorical structures were introduced. The $\mathsf{BI}(-)^{\bullet}$-algebra is one of such classes. A $\mathsf{BI}(-)^{\bullet}$-algebra $\mathcal{A}$ induces the structure of closed multicategories on $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$, and corresponds to the untyped planar lambda calculus by combinatory completeness. Here, the planar lambda calculus is the restricted linear lambda calculus that consists of linear lambda terms whose orders of bound variables can not be freely exchanged. The name "planar" comes from the fact that planar lambda terms correspond to graphically planar maps [16, 15].

The (non-symmetric) closed multicategories obtained from $\mathsf{BI}(-)^\bullet$-algebras can be used for modeling non-symmetric implicational linear logic. However, to model richer non-symmetric logics, we additionally want to express non-symmetric tensor products and it is natural to try to get non-symmetric monoidal closed categories by categorical realizability. Unlike the symmetric case, it is a quite subtle problem to realize non-symmetric tensor products. When we try to realize tensor products by $\mathsf{BI}(-)^\bullet$-algebras in the same way as PCAs and BCI-algebras, we notice that realizers for unitors and associators induces the C-combinator and the tensor products inevitably become symmetric (and the $\mathsf{BI}(-)^\bullet$-algebra is forced to be a BCI-algebra).

This difficulty can be understood by polymorphic encoding. In $\mathbf{Asm}(\mathcal{A})$ for a BCI-algebra $\mathcal{A} = (|\mathcal{A}|, \cdot)$, realizers for an element $x_1 \otimes x_2$ of $X_1 \otimes X_2$ are $\lambda^* z.(z \cdot \mathsf{a}_1 \cdot \mathsf{a}_2)$, where a_i are realizers of x_i respectively. The form of the realizer corresponds to that $X_1 \otimes X_2$ is expressed as $\forall T.(X_1 \multimap X_2 \multimap T) \multimap T$ for symmetric cases. Whereas, for non-symmetric cases, $X_1 \otimes X_2$ is expressed as $\forall T.(T \circ\!\!- X_2 \circ\!\!- X_1) \multimap T$ and we need to distinguish two kinds of implications $\multimap$ and $\circ\!\!-$. In an applicative structure like a $\mathsf{BI}(-)^\bullet$-algebra, elements acting as functions always receive elements acting as arguments from the right side and thus the corresponding types may express only one of $\multimap$ and $\circ\!\!-$.

Conversely, if elements acting as functions can receive elements acting as arguments from both left and right side, we may construct realizers for non-symmetric tensor products. In this paper, we introduce a new structure called bi-BDI-algebra, which has two kinds of applications. These two applications correspond to $\multimap$ and $\circ\!\!-$ respectively and we can realize non-symmetric monoidal structures in $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ on a bi-BDI-algebra $\mathcal{A}$.

Two applications of a bi-BDI-algebra are closely related to each other via its components $\overleftarrow{\mathsf{D}}$, $\vec{\mathsf{D}}$, $(-)^\triangleleft$ and $(-)^\triangleright$, which are introduced in order to let bi-BDI-algebras have certain combinatory completeness property. Thanks to these constructs, for a bi-BDI-algebra $\mathcal{A}$, $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ consequently become monoidal bi-closed categories. Recall that monoidal bi-closed categories are monoidal categories with two kinds of adjunctions $(X \otimes -) \dashv (X \multimap -)$ and $(- \otimes Y) \dashv (- \circ\!\!- Y)$. Natural transformations relating these adjunctions indeed have realizers. In particular, the natural isomorphism $X \multimap (Y \circ\!\!- Z) \cong (X \multimap Y) \circ\!\!- Z$ is realized by $\overleftarrow{\mathsf{D}}$ and $\vec{\mathsf{D}}$.

Furthermore, by the relationship between two applications of bi-BDI-algebras, bi-BDI-algebras can be seen as a non-symmetric generalization of BCI-algebras even though bi-BDI-algebras have additional applications that BCI-algebras do not have. We can reduce a bi-BDI-algebra to a BCI-algebra by assuming certain element expressing symmetry.

In this paper, we further investigate two sorts of modalities on non-symmetric linear logics using bi-BDI-algebras. The linear exponential modality ! allows linear logics to copy and discard arguments [5]. Categorical realizability for the !-modality was introduced in [1], which uses an endomorphism on a BCI-algebra with several extra elements. In [8], the endomorphisms are generalized to total relations with certain conditions, which are generalizations of adjoint pairs between BCI-algebras and PCAs. The generalized endomorphisms give rise to comonads on categories of assemblies (or modest sets), and the comonads model !-modalities. While originally linear exponential comonads are models of !-modalities on linear logics with symmetric tensor products, later, linear exponential comonads for non-symmetric linear logics were also investigated [7]. Just as for the symmetric case, we can obtain comonads modeling !-modalities on non-symmetric linear logics by certain endomorphisms on bi-BDI-algebras. The endomorphisms are generalizations of adjoint pairs between bi-BDI-algebras and PCAs.

The exchange modality introduced in [9] allows non-symmetric linear logics to exchange arguments. A categorical model for the logic with the exchange modality is given by a monoidal adjunction between a monoidal bi-closed category and an SMCC, called a Lambek

adjoint model. We obtain such an adjunction as a co-Kleisli adjunction on categories of assemblies (or modest sets) by the similar way for !-modalities. An adjoint pair between a bi-BDI-algebra and a BCI-algebra gives rise to an endomorphism inducing a Lambek adjoint model.

The rest of this paper is structured as follows. In Section 2, we recall some basic notions and results in categorical realizability. Also we recall three classes of applicative structures: PCAs, BCI-algebras and $\mathsf{BI}(-)^{\bullet}$-algebras, which induce CCCs, SMCCs and closed multicategories respectively. In Section 3, we introduce bi-BDI-algebras and the corresponding lambda calculus. We show that bi-BDI-algebras can be seen as a generalization of BCI-algebras and that bi-BDI-algebras induce monoidal bi-closed categories. In Section 4, we construct models of linear exponential modalities and exchange modalities by categorical realizability. In Section 5, we discuss related work. Finally, in Section 6, we summarize contents of this paper.

Basic knowledge of category theory and the lambda calculus is assumed.

2 Background

2.1 Applicative structures and categories of assemblies

First we recall some basic concepts of the categorical realizability. Notations and definitions in this subsection are from [11].

▶ **Definition 1.** *A* partial applicative structure $\mathcal{A}$ *is a pair of a set* $|\mathcal{A}|$ *and a partial binary operation* $(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x}\cdot\mathsf{y}$ *on* $|\mathcal{A}|$*. When the binary operation of* $\mathcal{A}$ *is total, we say* $\mathcal{A}$ *is a* total applicative structure.

Application associates to the left, and we often omit $\cdot$ and write it as juxtaposition. For instance, $\mathsf{xz(yz)}$ denotes $(\mathsf{x}\cdot\mathsf{z})\cdot(\mathsf{y}\cdot\mathsf{z})$. In the sequel, we use two notations $\downarrow$ and $\simeq$. The down arrow means "defined." For instance, for a partial applicative structure $(|\mathcal{A}|,\cdot)$, $xy\downarrow$ means that $x\cdot y$ is defined. "$\simeq$" denotes the Kleene equality, which means that if the one side of the equation is defined then the other side is also defined and they are equal.

▶ **Definition 2.** *Let* $\mathcal{A}$ *be a partial applicative structure.*

(i) *An* assembly *on* $\mathcal{A}$ *is a pair* $X := (|X|, \|\text{-}\|_X)$*, where* $|X|$ *is a set and* $\|\text{-}\|_X$ *is a function sending* $x \in |X|$ *to a non-empty subset* $\|x\|_X$ *of* $|\mathcal{A}|$*.*

(ii) *For assemblies* X *and* Y*, a* map of assemblies $f : X \to Y$ *is a function* $f : |X| \to |Y|$ *such that there exists an element* $\mathsf{r} \in |\mathcal{A}|$ *realizing* f*. Here "*r realizes f*" or "*r is a realizer of f*" means that* $\forall x \in |X|$, $\forall \mathsf{a} \in \|x\|_X$, $\mathsf{ra}\downarrow$ *and* $\mathsf{ra} \in \|f(x)\|_Y$*.*

If we assume two extra conditions on a partial applicative structure, we can construct two kinds of categories from assemblies and maps of assemblies.

▶ **Definition 3.** *Let* $\mathcal{A}$ *be a partial applicative structure satisfying that:*

1. $|\mathcal{A}|$ *has an element* I *such that for any* $\mathsf{x} \in |\mathcal{A}|$, $\mathsf{Ix}\downarrow$ *and* $\mathsf{Ix} = \mathsf{x}$*;*
2. *for any* $\mathsf{r}_1, \mathsf{r}_2 \in |\mathcal{A}|$*, there exists* $\mathsf{r}_{1,2} \in |\mathcal{A}|$ *such that for any* $\mathsf{x} \in |\mathcal{A}|$, $\mathsf{r}_{1,2}\mathsf{x} \simeq \mathsf{r}_1(\mathsf{r}_2\mathsf{x})$*.*

Then we construct categories as follows:

(i) *The* category $\mathbf{Asm}(\mathcal{A})$ of assemblies *on* $\mathcal{A}$ *consists of assemblies on* $\mathcal{A}$ *as its objects and maps of assemblies as its maps. Identity maps and composition maps are the same as those of* **Sets**.

(ii) *When an assembly* X *satisfies* $\forall x, y \in |X|$, $x \neq y \Rightarrow \|x\|_X \cap \|y\|_X = \emptyset$*, we say that* X *is a* modest set *on* $\mathcal{A}$*. The* category $\mathbf{Mod}(\mathcal{A})$ of modest sets *on* $\mathcal{A}$ *is the full subcategory of* $\mathbf{Asm}(\mathcal{A})$ *whose objects are modest sets on* $\mathcal{A}$*.*

$\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ are indeed categories. For any assembly $(|X|, \|\text{-}\|)$ on $\mathcal{A}$, the identity function on $|X|$ is realized by I. Given two maps of assemblies $X \xrightarrow{f} Y \xrightarrow{g} Z$ realized by r_2 and r_1 respectively, the composition function $g \circ f : |X| \to |Z|$ is realized by $\mathsf{r}_{1,2}$.

The categorical structure of $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ depends on $\mathcal{A}$. When we assume some conditions on $\mathcal{A}$, $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ have certain corresponding categorical structures. In the following three subsections, we introduce three classes of applicative structures: PCAs, BCI-algebras and $\mathsf{BI}(-)^\bullet$-algebras, which induce Cartesian closed categories (CCCs), symmetric monoidal closed categories (SMCCs) and closed multicategories respectively.

2.2 PCAs and Cartesian closed categories

In this subsection, we recall a well-known class of partial applicative structures called partial combinatory algebras (PCAs). PCAs correspond to the lambda calculus and assemblies on a PCA form a CCC. These results are from [11].

▶ **Definition 4.** *A* PCA *is a partial applicative structure $\mathcal{A}$ which contains two elements* S *and* K *satisfying:*

- $\forall \mathsf{x}, \mathsf{y} \in |\mathcal{A}|$, $\mathsf{Kx}\downarrow$, $\mathsf{Kxy}\downarrow$ *and* $\mathsf{Kxy} = \mathsf{x}$*;*
- $\forall \mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$, $\mathsf{Sx}\downarrow$, $\mathsf{Sxy}\downarrow$ *and* $\mathsf{Sxyz} \simeq \mathsf{xz(yz)}$.

▶ **Example 5.** Suppose infinite supply of variables $x, y, z, \ldots$. *Untyped lambda terms* are terms constructed from the following six rules:

$$\frac{}{x \vdash x}\ (\text{identity}) \qquad \frac{\Gamma \vdash M \qquad \Delta \vdash N}{\Gamma, \Delta \vdash MN}\ (\text{application}) \qquad \frac{\Gamma, x \vdash M}{\Gamma \vdash \lambda x.M}\ (\text{abstraction})$$

$$\frac{\Gamma, x, y, \Delta \vdash M}{\Gamma, y, x, \Delta \vdash M}\ (\text{exchange}) \qquad \frac{\Gamma, x, y \vdash M}{\Gamma, x \vdash M[x/y]}\ (\text{contraction}) \qquad \frac{\Gamma \vdash M}{\Gamma, x \vdash M}\ (\text{weakening})$$

Here, in the application rule, Γ and Δ are sequences of distinct variables and contain no common variables. In the contraction rule, $M[x/y]$ denotes the term obtained by substituting x for all free y in M. In the weakening rule, x is a variable not contained in Γ.

Note that abstraction rules are only applied to the rightmost variables. In order to apply the abstraction rule to a variable in a different position, we need to use exchange rules and move the variable to the rightmost place.

We define *β-equivalence relation* $=_\beta$ on lambda terms as the congruence of the relation $(\lambda x.M)N \sim M[N/x]$. Untyped lambda terms modulo $=_\beta$ form a PCA. The underlying set of the PCA consists of β-equivalence classes of untyped closed lambda terms (i.e., lambda terms with no free variables) and the application is defined as that of lambda terms. In this example, $\lambda xyz.xz(yz)$ is the representative of S and $\lambda xy.x$ is the representative of K.

PCAs are closely related to the untyped lambda calculus through the property called *combinatory completeness* as in Proposition 7.

▶ **Definition 6.** *Let $\mathcal{A}$ be a partial applicative structure. A* polynominal *over $\mathcal{A}$ is a syntactic expression generated by variables, elements of $|\mathcal{A}|$ and applications. For polynominals M and N over $\mathcal{A}$, $M \simeq N$ means that $M[\mathsf{a}_1/x_1, ..., \mathsf{a}_n/x_n] \simeq N[\mathsf{a}_1/x_1, ..., \mathsf{a}_n/x_n]$ holds in $\mathcal{A}$ for any $\mathsf{a}_1, ..., \mathsf{a}_n \in |\mathcal{A}|$, where $\{x_1, ..., x_n\}$ contains all the variables of M and N.*

▶ **Proposition 7** (combinatory completeness for PCAs)**.** *Let $\mathcal{A}$ be a PCA and M be a polynominal over $|\mathcal{A}|$. For any variable x, there exists a polynominal M' such that the free variables of M' are the free variables of M excluding x and $M'\mathsf{a} \simeq M[\mathsf{a}/x]$ for all $\mathsf{a} \in |\mathcal{A}|$. We write $\lambda^* x.M$ for such M'.*

Proof. We define $\lambda^* x.M$ by induction on the structure of M as follows: $\lambda^* x.x := \mathsf{SKK}$; $\lambda^* x.y := \mathsf{K}y$ (when $x \neq y$); $\lambda^* x.MN := \mathsf{S}(\lambda^* x.M)(\lambda^* x.N)$. ◀

For the special case of the above proposition, any closed lambda term is β-equivalent to some term constructed from $\lambda xyz.xz(yz)$ and $\lambda xy.x$ only using applications.

Although conditions of PCAs are simple, categorical structures induced by PCAs are quite strong and useful.

▶ **Proposition 8.** *Let $\mathcal{A}$ be a PCA. Then $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ are Cartesian closed and regular.*

The detailed proof is in [11]. What is important here is how to realize products and exponents by PCAs. Cartesian closed structures in $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ are defined as follows. The terminal object is $(\{*\}, \|\text{-}\|_1)$, where $\|*\|_1 := |\mathcal{A}|$. For objects X and Y, the product is $(|X| \times |Y|, \|\text{-}\|_{X\times Y})$, where $\|(x,y)\|_{X\times Y} := \{\lambda^* z.z\mathsf{a}\mathsf{a}' \mid \mathsf{a} \in \|x\|_X,\ \mathsf{a}' \in \|y\|_Y\}$. Projection maps are realized by K and $\lambda^* xy.y$. For objects X and Y, the exponent is defined as $(Hom_{\mathbf{Asm}(\mathcal{A})}(X,Y), \|\text{-}\|_{Y^X})$, where $\|f\|_{Y^X} := \{\mathsf{r} \mid \mathsf{r} \text{ realizes } f\}$. The evaluation map $ev : X \times Y^X \to Y$ is realized by $\lambda^* z.z(\lambda^* uv.vu)$.

2.3 BCI-algebras and symmetric monoidal closed categories

In this subsection we recall another class of applicative structures called BCI-algebra. BCI-algebras are related to linear structures whereas PCAs are not. The results given below are from [2, 8].

▶ **Definition 9.** *A* BCI-algebra *is a total applicative structure $\mathcal{A}$ which contains three elements* B, C *and* I *such that for any* $\mathsf{x},\mathsf{y},\mathsf{z} \in |\mathcal{A}|$, $\mathsf{Bxyz} = \mathsf{x}(\mathsf{yz})$, $\mathsf{Cxyz} = \mathsf{xzy}$ *and* $\mathsf{Ix} = \mathsf{x}$.

▶ **Example 10.** *Untyped linear lambda terms* are untyped lambda terms constructed without using weakening and contraction rules, i.e., untyped lambda terms whose each variable appears just once. Untyped linear lambda terms modulo $=_\beta$ form a BCI-algebra. Here $\lambda xyz.x(yz)$, $\lambda xyz.xzy$ and $\lambda x.x$ are the representatives of B, C and I respectively.

▶ **Proposition 11** (combinatory completeness for BCI-algebras)**.** *Let $\mathcal{A}$ be a* BCI*-algebra and M be a polynominal over $|\mathcal{A}|$. For any variable x appearing exactly once in M, there exists a polynominal $\lambda^* x.M$ such that the free variables of $\lambda^* x.M$ are the free variables of M excluding x and $(\lambda^* x.M)\mathsf{a} = M[\mathsf{a}/x]$ for all $\mathsf{a} \in |\mathcal{A}|$.*

Proof. We define $\lambda^* x.M$ by induction on the structure of M as follows:

- $\lambda^* x.x := \mathsf{I}$
- $\lambda^* x.MN := \begin{cases} \mathsf{C}(\lambda^* x.M)N & (x \in FV(M)) \\ \mathsf{B}M(\lambda^* x.N) & (x \in FV(N)) \end{cases}$ ◀

For the special case of the above proposition, any closed linear lambda term is β-equivalent to some term constructed from $\lambda xyz.x(yz)$, $\lambda xyz.xzy$ and $\lambda x.x$ only using applications.

Since BCI-algebras are related to the linear lambda calculus, categorical structures of assemblies on BCI-algebras are also linear ones.

▶ **Proposition 12.** *Let $\mathcal{A}$ be a* BCI*-algebra. Then $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ are SMCCs.*

The monoidal structure in $\mathbf{Asm}(\mathcal{A})$ are defined as follows: The unit object is $(\{*\}, \|\text{-}\|_I)$, where $\|*\|_I := \{\mathsf{I}\}$. For objects X and Y, the tensor product is defined as $(|X| \times |Y|, \|\text{-}\|_{X\otimes Y})$, where $\|(x,y)\|_{X\otimes Y} := \{\lambda^* z.z\mathsf{a}\mathsf{a}' \mid \mathsf{a} \in \|x\|_X,\ \mathsf{a}' \in \|y\|_Y\}$. Realizers for natural isomorphisms

follows by the combinatory completeness. For instance, the associator $a : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$ is realized by $\lambda^* w.w(\lambda^* w' r.w'(\lambda^* pq.(\lambda^* u.up(\lambda^* v.vqr))))$. The monoidal structures of $\mathbf{Mod}(\mathcal{A})$ are not the same as $\mathbf{Asm}(\mathcal{A})$, since $X \otimes Y$ may not be a modest set even if X and Y are modest. We can define the monoidal structures of $\mathbf{Mod}(\mathcal{A})$ by the monoidal adjunction associated with $\mathbf{Mod}(\mathcal{A})$ being a reflective fullsubcategory of $\mathbf{Asm}(\mathcal{A})$. (See Section 3 in [8].)

2.4 $\mathsf{BI}(-)^\bullet$-algebras and closed multicategories

In this subsection we give another class of applicative structure called $\mathsf{BI}(-)^\bullet$-algebras. They are generalizations of BCI-algebras by excluding C that expresses exchanging arguments. $\mathsf{BI}(-)^\bullet$-algebras are related to non-symmetric linear structures. These results are from [14].

▶ **Definition 13.** *We say that a total applicative structure $\mathcal{A}$ is a* $\mathsf{BI}(-)^\bullet$-algebra *iff it contains* B, I *and* $\mathsf{a}^\bullet$ *for each* $\mathsf{a} \in |\mathcal{A}|$, *where* $\mathsf{a}^\bullet$ *is an element of* $|\mathcal{A}|$ *such that* $\mathsf{a}^\bullet \mathsf{x} = \mathsf{xa}$ *for all* $\mathsf{x} \in |\mathcal{A}|$.

Like PCAs and BCI-algebras, $\mathsf{BI}(-)^\bullet$-algebras are related to a part of the untyped lambda calculus, which is called the *planar lambda calculus*.

▶ **Example 14.** *Untyped planar lambda terms* are untyped lambda terms constructed without using weakening, contraction nor exchange rules. Untyped closed planar lambda terms modulo $=_\beta$ form a $\mathsf{BI}(-)^\bullet$-algebra. Here $\lambda xyz.x(yz)$ and $\lambda x.x$ are the representatives of B and I. Given a representative M of a, $\lambda x.xM$ is also a closed planar term and is the representative of $\mathsf{a}^\bullet$.

▶ **Proposition 15** (combinatory completeness for $\mathsf{BI}(-)^\bullet$-algebras)**.** *Let $\mathcal{A}$ be a $\mathsf{BI}(-)^\bullet$-algebra and M be a polynominal over $|\mathcal{A}|$. For the rightmost variable x, which has to appear exactly once in M, there exists a polynominal $\lambda^* x.M$ such that the free variables of $\lambda^* x.M$ are the free variables of M in the same order excluding x and $(\lambda^* x.M)\mathsf{a} = M[\mathsf{a}/x]$ for all $\mathsf{a} \in |\mathcal{A}|$.*

Proof. We define $\lambda^* x.M$ by induction on the structure of M as follows:

- $\lambda^* x.x := \mathsf{I}$
- $\lambda^* x.MN := \begin{cases} \mathsf{B}N^\bullet(\lambda^* x.M) & (x \in FV(M)) \\ \mathsf{B}M(\lambda^* x.N) & (x \in FV(N)) \end{cases}$

Note that for $\lambda^* x.MN$, x is the rightmost free variable in MN. Therefore, if x is in $FV(M)$, N has no free variables and $N^\bullet$ can be defined. ◀

For the special case of the above proposition, any closed planar lambda term is β-equivalent to some term constructed from $\lambda xyz.x(yz)$ and $\lambda x.x$ using applications and the unary operation $(-)^\bullet : M \mapsto \lambda x.xM$.

$\mathsf{BI}(-)^\bullet$-algebras are related to the planar lambda calculus, that is, a fragment of the linear lambda calculus without symmetry. Therefore, categorical structures of assemblies on $\mathsf{BI}(-)^\bullet$-algebras also are non-symmetric and linear.

▶ **Proposition 16.** *For a* $\mathsf{BI}(-)^\bullet$*-algebra $\mathcal{A}$,* $\mathbf{Asm}(\mathcal{A})$ *and* $\mathbf{Mod}(\mathcal{A})$ *are closed multicategories.*

Here closed multicategories are one of generalizations of categories with objects expressing internal hom [12]. What we need to refer here is that closed multicategories are generalization of monoidal closed categories and they not generally have tensors. When we try to construct realizers for tensor products in $\mathbf{Asm}(\mathcal{A})$ as we did in Proposition 8 and Proposition 12 ($\|x \otimes y\| := \{\lambda^* z.z\mathsf{aa}' \mid \mathsf{a} \in \|x\|, \mathsf{a}' \in \|y\|\}$), we notice that realizers for associators and unitors do not generally exist. Even if we assume these realizers in a $\mathsf{BI}(-)^\bullet$-algebra, we can

construct an element acting as C from these realizers. Take an assembly X as $|X| := |\mathcal{A}|$ and $\|\mathsf{a}\|_X := \{\mathsf{a}\}$. Assuming that the unitor $X \to I \otimes X$ has a realizer r, $\mathsf{rxz} = \mathsf{zIx}$ holds for any x and z. Let $C' := \lambda^* xz.\mathsf{r}x(\mathsf{B}z)$ and $C := \lambda^* xy.C'x(\mathsf{B}(C'y))$. Then C satisfies the axiom of the C-combinator and thus $\mathcal{A}$ inevitably becomes a BCI-algebra.

2.5 Applicative morphisms

In this subsection, we recall the notion of applicative morphisms from [11]. In [11], applicative morphisms are defined not for arbitrary applicative structures but only for PCAs. However, the same definition makes sense for a large class of applicative structures including PCAs, BCI-algebras and $\mathsf{BI}(-)^\bullet$-algebras.

In this subsection, let $\mathcal{A}$ and $\mathcal{B}$ range over PCAs and $\mathsf{BI}(-)^\bullet$-algebras.

▶ **Definition 17.** *An* applicative morphism $\gamma : \mathcal{A} \to \mathcal{B}$ *is a total relation from* $|\mathcal{A}|$ *to* $|\mathcal{B}|$ *such that there is a* realizer $\mathsf{r}_\gamma \in |\mathcal{B}|$ *of* γ *satisfying that for any* $\mathsf{x}, \mathsf{x}' \in |\mathcal{A}|$, $\mathsf{y} \in \gamma\mathsf{x}$ *and* $\mathsf{y}' \in \gamma\mathsf{x}'$, $\mathsf{r}_\gamma\mathsf{y}\mathsf{y}' \in \gamma(\mathsf{x}\mathsf{x}')$ *holds whenever* $\mathsf{x}\mathsf{x}' \downarrow$. *(We often write such a condition as* $\mathsf{r}_\gamma(\gamma\mathsf{x})(\gamma\mathsf{x}') \subseteq \gamma(\mathsf{x}\mathsf{x}')$*).*

▶ **Definition 18.** *For two applicative morphisms* $\gamma, \delta : \mathcal{A} \to \mathcal{B}$, $\gamma \preceq \delta$ *iff there exists an element* $\mathsf{r} \in |\mathcal{B}|$ *called* realizer *of* $\gamma \preceq \delta$ *such that* $\mathsf{ry} \in \delta\mathsf{x}$ *for any* $\mathsf{x} \in |\mathcal{A}|$ *and* $\mathsf{y} \in \gamma\mathsf{x}$.

By the preorder $\preceq$, we can define adjunctions and comonads on applicative structures.

▶ **Definition 19.** *For applicative morphisms* $\gamma : \mathcal{A} \to \mathcal{B}$ *and* $\delta : \mathcal{B} \to \mathcal{A}$, γ *is a* right adjoint *of* δ *iff* $\delta \circ \gamma \preceq id_{\mathcal{A}}$ *and* $id_{\mathcal{B}} \preceq \gamma \circ \delta$. *We often write these settings as* $(\delta \dashv \gamma) : \mathcal{A} \to \mathcal{B}$.

▶ **Definition 20.** *We say an applicative morphism* $\gamma : \mathcal{A} \to \mathcal{A}$ *is a* comonadic applicative morphism *when* $\mathcal{A}$ *has two element* e *and* d *such that* $\mathsf{e}(\gamma\mathsf{x}) \subseteq \{\mathsf{x}\}$ *and* $\mathsf{d}(\gamma\mathsf{x}) \subseteq \gamma(\gamma\mathsf{x})$ *for any* $\mathsf{x} \in |\mathcal{A}|$.

Given an adjoint pair of applicative morphisms $(\delta \dashv \gamma) : \mathcal{A} \to \mathcal{B}$, we obtain a comonadic applicative morphism $(\delta \circ \gamma) : \mathcal{A} \to \mathcal{A}$. e is given as a realizer of $\delta \circ \gamma \preceq id_{\mathcal{A}}$ and d is given as $\mathsf{r}_\delta(\delta\mathsf{r})$, where r_δ is a realizer of δ and r is a realizer of $id_{\mathcal{B}} \preceq \gamma \circ \delta$.

Any applicative morphism $\gamma : \mathcal{A} \to \mathcal{B}$ gives rise to a functor $\gamma_* : \mathbf{Asm}(\mathcal{A}) \to \mathbf{Asm}(\mathcal{B})$. Furthermore, any adjoint pair $(\delta \dashv \gamma) : \mathcal{A} \to \mathcal{B}$ gives rise to an adjunction $\delta_* \dashv \gamma_*$.

▶ **Definition 21.** *For an applicative morphism* $\gamma : \mathcal{A} \to \mathcal{B}$, $\gamma_* : \mathbf{Asm}(\mathcal{A}) \to \mathbf{Asm}(\mathcal{B})$ *is the functor sending an object* $(|X|, \|\text{-}\|_X)$ *to* $(|X|, \gamma\|\text{-}\|_X)$ *and sending a map* $f : X \to Y$ *to the same function.*

The realizer of $\gamma_* f$ exists in $\mathsf{r}_\gamma(\gamma\mathsf{r}_f)$, where r_γ and r_f are realizers of γ and f respectively.

▶ **Proposition 22.** *An adjoint pair of applicative morphisms* $(\delta \dashv \gamma) : \mathcal{A} \to \mathcal{B}$ *gives rise to an adjunction* $\delta_* \dashv \gamma_* : \mathbf{Asm}(\mathcal{A}) \to \mathbf{Asm}(\mathcal{B})$.

The unit and counit are realized by the realizers of $id_{\mathcal{B}} \preceq \gamma \circ \delta$ and $\delta \circ \gamma \preceq id_{\mathcal{A}}$ respectively.

For a comonadic applicative morphism $\gamma : \mathcal{A} \to \mathcal{A}$, γ_* is a comonad on $\mathbf{Asm}(\mathcal{A})$. The counit is realized by e and the comultiplication is realized by d.

This subsection can be summarized as follows. PCAs and $\mathsf{BI}(-)^\bullet$-algebras form a preorder enriched category and $\mathbf{Asm}(-)$ extends to a 2-functor from this 2-category to the 2-category of categories of assemblies (on PCAs and $\mathsf{BI}(-)^\bullet$-algebras). Details for the 2-functor $\mathbf{Asm}(-)$ (for PCAs) are in Section 2.2 of [11].

▶ Remark 23. Given an applicative morphism $\gamma : \mathcal{A} \to \mathcal{B}$, we can not generally obtain a functor $\gamma_* : \mathbf{Mod}(\mathcal{A}) \to \mathbf{Mod}(\mathcal{B})$ as the same for categories of assemblies, since $\|x\|_X \cap \|x'\|_X = \emptyset$ does not imply $\gamma(\|x\|_X) \cap \gamma(\|x'\|_X) = \emptyset$ and $\gamma_* X$ may not be in $\mathbf{Mod}(\mathcal{B})$. However, for a comocadic applicative morphism $\gamma : \mathcal{A} \to \mathcal{A}$, γ_* can be restricted to an endofunctor on $\mathbf{Mod}(\mathcal{A})$. Indeed, for a modest set X, if $\mathsf{a} \in (\gamma\|x\|_X) \cap (\gamma\|x'\|_X)$ then $\mathsf{ea} \in \|x\|_X \cap \|x'\|_X$ and thus $x = x'$. Just like for $\mathbf{Asm}(\mathcal{A})$, the γ_* is a comonad on $\mathbf{Mod}(\mathcal{A})$.

3 Bi-BDI-algebras and monoidal bi-closed categories

As we said in Section 2.4, it is difficult to construct non-symmetric monoidal structure on $\mathbf{Asm}(\mathcal{A})$ by $\mathsf{BI}(-)^\bullet$-algebras in the same way as BCI-algebras and PCAs. We need some major modification on the definition of realizers of tensor products in $\mathbf{Asm}(\mathcal{A})$.

Here we introduce a new class of applicative structures called bi-BDI-algebras, which are very different from classes of applicative structures we have seen so far since bi-BDI-algebras contain two sorts of applications. We use the two applications to realize tensor products while avoiding intrusion of C-combinators.

3.1 Bi-BDI-algebras and the bi-planar lambda calculus

First we introduce a variant of the lambda calculus that we call the *bi-planar lambda calculus* here, which contains two sides of applications and abstractions[1].

▶ **Definition 24.** Bi-planar lambda terms *are constructed by the following rules:*

$$\frac{}{x \vdash x}\ \textit{(identity)} \qquad \frac{\Gamma, x \vdash M}{\Gamma \vdash (M \leftarrow\!\!\shortmid x)}\ \textit{(right abstraction)} \qquad \frac{x, \Gamma \vdash M}{\Gamma \vdash (x \mapsto\!\!\shortmid M)}\ \textit{(left abstraction)}$$

$$\frac{\Gamma \vdash M \qquad \Delta \vdash N}{\Gamma, \Delta \vdash M \mathbin{\overleftarrow{@}} N}\ \textit{(right application)} \qquad \frac{\Gamma \vdash M \qquad \Delta \vdash N}{\Gamma, \Delta \vdash M \mathbin{\vec{@}} N}\ \textit{(left application)}$$

Here is none of weakening, contraction nor exchange rules.

Although bi-planar lambda terms seem very different from ordinary lambda terms, when we construct terms only using identity, right application and right abstraction rules, they are planar lambda terms. In this case $M \mathbin{\overleftarrow{@}} N$ denotes MN and $(M \leftarrow\!\!\shortmid x)$ denotes $\lambda x.M$.

For the sake of clarity, we classify right and left by red and blue. That is, we write each of them as $M \mathbin{\overleftarrow{@}} N$, $(M \leftarrow\!\!\shortmid x)$, $N \mathbin{\vec{@}} M$ and $(x \mapsto\!\!\shortmid M)$.

▶ **Definition 25.** *We define a relation $\to_\beta$ on bi-planar lambda terms as the congruence of the following relations:*

- *(right β-reduction)* $(M \leftarrow\!\!\shortmid x) \mathbin{\overleftarrow{@}} N \to_\beta M[N/x]$
- *(left β-reduction)* $N \mathbin{\vec{@}} (x \mapsto\!\!\shortmid M) \to_\beta M[N/x]$

The bi-planar lambda calculus *consists of bi-planar lambda terms and the reflexive, symmetric and transitive closure of $\to_\beta$ as the equational relation $=_\beta$.*

Basic properties about $\to_\beta$, such as the confluence and the strongly normalizing property, can be shown in the same way as the proof for the linear lambda calculus.

[1] The terminology left and right abstractions corresponds to left and right closed structures of monoidal categories: $(X \otimes -) \dashv (X \multimap -)$ and $(- \otimes Y) \dashv (- \circ\!\!- Y)$.

▶ Remark 26. The bi-planar lambda calculus is essentially not a new concept, since it often appears as the Curry-Howard corresponding calculus with the Lambek calculus (cf. [9]). However, note that unlike the calculus corresponding to the Lambek calculus, the bi-planar lambda calculus is based on untyped setting. The reason why we use a less-standard notation is to shorten the length of the realizers and to make them easier to read.

Next we introduce the notion of bi-BDI-algebra, which corresponds to the bi-planar lambda calculus. In order to express the bi-planar lambda calculus by an algebraic structure, the structure is not enough to have two sides of applications, but also need some conditions for relating these two applications. In bi-BDI-algebras, $\overleftarrow{\mathsf{D}}$, $\vec{\mathsf{D}}$, $(-)^{\triangleleft}$ and $(-)^{\triangleright}$ express such conditions.

▶ **Definition 27.** *We say a total applicative structure* $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@})$ *is a* bi-BDI-algebra *when there is an additional total binary operation* $\vec{@}$ *on* $|\mathcal{A}|$ *and* $|\mathcal{A}|$ *contains several special elements:*

- $\overleftarrow{\mathsf{B}} \in |\mathcal{A}|$ *such that* $((\overleftarrow{\mathsf{B}} \overleftarrow{@} \mathsf{x}) \overleftarrow{@} \mathsf{y}) \overleftarrow{@} \mathsf{z} = \mathsf{x} \overleftarrow{@} (\mathsf{y} \overleftarrow{@} \mathsf{z})$ *for any* $\mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$.
- $\vec{\mathsf{B}} \in |\mathcal{A}|$ *such that* $\mathsf{z} \vec{@} (\mathsf{y} \vec{@} (\mathsf{x} \vec{@} \vec{\mathsf{B}})) = (\mathsf{z} \vec{@} \mathsf{y}) \vec{@} \mathsf{x}$ *for any* $\mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$.
- $\overleftarrow{\mathsf{D}} \in |\mathcal{A}|$ *such that* $\mathsf{x} \vec{@} ((\overleftarrow{\mathsf{D}} \overleftarrow{@} \mathsf{y}) \overleftarrow{@} \mathsf{z}) = (\mathsf{x} \vec{@} \mathsf{y}) \overleftarrow{@} \mathsf{z}$ *for any* $\mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$.
- $\vec{\mathsf{D}} \in |\mathcal{A}|$ *such that* $(\mathsf{z} \vec{@} (\mathsf{y} \vec{@} \vec{\mathsf{D}})) \overleftarrow{@} \mathsf{x} = \mathsf{z} \vec{@} (\mathsf{y} \overleftarrow{@} \mathsf{x})$ *for any* $\mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$.
- $\overleftarrow{\mathsf{I}} \in |\mathcal{A}|$ *such that* $\overleftarrow{\mathsf{I}} \overleftarrow{@} \mathsf{x} = \mathsf{x}$ *for any* $\mathsf{x} \in |\mathcal{A}|$.
- $\vec{\mathsf{I}} \in |\mathcal{A}|$ *such that* $\mathsf{x} \vec{@} \vec{\mathsf{I}} = \mathsf{x}$ *for any* $\mathsf{x} \in |\mathcal{A}|$.
- *For each* $\mathsf{a} \in |\mathcal{A}|$, $\mathsf{a}^{\triangleleft} \in |\mathcal{A}|$ *such that* $(\mathsf{a}^{\triangleleft}) \overleftarrow{@} \mathsf{x} = \mathsf{x} \vec{@} \mathsf{a}$ *for any* $\mathsf{x} \in |\mathcal{A}|$.
- *For each* $\mathsf{a} \in |\mathcal{A}|$, $\mathsf{a}^{\triangleright} \in |\mathcal{A}|$ *such that* $\mathsf{x} \vec{@} (\mathsf{a}^{\triangleright}) = \mathsf{a} \overleftarrow{@} \mathsf{x}$ *for any* $\mathsf{x} \in |\mathcal{A}|$.

We call $\overleftarrow{@}$ *and* $\vec{@}$ *as* right application *and* left application *respectively. In the sequel, we use* $\overleftarrow{@}$ *as a left-associative operation and often omit unnecessary parentheses, while we do not omit parentheses for* $\vec{@}$.

As can be seen from the definition, though the left application is an extra component in a bi-BDI-algebra, the conditions required for left and right applications are dual. We often write $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@}, \vec{@})$ as a bi-BDI-algebra $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@})$ with the left application $\vec{@}$.

▶ Remark 28. Here we deal with bi-BDI-algebras only as total applicative structures while we can define *partial* bi-BDI-algebras. Given a partial bi-BDI-algebra $\mathcal{A}$, we always can extend $\mathcal{A}$ to a total bi-BDI-algebra $\mathcal{A}'$ by adding an extra element expressing "undefined." Then $\mathbf{Asm}(\mathcal{A})$ is a full subcategory of $\mathbf{Asm}(\mathcal{A}')$. The same discussion for partial BCI-algebras is in Remark 1 of [8].

▶ **Example 29.** Closed bi-planar lambda terms modulo $=_\beta$ form a bi-BDI-algebra. For instance, $(((x \overleftarrow{@} (y \overleftarrow{@} z)) \leftarrow\!\shortmid z) \leftarrow\!\shortmid y) \leftarrow\!\shortmid x)$, $(((x \mapsto (x \vec{@} y) \overleftarrow{@} z) \leftarrow\!\shortmid z) \leftarrow\!\shortmid y)$ and $(x \mapsto M \overleftarrow{@} x)$ are the representative of $\overleftarrow{\mathsf{B}}$, $\overleftarrow{\mathsf{D}}$ and $M^{\triangleright}$.

Combinatory completeness holds for bi-BDI-algebras and the bi-planar lambda calculus.

▶ **Proposition 30** (Combinatory completeness for bi-BDI-algebras)**.** *Let* $\mathcal{A}$ *be a bi-*BDI*-algebra. We define a* polynominal over $\mathcal{A}$ *as a syntactic expression generated by variables, elements of* $|\mathcal{A}|$ *and left and right applications of* $\mathcal{A}$*. Suppose a polynominal* M *over* $\mathcal{A}$ *and the rightmost variable* x *appears only once in* M*. There exists a polynominal* M' *such that the free variables of* M' *are the free variables of* M *excluding* x *and* $M' \overleftarrow{@} \mathsf{a} = M'[\mathsf{a}/x]$ *for any* $\mathsf{a} \in |\mathcal{A}|$*. We write* $(M \leftarrow\!\shortmid x)$ *for such* M'*. Also, if the leftmost variable* x *appears only once in* M*, there exists a polynominal* M'' *such that the free variables of* M'' *are the free variables of* M *excluding* x *and* $\mathsf{a} \vec{@} M'' = M''[\mathsf{a}/x]$ *for any* $\mathsf{a} \in |\mathcal{A}|$*. We write* $(x \mapsto M)$ *for such* M''.

Proof. We define $(x{\mapsto}M)$ by induction on the structure of M.

- $(x{\mapsto}x) := \vec{\mathsf{I}}$
- $(x{\mapsto}(N\,\vec{@}\,M)) := \begin{cases} (x{\mapsto}M)\,\vec{@}\,((N\,\vec{@}\,(\overleftarrow{\mathsf{I}}\,\vec{@}\,\vec{\mathsf{D}}))^{\triangleright}\,\vec{@}\,\vec{\mathsf{B}}) & (x \in FV(M)) \\ (x{\mapsto}N)\,\vec{@}\,(M\,\vec{@}\,\vec{\mathsf{B}}) & (x \in FV(N)) \end{cases}$
  For the case $x \in FV(M)$, x is the leftmost free variable of $N\,\vec{@}\,M$. Hence N contains no variables and $(N\,\vec{@}\,(\overleftarrow{\mathsf{I}}\,\vec{@}\,\vec{\mathsf{D}}))^{\triangleright}$ can be defined.
- $(x{\mapsto}(M\,\overleftarrow{@}\,N)) := \begin{cases} (\overleftarrow{\mathsf{D}}\,\overleftarrow{@}\,(x{\mapsto}M))\,\overleftarrow{@}\,N & (x \in FV(M)) \\ (x{\mapsto}N)\,\vec{@}\,((M^{\triangleright})\,\vec{@}\,\vec{\mathsf{B}}) & (x \in FV(N)) \end{cases}$
  For the case $x \in FV(N)$, x is the leftmost free variable of $M\,\overleftarrow{@}\,N$. Hence M contains no variables and $M^{\triangleright}$ can be defined.

The case of the right abstractions $(M{\leftarrow}x)$ is given in the same way, with all the left and right constructs exchanged. ◀

It immediately follows that for any bi-BDI-algebra $(|\mathcal{A}|, \overleftarrow{@}, \vec{@})$, the applicative structure $(|\mathcal{A}|, \overleftarrow{@})$ is a $\mathsf{BI}(-)^{\bullet}$-algebra since $\mathsf{a}^{\bullet}$ can be defined as $(x\,\overleftarrow{@}\,\mathsf{a}{\leftarrow}x)$.

We can show that the left application of a bi-BDI-algebra is unique up to isomorphism.

▶ **Proposition 31.** *Suppose that $\mathcal{A}_1 = (|\mathcal{A}|, \overleftarrow{@}, \vec{@}_1)$ and $\mathcal{A}_2 = (|\mathcal{A}|, \overleftarrow{@}, \vec{@}_2)$ are bi-BDI-algebras. Then $(|\mathcal{A}|, \overleftarrow{@}_1)$ and $(|\mathcal{A}|, \overleftarrow{@}_2)$ are isomorphic as applicative structures, where $x\,\overleftarrow{@}_i\,y := y\,\vec{@}_i\,x$.*

To the end of this subsection, we additionally give an example of a bi-BDI-algebra.

▶ **Example 32.** Take an ordered group $(G, \cdot, e, \leq)$. Let T be the set of terms t constructed as follows:

$$t ::= g \mid t \circ\!\!- t \mid t -\!\!\circ t \quad (g \in G).$$

We define a function $|\text{-}| : T \to G$ by induction: $|g| := g$, $|t_1 \circ\!\!- t_2| := |t_1| \cdot |t_2|^{-1}$ and $|t_2 -\!\!\circ t_1| := |t_2|^{-1} \cdot |t_1|$.

Let $\mathcal{T}$ be the powerset of $\{t \in T \mid e \leq |t|\}$. Then $\mathcal{T}$ forms a bi-BDI-algebra.

- For $M, N \in \mathcal{T}$, $M\,\overleftarrow{@}\,N := \{t_1 \mid \exists t_2 \in N, (t_1 \circ\!\!- t_2) \in M\}$.
- For $M, N \in \mathcal{T}$, $N\,\vec{@}\,M := \{t_1 \mid \exists t_2 \in N, (t_2 -\!\!\circ t_1) \in M\}$.
- $\overleftarrow{\mathsf{B}} := \{((t_1 \circ\!\!- t_3) \circ\!\!- (t_2 \circ\!\!- t_3)) \circ\!\!- (t_1 \circ\!\!- t_2) \mid t_1, t_2, t_3 \in T\}$, dual for $\vec{\mathsf{B}}$.
- $\overleftarrow{\mathsf{D}} := \{((t_1 -\!\!\circ t_2) \circ\!\!- t_3) \circ\!\!- (t_1 -\!\!\circ (t_2 \circ\!\!- t_3)) \mid t_1, t_2, t_3 \in T\}$, dual for $\vec{\mathsf{D}}$.
- $\overleftarrow{\mathsf{I}} := \{t_1 -\!\!\circ t_1 \mid t_1 \in T\}$, same for $\vec{\mathsf{I}}$.
- For $M \in \mathcal{T}$, $M^{\triangleleft} := \{t_1 \circ\!\!- t_2 \mid (t_2 -\!\!\circ t_1) \in M\}$, same for $M^{\triangleright}$.

The construction of this example is the same one as that for a $\mathsf{BI}(-)^{\bullet}$-algebra in Section 6 of [14], which is based on a reflexive object of a pivotal category $\mathcal{C}omod(\overline{G})$ introduced in [6].

3.2 Bi-BDI-algebras and BCI-algebras

In this subsection, we show that bi-BDI-algebras can be seen as non-commutative generalizations of BCI-algebras. First, we show that the BCI-algebra is a special case of the bi-BDI-algebra.

▶ **Proposition 33.** *Let $\mathcal{B} = (|\mathcal{B}|, \cdot)$ be a BCI-algebra. When we take two binary operations $\vec{@}$ and $\overleftarrow{@}$ by $y\,\vec{@}\,x = x\,\overleftarrow{@}\,y := x \cdot y$, $(|\mathcal{B}|, \overleftarrow{@}, \vec{@})$ is a bi-BDI-algebra.*

Proof. B satisfies axioms for $\vec{\mathsf{B}}$ and $\overleftarrow{\mathsf{B}}$. I satisfies axioms for $\vec{\mathsf{I}}$ and $\overleftarrow{\mathsf{I}}$. C satisfies axioms for $\overleftarrow{\mathsf{D}}$ and $\vec{\mathsf{D}}$. x satisfies axioms for $\mathsf{x}^{\triangleright}$ and $\mathsf{x}^{\triangleleft}$. ◀

Conversely, not all bi-BDI-algebras are BCI-algebras. Indeed, the bi-planar lambda calculus is not a BCI-algebra. Here we show that when one of the left/right applicative structures of a bi-BDI-algebra is a BCI-algebra, so is the other applicative structure and moreover they coincide.

▶ **Proposition 34.** *Let $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@}, \vec{@})$ be a bi-*BDI*-algebra. Take an applicative structures $\mathcal{A}' = (|\mathcal{A}|, \overleftarrow{@}')$ by $\mathsf{x} \overleftarrow{@}' \mathsf{y} := \mathsf{y} \vec{@} \mathsf{x}$. Then $\mathcal{A}$ is a* BCI*-algebra iff $\mathcal{A}'$ is a* BCI*-algebra. Moreover, in such a case, these* BCI*-algebras are isomorphic as applicative structures.*

3.3 Bi-BDI-algebras and monoidal bi-closed categories

Now we show the main result that bi-BDI-algebras induce monoidal bi-closed categories. For a bi-BDI-algebra $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@}, \vec{@})$, $\mathbf{Asm}(\mathcal{A})$ is constructed by considering $\mathcal{A}$ as an applicative structure $(|\mathcal{A}|, \overleftarrow{@})$.

▶ **Proposition 35.** *For a bi-*BDI*-algebra $\mathcal{A}$, $\mathbf{Asm}(\mathcal{A})$ is a monoidal bi-closed category.*

Proof. (Sketch)

- For objects X and Y, the underlying set of $X \otimes Y$ is $|X| \times |Y|$. Realizers are defined as $\|x \otimes y\| := \{(z \mapsto z \overleftarrow{@} \mathsf{a} \overleftarrow{@} \mathsf{a}') \mid \mathsf{a} \in \|x\|_X, \mathsf{a}' \in \|y\|_Y\}$.
- For $f : X \to X'$ and $g : Y \to Y'$, the map $f \otimes g$ is a function sending $x \otimes y$ to $f(x) \otimes g(y)$. The realizer for $f \otimes g$ is $((((z \mapsto z \overleftarrow{@} (\mathsf{r}_f \overleftarrow{@} p) \overleftarrow{@} (\mathsf{r}_g \overleftarrow{@} q)) \leftarrow q) \leftarrow p) \vec{@} w \leftarrow w)$.
- The underlying set of the unit object I is a singleton $\{*\}$. The realizer is $\|*\|_I := \{\overleftarrow{\mathsf{I}}\}$.
- The left unitor $l : I \otimes X \to X$ sends $* \otimes x$ to x, whose realizer is $(\overleftarrow{\mathsf{I}} \vec{@} w \leftarrow w)$. The realizer of l^{-1} is $((z \mapsto z \overleftarrow{@} \overleftarrow{\mathsf{I}} \overleftarrow{@} x) \leftarrow x)$.
- For objects X and Y, the underlying set of $X \multimap Y$ is the set of maps from X to Y. $\|f\| := \{\mathsf{r} \in |\mathcal{A}| \mid \mathsf{a} \vec{@} \mathsf{r} \in \|f(x)\|_Y$ for any $x \in |X|$ and $\mathsf{a} \in \|x\|_X\}$. This set is not empty since $(\mathsf{r}_f)^{\triangleright}$ is in the set for a realizer r_f of f.
- For $f : X' \to X$ and $g : Y \to Y'$, $f \multimap g$ is a function sending a map $h : X \to Y$ to a map $g \circ h \circ f : X' \to Y'$. The realizer for $f \multimap g$ is $((x \mapsto \mathsf{r}_g \overleftarrow{@} ((\mathsf{r}_f \overleftarrow{@} x) \vec{@} w)) \leftarrow w)$.
- The evaluation map $ev : X \otimes (X \multimap Y) \to Y$ sends $x \otimes f$ to $f(x)$. The realizer is $(((x \vec{@} v \leftarrow v) \leftarrow x) \vec{@} w \leftarrow w)$.
- For any map $f : X \otimes Z \to Y$, there exists a unique map $g : Z \to X \multimap Y$ which satisfies $ev \circ (id_X \otimes g) = f$. This g is given as a function sending z to a function $x \mapsto f(x \otimes z)$. The realizer of g is $((x \mapsto \mathsf{r}_f \overleftarrow{@} (t \mapsto t \overleftarrow{@} x \overleftarrow{@} z)) \leftarrow z)$.
- For objects X and Y, the underlying set of $Y \multimap X$ is the set of maps from X to Y. $\|f\| := \{\mathsf{r} \mid \mathsf{r}$ is a realizer of $f\}$. ◀

Here what important is the way to take realizers of tensor products. We take the realizers as $\|x \otimes y\| := \{(z \mapsto z \overleftarrow{@} \mathsf{a} \overleftarrow{@} \mathsf{a}') \mid \mathsf{a} \in \|x\|_X, \mathsf{a}' \in \|y\|_Y\}$, while we would take $(z \overleftarrow{@} \mathsf{a} \overleftarrow{@} \mathsf{a}' \leftarrow z)$ if we define in the same way as Proposition 8 and Proposition 12.

▶ Remark 36. Take an object $A := (|\mathcal{A}|, \|\text{-}\|)$, where $\|\mathsf{a}\| := \{\mathsf{a}\}$. If we assume $\mathbf{Asm}(\mathcal{A})$ is an SMCC and the natural transformation for the symmetry sends $x \otimes y$ to $y \otimes x$, then there is a realizer r for the symmetry $A \otimes A \to A \otimes A$, which satisfies $\mathsf{r} \overleftarrow{@} (z \mapsto z \overleftarrow{@} \mathsf{a} \overleftarrow{@} \mathsf{a}') = (z \mapsto z \overleftarrow{@} \mathsf{a}' \overleftarrow{@} \mathsf{a})$ for arbitrary $\mathsf{a}, \mathsf{a}' \in |\mathcal{A}|$. Then we have $\overleftarrow{\mathsf{C}}' := ((\overleftarrow{\mathsf{I}} \vec{@} (\mathsf{r} \overleftarrow{@} (z \mapsto z \overleftarrow{@} x \overleftarrow{@} y)) \leftarrow y) \leftarrow x)$, which make $\mathcal{A}$ a BCI-algebra. Hence, when $\mathcal{A}$ is a bi-BDI-algebra and not a BCI-algebra, $\mathbf{Asm}(\mathcal{A})$ is not an SMCC (as long as we try to take the symmetry map in the natural way).

In the above proposition, we choose $\overleftarrow{@}$ to give $\mathbf{Asm}(\mathcal{A})$. However, it does not matter even if we choose $\vec{@}$.

▶ **Proposition 37.** *Let* $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@}, \vec{@})$ *be a bi-*BDI*-algebra. If we take an applicative structures* $\mathcal{A}' := (|\mathcal{A}|, \overleftarrow{@}')$ *for* $x \overleftarrow{@}' y := y \vec{@} x$*, then* $\mathbf{Asm}(\mathcal{A})$ *and* $\mathbf{Asm}(\mathcal{A}')$ *are isomorphic as categories. Moreover,* $\mathbf{Asm}(\mathcal{A})$ *is monoidally isomorphic to* $\mathbf{Asm}(\mathcal{A}')$ *with the reversed tensor products.*

We can also show $\mathbf{Mod}(\mathcal{A})$ is a monoidal bi-closed category for a bi-BDI-algebra $\mathcal{A}$. However, we cannot take tensor product as $\mathbf{Asm}(\mathcal{A})$ since $X \otimes Y$ in $\mathbf{Asm}(\mathcal{A})$ may not be a modest set even if X and Y are modest. We use a functor T that is the left adjoint of the inclusion functor $i : \mathbf{Mod}(\mathcal{A}) \hookrightarrow \mathbf{Asm}(\mathcal{A})$. T sends an assembly $(|X|, \|\text{-}\|_X)$ to a modest set $(|Z|, \|\text{-}\|_Z)$. Here $|Z| := |X|/\approx$, where the relation $\approx$ is the transitive closure of $\sim$ defined as $x \sim x' :\Leftrightarrow \|x\|_X \cap \|x'\|_X \neq \emptyset$. The realizers of $|Z|$ are $\|z\|_Z := \bigcup_{x \in z} \|x\|_X$. T sends a map f of $\mathbf{Asm}(\mathcal{A})$ to the canonical one of $\mathbf{Mod}(\mathcal{A})$, whose realizers are those of f. We define the tensor product $\boxtimes$ in $\mathbf{Mod}(\mathcal{A})$ as $X \boxtimes Y := T(iX \otimes iY)$. By using the same realizers in the proof of Proposition 35, we can prove that this $\boxtimes$ makes $\mathbf{Mod}(\mathcal{A})$ a monoidal bi-closed category. The same discussion for BCI-algebras is in [8] and the more general discussion about monoidal structures of reflective subcategories (for symmetric cases) is in [3].

▶ **Proposition 38.** $\mathbf{Mod}(\mathcal{A})$ *is a monoidal bi-closed category for a bi-*BDI*-algebra* $\mathcal{A}$*.*

To end of this subsection, we give a property about morphisms between bi-BDI-algebras.

▶ **Proposition 39.** *For bi-*BDI*-algebras* $(|\mathcal{A}_1|, \overleftarrow{@}_1, \vec{@}_1)$ *and* $(|\mathcal{A}_2|, \overleftarrow{@}_2, \vec{@}_2)$ *and an applicative morphism* $\gamma : (|\mathcal{A}_1|, \overleftarrow{@}_1) \to (|\mathcal{A}_2|, \overleftarrow{@}_2)$*,* $\gamma_* : \mathbf{Asm}(\mathcal{A}_1) \to \mathbf{Asm}(\mathcal{A}_2)$ *is a lax monoidal functor.*

▶ Remark 40. γ_* is not generally oplax monoidal since neither realizers for $\gamma_* I_1 \to I_2$ nor $\gamma_*(X \otimes_1 Y) \to \gamma_*(X) \otimes_2 \gamma_*(Y)$ exist.

4 Realizability models for modalities

In this section we relate our non-symmetric categorical realizability to the standard realizability based on BCI-algebras and PCAs. Our approach is similar to the case of linear combinatory algebras (LCAs) which relate BCI-algebras and PCAs. An LCA consists of a BCI-algebra $\mathcal{A}$, an endofunction $! : |\mathcal{A}| \to |\mathcal{A}|$ and several kinds of elements, such that the functor $!_*$ on $\mathbf{Asm}(\mathcal{A})$ (or $\mathbf{Mod}(\mathcal{A})$) becomes a linear exponential comonad on the SMCC [1]. While ! in an LCA is a function, in [8], ! is generalized to a total relation and the generalized LCAs are called relational linear combinatory algebras (rLCAs). We can obtain an rLCA from an adjoint pair between a BCI-algebra and a PCA.

The same construction can be applied to bi-BDI-algebras. That is, we can reformulate rLCAs for bi-BDI-algebras (Here we call *exp-rPLCAs*), and adjoint pairs between bi-BDI-algebras and PCAs induce exp-rPLCAs. Using exp-rPLCAs, we get models of !-modalities on non-symmetric multiplicative intuitionistic linear logic (MILL).

Also, we can obtain models for *exchange modalities* relating non-symmetric linear logics and symmetric MILL. A model of the logic with the exchange modality is given as a monoidal adjunction between an SMCC and a monoidal bi-closed category [9]. We can construct the model from a comonadic applicative morphism with certain conditions (Here we call an *exch-rPLCA*), and an adjoint pair between a bi-BDI-algebra and a BCI-algebra induces an exch-rPLCA.

4.1 Realizability models for !-modalities on non-symmetric linear logics

Linear exponential comonads on non-symmetric monoidal categories are investigated in [7], which model !-modalities on non-symmetric MILL.

▶ **Definition 41.** *A* linear exponential comonad ! *on a (non-symmetric) monoidal category* $\mathbf{C}$ *is a monoidal comonad on* $\mathbf{C}$ *such that the induced monoidal structure of the category of Eilenberg-Moore coalgebras is Cartesian.*

The above characterization is by Theorem 5 of [7]. However, originally linear exponential comonads are defined explicitly in [7] as tuples which consists of a monoidal comonad ! on $\mathbf{C}$ and monoidal natural transformations $e_X : !X \to I$ and $d_X : !X \to !X \otimes !X$ satisfying several conditions.

In this subsection, we generalize rLCAs to the non-symmetric case and show they give rise to linear exponential comonads.

▶ **Definition 42.** *An* exponential relational planar linear combinatory algebra *(exp-rPLCA) consists of a bi-*BDI*-algebra* $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{\mathbb{D}}, \vec{\mathbb{a}})$ *and a comonadic applicative morphism* $(!, \mathsf{e}, \mathsf{d})$ *on* $(|\mathcal{A}|, \overleftarrow{\mathbb{D}})$ *which satisfies the following.*

- *There is* $\mathsf{k} \in |\mathcal{A}|$ *such that* $\mathsf{k} \mathbin{\overleftarrow{\mathbb{D}}} \mathsf{x} \mathbin{\overleftarrow{\mathbb{D}}} (!\mathsf{y}) \subseteq \{\mathsf{x}\}$ *for any* $\mathsf{x}, \mathsf{y} \in |\mathcal{A}|$.
- *There is* $\mathsf{w} \in |\mathcal{A}|$ *such that* $\mathsf{w} \mathbin{\overleftarrow{\mathbb{D}}} \mathsf{x} \mathbin{\overleftarrow{\mathbb{D}}} (!\mathsf{y}) \subseteq \mathsf{x} \mathbin{\overleftarrow{\mathbb{D}}} (!\mathsf{y}) \mathbin{\overleftarrow{\mathbb{D}}} (!\mathsf{y})$ *for any* $\mathsf{x}, \mathsf{y} \in |\mathcal{A}|$.

▶ **Proposition 43.** *For an exp-rPLCA* $(\mathcal{A}, !)$, $!_*$ *is a linear exponential comonad on* $\mathbf{Asm}(\mathcal{A})$.

This proposition is proven by giving realizers for natural transformations associated with a linear exponential comonad. $e_X : !_*X \to I$ sending x to $*$ has a realizer $\mathsf{k} \mathbin{\overleftarrow{\mathbb{D}}} \overleftarrow{\mathsf{I}}$. $d_X : !_*X \to !_*X \otimes !_*X$ sending x to $x \otimes x$ has a realizer $\mathsf{w} \mathbin{\overleftarrow{\mathbb{D}}} (((z \mapsto z \mathbin{\overleftarrow{\mathbb{D}}} x \mathbin{\overleftarrow{\mathbb{D}}} y) \leftarrow\!\shortmid y) \leftarrow\!\shortmid x)$.

Although now we get a linear exponential comonad $!_*$ on $\mathbf{Asm}(\mathcal{A})$, at this point it has not been concluded that we get linear-non-linear models (i.e., monoidal adjunctions between monoidal closed categories and CCCs) by categorical realizability since we have not shown that the co-Kleisli adjunction between $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Asm}(\mathcal{A})_{!_*}$ is monoidal. In order to show that the co-Kleisli adjunction is indeed monoidal, it is enough to show that $\mathbf{Asm}(\mathcal{A})$ has Cartesian products. (It follows from Proposition 3 in [7].)

▶ **Proposition 44.** *For an exp-rPLCA* $(\mathcal{A}, !)$, $\mathbf{Asm}(\mathcal{A})$ *has Cartesian products, and thus the co-Kleisli adjunction between* $\mathbf{Asm}(\mathcal{A})$ *and a CCC* $\mathbf{Asm}(\mathcal{A})_{!_*}$ *is monoidal.*

The proof is almost the same as the proof of Proposition 12 of [8]. For instance, we take $\|(x, y)\| := \{(z \mapsto z \mathbin{\overleftarrow{\mathbb{D}}} (w \mapsto w \mathbin{\overleftarrow{\mathbb{D}}} u \mathbin{\overleftarrow{\mathbb{D}}} v) \mathbin{\overleftarrow{\mathbb{D}}} a) \mid \exists p, \exists q,\ u \in\ !p,\ v \in\ !q,\ p \mathbin{\overleftarrow{\mathbb{D}}} a \in \|x\|_X$ and $q \mathbin{\overleftarrow{\mathbb{D}}} a \in \|y\|_Y\}$ as realizers for Cartesian product. Here note that the Cartesian product $X \times Y$ is a modest set when X and Y are modest. Hence the above proposition for $\mathbf{Asm}(\mathcal{A})$ can be shown similarly for $\mathbf{Mod}(\mathcal{A})$. (See Remark 23 for restricting $!_*$ to $\mathbf{Mod}(\mathcal{A})$.)

▶ **Proposition 45.** *For an exp-rPLCA* $(\mathcal{A}, !)$, $!_*$ *is a linear exponential comonad on* $\mathbf{Mod}(\mathcal{A})$. *Furthermore,* $\mathbf{Mod}(\mathcal{A})$ *has Cartesian products and thus the co-Kleisli adjunction between* $\mathbf{Mod}(\mathcal{A})$ *and* $\mathbf{Mod}(\mathcal{A})_{!_*}$ *is monoidal.*

Next, we show that an adjoint pair between a bi-BDI-algebra and a PCA gives rise to an exp-rPLCA. First, note that it does not matter which application we choose when we take an adjoint pair, as shown in the next proposition.

▶ **Proposition 46.** *Let* $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{\mathbb{D}}, \vec{\mathbb{a}})$ *be a bi-*BDI*-algebra and* $\mathcal{B} = (|\mathcal{B}|, \cdot)$ *be a* BCI*-algebra. Given an adjoint pair* $(\delta \dashv \gamma) : (|\mathcal{A}|, \overleftarrow{\mathbb{D}}) \to \mathcal{B}$, *there is an adjoint pair* $(\delta' \dashv \gamma') : (|\mathcal{A}|, \overleftarrow{\mathbb{D}}') \to \mathcal{B}$ *for* $x \mathbin{\overleftarrow{\mathbb{D}}}' y := y \mathbin{\vec{\mathbb{a}}} x$, *such that* $\gamma = \gamma'$ *and* $\delta = \delta'$ *as total relations.*

▶ **Proposition 47.** *Let* $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{\mathbb{D}}, \vec{\mathbb{a}})$ *be a bi-*BDI*-algebra and* $\mathcal{B} = (|\mathcal{B}|, \cdot)$ *be a PCA. For an adjoint pair* $(\delta \dashv \gamma) : \mathcal{A} \to \mathcal{B}$, $(\delta \circ \gamma)$ *forms an exp-rPLCA.*

Proof. Let e and d be realizers associated with $\delta \circ \gamma$ being a comonadic applicative morphism. $\mathsf{k} \in ((x \vec{@} (\mathsf{e} \overleftarrow{@} (\mathsf{r}_\delta \overleftarrow{@} \delta M \overleftarrow{@} y)) \leftarrow\!\shortmid y) \leftarrow\!\shortmid x)$, where $M \in \lambda^* z.(\gamma \vec{\mathsf{I}})$. $\mathsf{w} \in ((x \vec{@} (\mathsf{e} \overleftarrow{@} (\mathsf{r}_\delta \overleftarrow{@} \delta M \overleftarrow{@} (\mathsf{d} \overleftarrow{@} y))) \leftarrow\!\shortmid y) \leftarrow\!\shortmid x)$, where $M := \lambda^* z.\mathsf{r}_\gamma \cdot (\mathsf{r}_\gamma \cdot \gamma N \cdot z) \cdot z$ and $N := (((t \mapsto\!\shortmid t \overleftarrow{@} u \overleftarrow{@} v) \leftarrow\!\shortmid v) \leftarrow\!\shortmid u)$. ◀

Next, we give an example of an adjoint pair between a bi-BDI-algebra and a PCA. This example is a planar variant of the untyped linear lambda calculus with ! [13].

▶ **Example 48.** Suppose infinite supply of variables $x, y, z, \ldots$. Terms are defined grammatically as follows:

$$M ::= x \mid M \overleftarrow{@} M' \mid M \vec{@} M' \mid (M \leftarrow\!\shortmid x) \mid (x \mapsto\!\shortmid M) \mid !M \mid \lambda !x.M$$

Here x of $(M \leftarrow\!\shortmid x)$ (or $(x \mapsto\!\shortmid M)$) is the rightmost (or leftmost) free variable of M, appears exactly once in M and not in any scope of !. Take an equational relation on the terms as the congruence of the three equational axioms $(M \leftarrow\!\shortmid x) \overleftarrow{@} N = M[N/x]$, $N \vec{@} (x \mapsto\!\shortmid M) = M[N/x]$ and $(\lambda !x.M) \overleftarrow{@} (!N) = M[N/x]$. Let Λ be a set of equivalence classes of closed terms.

Then we obtain a bi-BDI-algebra $\mathcal{A} := (\Lambda, \overleftarrow{@}, \vec{@})$ and a PCA $\mathcal{B} := (\Lambda, \cdot)$, where $M \cdot N := M \overleftarrow{@} !N$. Here K and S exist in $\mathcal{B}$ as $\lambda !x.\lambda !y.x$ and $\lambda !x.\lambda !y.\lambda !z.x \overleftarrow{@} !z \overleftarrow{@} !(y \overleftarrow{@} !z)$.

Take an applicative morphism $\gamma : \mathcal{A} \to \mathcal{B}$ as the identity whose realizer is $\lambda !x.\lambda !y.x \overleftarrow{@} y$. Take $\delta : \mathcal{B} \to \mathcal{A}$ sending M to $!M$ whose realizer is $\lambda !x.\lambda !y.!(x \overleftarrow{@} !y)$. Then $\delta \dashv \gamma$.

4.2 Realizability models for exchange modalities

The Lambek calculus with the exchange modality and its categorical models are introduced in [9]. Here by "Lambek calculus" we mean the non-symmetric MILL with left and right implications. Its extension with the exchange modality is the *commutative/non-commutative (CNC) logic*, which is a sequent calculus composed of two (commutative and non-commutative) logics. Categorical models of CNC logics are given as monoidal adjunctions between monoidal bi-closed categories and SMCCs, and are called *Lambek adjoint models*. As well as exp-rPLCAs, we can define comonadic applicative morphisms giving rise to Lambek adjoint models.

▶ **Definition 49.** *An* exchange relational planar linear combinatory algebra *(exch-rPLCA) consists of a bi-*BDI*-algebra* $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@}, \vec{@})$ *and a comonadic applicative morphism* $(\xi, \mathsf{e}, \mathsf{d})$ *on* $\mathcal{A}$ *with* $\overleftarrow{\mathsf{c}} \in |\mathcal{A}|$ *satisfying* $\overleftarrow{\mathsf{c}} \overleftarrow{@} \mathsf{x} \overleftarrow{@} (\xi \mathsf{y}) \overleftarrow{@} (\xi \mathsf{z}) \subseteq \mathsf{x} \overleftarrow{@} (\xi \mathsf{z}) \overleftarrow{@} (\xi \mathsf{y})$ *for any* $\mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$.

▶ **Proposition 50.** *For an exch-rPLCA* $(\mathcal{A}, \xi)$*, the co-Kleisli category* $\mathbf{Asm}(\mathcal{A})_{\xi_*}$ *is an SMCC and the co-Kleisli adjunction is monoidal.*

Proof (Sketch).

- We define tensor products in $\mathbf{Asm}(\mathcal{A})_{\xi_*}$ as $X \otimes' Y := (|X| \times |Y|, \|\text{-}\|)$, where $\|x \otimes' y\| := \{(z \mapsto\!\shortmid z \overleftarrow{@} \mathsf{a} \overleftarrow{@} \mathsf{a}') \mid \mathsf{a} \in \xi \|x\|_X \text{ and } \mathsf{a}' \in \xi \|y\|_Y\}$.
- For $f : X \to X'$ and $g : Y \to Y'$ in $\mathbf{Asm}(\mathcal{A})_{\xi_*}$, $f \otimes' g$ sends (x, y) to $(f(x), g(y))$. The realizer for $f \otimes' g$ is $(M \vec{@} (\mathsf{e} \overleftarrow{@} w) \leftarrow\!\shortmid w)$, where $M \in (((z \mapsto\!\shortmid z \overleftarrow{@} (\mathsf{r}_\xi \overleftarrow{@} (\xi \mathsf{r}_f) \overleftarrow{@} (\mathsf{d} \overleftarrow{@} x)) \overleftarrow{@} (\mathsf{r}_\xi \overleftarrow{@} (\xi \mathsf{r}_g) \overleftarrow{@} (\mathsf{d} \overleftarrow{@} y))) \leftarrow\!\shortmid y) \leftarrow\!\shortmid x)$.
- We define the unit object J in $\mathbf{Asm}(\mathcal{A})_{\xi_*}$ as $(\{*\}, \|\text{-}\|)$, where $\|*\| := \{\overleftarrow{\mathsf{I}}\}$.
- The symmetry $\sigma : X \otimes' Y \to Y \otimes' X$ sends $x \otimes' y$ to $y \otimes' x$. The realizers for σ and σ^{-1} are $(M \vec{@} (\mathsf{e} \overleftarrow{@} w) \leftarrow\!\shortmid w)$, where $M := \overleftarrow{\mathsf{c}} \overleftarrow{@} (((z \mapsto\!\shortmid z \overleftarrow{@} y \overleftarrow{@} x) \leftarrow\!\shortmid x) \leftarrow\!\shortmid y)$.
- For objects X and Y, the underlying set of the linear exponent $Y \multimap X$ is $Hom_{\mathbf{Asm}(\mathcal{A})}(\xi_* X, Y)$. $\|f\| := \{\mathsf{r} \in |\mathcal{A}| \mid \mathsf{r} \text{ is the realizer of } f\}$.

- For $f : X' \to X$ and $g : Y \to Y'$ in $\mathbf{Asm}(\mathcal{A})_{\xi_*}$, $g \circ\!\!- f$ sends a map $h : \xi_* X \to Y$ in $\mathbf{Asm}(\mathcal{A})$ to a map $g \circ (\xi_* h) \circ d_X \circ (\xi_* f) \circ d_{X'}$ from $\xi_* X'$ to Y' in $\mathbf{Asm}(\mathcal{A})$ (that is, a map from X' to Y' in $\mathbf{Asm}(\mathcal{A})_{\xi_*}$), where $d_X : \xi_* X \to \xi_* \xi_* X$ is the comultiplication of ξ_*. The realizer for $g \circ\!\!- f$ is $((\mathsf{r}_g \mathbin{\overleftarrow{@}} (\mathsf{r}_\xi \mathbin{\overleftarrow{@}} w \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} (\mathsf{r}_\xi \mathbin{\overleftarrow{@}} (\xi \mathsf{r}_f) \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} v)))) \leftarrow\!\!\mapsto v) \leftarrow\!\!\mapsto w)$.
- For any map $f : Z \otimes' X \to Y$ in $\mathbf{Asm}(\mathcal{A})_{\xi_*}$, there exists a unique map $g : Z \to Y \circ\!\!- X$ in $\mathbf{Asm}(\mathcal{A})_{\xi_*}$, which sends z to $x \mapsto f(z \otimes x)$. The realizer of g is $((\mathsf{r}_f \mathbin{\overleftarrow{@}} (\mathsf{r}_\xi \mathbin{\overleftarrow{@}} (\mathsf{r}_\xi \mathbin{\overleftarrow{@}} (\xi M) \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} z)) \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} x)) \leftarrow\!\!\mapsto x) \leftarrow\!\!\mapsto z)$, where $M := (((w \mapsto w \mathbin{\overleftarrow{@}} z \mathbin{\overleftarrow{@}} x) \leftarrow\!\!\mapsto x) \leftarrow\!\!\mapsto z)$.
- The co-Kleisli functor $\xi_* : \mathbf{Asm}(\mathcal{A})_{\xi_*} \to \mathbf{Asm}(\mathcal{A})$ is strong monoidal. Realizers for $\xi_* J \to I$ and $\xi_*(X \otimes' Y) \to \xi_* X \otimes \xi_* Y$ in $\mathbf{Asm}(\mathcal{A})$ are e. A realizer for $I \to \xi_* J$ is in $(w \mathbin{\overleftarrow{@}} (\xi \overleftarrow{\mathsf{I}}) \leftarrow\!\!\mapsto w)$. A realizer for $\xi_* X \otimes \xi_* Y \to \xi_*(X \otimes' Y)$ is in $(((\mathsf{r}_\xi \mathbin{\overleftarrow{@}} (\mathsf{r}_\xi \mathbin{\overleftarrow{@}} \xi M \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} x)) \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} y) \leftarrow\!\!\mapsto y) \leftarrow\!\!\mapsto x) \mathbin{\overrightarrow{@}} w \leftarrow\!\!\mapsto w)$, where $M := (((z \mapsto z \mathbin{\overleftarrow{@}} x \mathbin{\overleftarrow{@}} y) \leftarrow\!\!\mapsto y) \leftarrow\!\!\mapsto x)$. ◀

When we consider ξ_* on $\mathbf{Mod}(\mathcal{A})$, we can not use the same definition of the tensor product $\otimes'$ as $\mathbf{Asm}(\mathcal{A})_{\xi_*}$ since $X \otimes' Y$ may not be a modest set. We again use the functor T that is the left adjoint of the inclusion functor $i : \mathbf{Mod}(\mathcal{A}) \hookrightarrow \mathbf{Asm}(\mathcal{A})$ and define the tensor product $X \boxtimes Y$ in $\mathbf{Mod}(\mathcal{A})_{\xi_*}$ as $T(iX \otimes' iY)$. Then we can prove that $\mathbf{Mod}(\mathcal{A})_{\xi_*}$ becomes an SMCC with $\boxtimes$ in the same way as Proposition 50.

▶ **Proposition 51.** *For an exch-rPLCA $(\mathcal{A}, \xi)$, the co-Kleisli category $\mathbf{Mod}(\mathcal{A})_{\xi_*}$ is an SMCC and the co-Kleisli adjunction is monoidal.*

Similar to exp-rPLCAs, we can obtain an exch-rPLCA from an adjoint pair.

▶ **Proposition 52.** *Let $\mathcal{A} = (|\mathcal{A}|, \overleftarrow{@}, \overrightarrow{@})$ be a bi-*BDI*-algebra and $\mathcal{B} = (|\mathcal{B}|, \cdot)$ be a* BCI*-algebra. For an adjoint pair $(\delta \dashv \gamma) : \mathcal{A} \to \mathcal{B}$, $(\delta \circ \gamma)$ forms an exch-rPLCA.*

Proof. Let e and d be realizers associated with $\delta \circ \gamma$ being a comonadic applicative morphism. $\overleftarrow{\mathsf{c}} \in (((x \mathbin{\overrightarrow{@}} (\mathsf{e} \mathbin{\overleftarrow{@}} (\mathsf{r}_\delta \mathbin{\overleftarrow{@}} (\mathsf{r}_\delta \mathbin{\overleftarrow{@}} \delta M \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} y)) \mathbin{\overleftarrow{@}} (\mathsf{d} \mathbin{\overleftarrow{@}} z))) \leftarrow\!\!\mapsto z) \leftarrow\!\!\mapsto y) \leftarrow\!\!\mapsto x)$, where $M \in \lambda^* y.\lambda^* z.\mathsf{r}_\gamma \cdot (\mathsf{r}_\gamma \cdot \gamma N \cdot z) \cdot y$ and $N := (((x \mapsto x \mathbin{\overleftarrow{@}} z \mathbin{\overleftarrow{@}} y) \leftarrow\!\!\mapsto y) \leftarrow\!\!\mapsto z)$. ◀

▶ **Example 53.** Take an ordered group $(G, \cdot, e, \leq)$. Let T be the same set as in Example 32. We get a BCI-algebra $\mathcal{T}'$ as the powerset of $\{t \in T \mid e = |t|\}$, where the application is the same as the right application of $\mathcal{T}$ and $\mathsf{C} := \{((t_1 \circ\!\!- t_2) \circ\!\!- t_3) \circ\!\!- ((t_1 \circ\!\!- t_3) \circ\!\!- t_2) \mid |t_i| = e\}$.

In Example 32, we get a bi-BDI-algebra $\mathcal{T}$. We obtain two applicative morphisms $\gamma : \mathcal{T} \to \mathcal{T}'$ as a function $M \mapsto \{(t \circ\!\!- t) \mid t \in M\}$ and $\delta : \mathcal{T}' \to \mathcal{T}$ as the identity function. Here the realizer for γ is $\{((t_1 \circ\!\!- t_1) \circ\!\!- (t_2 \circ\!\!- t_2)) \circ\!\!- ((t_1 \circ\!\!- t_2) \circ\!\!- (t_1 \circ\!\!- t_2)) \mid t_1, t_2 \in T\}$ and the realizer for δ is $\{t \circ\!\!- t \mid t \in T\}$. γ and δ form an adjoint pair, where the realizer for $id \preceq \gamma \circ \delta$ is $\{t \circ\!\!- (t \circ\!\!- t) \mid e \leq |t|\}$ and the realizer for $\delta \circ \gamma \preceq id$ is $\{((t \circ\!\!- t) \circ\!\!- (t \circ\!\!- t)) \circ\!\!- (t \circ\!\!- t) \mid t \in T\}$.

▶ Remark 54. The above construction can not be applied to exp-rPLCAs. No matter how we take a powerset $\mathcal{T}_0$, since $M \mathbin{\overleftarrow{@}} N \mathbin{\overleftarrow{@}} \emptyset = \emptyset$ for any $M, N \in \mathcal{T}_0$, $\mathcal{T}_0$ does not contain the element acting as the K-combinator and thus cannot be a PCA.

5 Related work

In [15], the relationships between the planar lambda calculus and planar maps are investigated. It is shown that we can generate rooted planar maps with orientations by combining a few kinds of "imploid moves", that corresponds to the combinatory completeness of $\mathsf{BI}(-)^\bullet$-algebras and the planar lambda calculus. We may apply the correspondence between planar

lambda terms and rooted planar maps to our bi-planar lambda terms. Here the corresponding rooted maps have two kinds of vertexes with two inputs and one output (or, two outputs and one input) while rooted maps for planar lambda terms have one kind.

Although we use the word "Lambek calculus" as a variant of non-symmetric MILL with left and right implications in this paper, the word "Lambek calculus" has various meanings as logics. The basics about the Lambek calculus is in [10]. Our treatment in this paper is from [9].

Conditions of bi-BDI-algebras may look like "dual combinators" introduced in [4]. In both of them, elements acting as functions can act to an argument from both left and right sides. However, a dual combinatory logic has only one sort of application and the reductions does not satisfy the confluence, whereas bi-BDI-algebras have two sorts of applications and the confluence for the bi-planar lambda calculus holds.

Realizability for (symmetric) linear exponentials are introduced in [1] as LCAs and generalized to rLCAs in [8]. Section 4.1 of this paper is the reformulations of some contents of [8] to the planar case. The original definition of rLCA is described using not k and w but $\preceq$. We can also define exp-rPLCAs without using k nor w, however, we are not sure whether we can define exch-rPLCAs without using $\overline{\mathsf{c}}$.

6 Conclusion

In this paper, we presented a new class of applicative structures called bi-BDI-algebras. Bi-BDI-algebras lie between $\mathsf{BI}(-)^\bullet$-algebras and BCI-algebras and correspond to the bi-planar lambda calculus. Given a bi-BDI-algebra $\mathcal{A}$, we obtain monoidal bi-closed categories $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$. We also introduced exp-rPLCAs and exch-rPLCAs which induce categorical models for !-modalities and exchange modalities on non-symmetric logics. We can get exp-rPLCAs from adjoint pairs between bi-BDI-algebras and PCAs, and exch-rPLCAs from adjoint pairs between bi-BDI-algebras and BCI-algebras.

We conclude this paper by describing three issues for future work. First, while we have shown that a bi-BDI-algebra $\mathcal{A}$ induces a monoidal bi-closed category $\mathbf{Asm}(\mathcal{A})$, it is not clear whether being $\mathbf{Asm}(\mathcal{A})$ a monoidal bi-closed category leads that $\mathcal{A}$ is a bi-BDI-algebra. For some other classes, we can show such a proposition. For instance, being $\mathbf{Asm}(\mathcal{A})$ a CCC/SMCC/closed multicategory leads that $\mathcal{A}$ is a PCA/BCI-algebra/$\mathsf{BI}(-)^\bullet$-algebra under some natural conditions. (See Proposition 19 of [14] for the case of $\mathsf{BI}(-)^\bullet$-algebras.)

Second, there are a few points that exp-rPLCA and exch-rPLCA do not behave in the same way, and we would like to clarify them. As we said in the previous section, while exp-rPLCA can be defined without using k nor w, we are not sure that exch-rPLCA can be defined in such a style. Also, while an adjoint pair between a bi-BDI-algebra $\mathcal{A}$ and a PCA $\mathcal{B}$ induce a monoidal adjunction between $\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Asm}(\mathcal{B})$, the adjunction between $\mathbf{Asm}(\mathcal{A}')$ and $\mathbf{Asm}(\mathcal{B}')$ induced from an adjoint pair between a bi-BDI-algebra $\mathcal{A}'$ and a BCI-algebra $\mathcal{B}'$ is not generally monoidal.

Finally, we are yet to find more interesting concrete examples of bi-BDI-algebras and adjoint pairs, which should be useful for investing non-commutative logics and their models in a systematic way.

References

1 Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

2 Samson Abramsky and Marina Lenisa. Linear realizability and full completeness for typed lambda-calculi. *Annals of Pure and Applied Logic*, 134(2-3):122–168, 2005.

3 Brian Day. A reflection theorem for closed categories. *Journal of pure and applied algebra*, 2(1):1–11, 1972.

4 J Michael Dunn and Robert K Meyer. Combinators and structurally free logic. *Logic Journal of IGPL*, 5(4):505–537, 1997.

5 Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

6 Masahito Hasegawa. A quantum double construction in Rel. *Mathematical Structures in Computer Science*, 22(4):618–650, 2012.

7 Masahito Hasegawa. Linear exponential comonads without symmetry. *Electronic Proceedings in Theoretical Computer Science*, 238:54–63, 2016.

8 Naohiko Hoshino. Linear realizability. In *International Workshop on Computer Science Logic*, pages 420–434. Springer, 2007.

9 Jiaming Jiang, Harley Eades III, and Valeria de Paiva. On the lambek calculus with an exchange modality. In Thomas Ehrhard, Maribel Fernández, Valeria de Paiva, and Lorenzo Tortora de Falco, editors, *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity-TLLA@FLoC 2018, Oxford, UK, 7-8 July 2018*, volume 292 of *EPTCS*, pages 43–89, 2018.

10 Joachim Lambek. Deductive systems and categories II. standard constructions and closed categories. In *Category theory, homology theory and their applications I*, pages 76–122. Springer, 1969.

11 John R Longley. *Realizability toposes and language semantics*. PhD thesis, University of Edinburgh, 1995.

12 Oleksandr Manzyuk. Closed categories vs. closed multicategories. *Theory and Applications of Categories*, 26(5):132–175, 2012.

13 Alex Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In *International Conference on Rewriting Techniques and Applications*, pages 219–234. Springer, 2005.

14 Haruka Tomita. Realizability Without Symmetry. In *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021.

15 Noam Zeilberger. A theory of linear typings as flows on 3-valent graphs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 919–928. ACM, 2018.

16 Noam Zeilberger and Alain Giorgetti. A correspondence between rooted planar maps and normal planar lambda terms. *Log. Methods Comput. Sci.*, 11, 2015.

Number of Variables for Graph Differentiation and the Resolution of GI Formulas

Jacobo Torán
Universität Ulm, Germany

Florian Wörz
Universität Ulm, Germany

Abstract

We show that the number of variables and the quantifier depth needed to distinguish a pair of graphs by first-order logic sentences exactly match the complexity measures of clause width and positive depth needed to refute the corresponding graph isomorphism formula in propositional narrow resolution.

Using this connection, we obtain upper and lower bounds for refuting graph isomorphism formulas in (normal) resolution. In particular, we show that if k is the number of variables needed to distinguish two graphs with n vertices each, then there is an $n^{O(k)}$ resolution refutation size upper bound for the corresponding isomorphism formula, as well as lower bounds of 2^{k-1} and k for the tree-like resolution size and resolution clause space for this formula. We also show a (normal) resolution size lower bound of $\exp\big(\Omega(k^2/n)\big)$ for the case of colored graphs with constant color class sizes.

Applying these results, we prove the first exponential lower bound for graph isomorphism formulas in the proof system SRC-1, a system that extends resolution with a global symmetry rule, thereby answering an open question posed by Schweitzer and Seebach.

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Complexity theory and logic; Mathematics of computing → Graph theory

Keywords and phrases Proof Complexity, Resolution, Narrow Width, Graph Isomorphism, k-variable fragment first-order logic $\mathcal{L}_k$, Immerman's Pebble Game, Symmetry Rule, SRC-1

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.36

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2021/097/` [37]

Funding This research was supported by the Deutsche Forschungsgemeinschaft (DFG) under project number 430150230, "Complexity measures for solving propositional formulas".

Acknowledgements We thank Pascal Schweitzer for helpful discussions. We would also like to thank the anonymous reviewers for many insightful comments and suggestions.

1 Introduction

In an attempt to give a logical characterization of polynomial-time decidable graph properties, as well as a description of general classes of graph canonization algorithms, Immerman identified certain fragments of first-order logic suitable for expressing graph properties [21, 22]. In this setting, for such a language $\mathcal{L}$ of first-order logic sentences, two graphs G and H are *$\mathcal{L}$-equivalent*, denoted by $G \equiv_{\mathcal{L}} H$, if for all sentences $\psi \in \mathcal{L}$ it holds that $G \models \psi \iff H \models \psi$. Immerman noticed that the number of variables needed for expressing a property is a good complexity measure and defined the *k-variable fragment of first-order logic* $\mathcal{L}_k$ as the set of first-order logic formulas with the edge and equality relations that use at most k different variables (possibly re-quantifying them). He also defined the stronger class $\mathcal{C}_k$ by adding counting quantifiers to the class $\mathcal{L}_k$ and defined two pebble games for proving (non)equivalence of structures in these classes.

It was shown in [8] that two graphs are $\mathcal{C}_k$-equivalent if and only if they cannot be distinguished with the $(k-1)$-dimensional Weisfeiler–Leman algorithm, a well-known method for testing graph isomorphism. Roughly speaking, the 1-dimensional Weisfeiler–Leman (WL) algorithm [41, 40], or color refinement algorithm, identifies non-isomorphic colored graphs by updating in a series of steps the original vertex colors according to the multiset of colors of their neighbors. This basic step is applied repeatedly until the coloring stabilizes. This procedure can be generalized to the k-dimensional Weisfeiler–Leman algorithm (k-WL) by partitioning the set of k-tuples of vertices into automorphism-invariant equivalence classes (see e. g., [8, 23, 24] for excellent overviews of the powers and limits of this procedure).

The graph isomorphism problem (GraphIso), deciding whether two given graphs are isomorphic, has been intensively studied, as it is one of the few problems in NP that is not known to be complete for this class nor to be in P. Also unknown is whether the problem is in co-NP. It had been conjectured that GraphIso is solvable using the k-dimensional Weisfeiler–Leman algorithm, with k being sublinear in the number of vertices of the graphs. However, this was shown to be false in the seminal work of Cai, Fürer, and Immerman [8], using the $\mathcal{C}_k$ pebble game as a central tool. The Weisfeiler–Leman method still plays a central role in the algorithmic research on GraphIso; for example, Babai's celebrated algorithm for GraphIso [4] uses the k-WL method as a subroutine, with k being polylogarithmic in the number of vertices.

The field of proof complexity provides a different approach for studying the complexity of the GraphIso problem. Roughly speaking, in this setting, one tries to find out the smallest size of a proof in a concrete system of the fact that two graphs are non-isomorphic. It holds that GraphIso is in co-NP if and only if there is a concrete proof system with polynomial-size proofs of non-isomorphism. Similar to the Cook–Reckhow program [10] for the unsatisfiability problem UNSAT, this defines a clear line of research trying to provide superpolynomial size lower bounds for refuting graph (non)isomorphism formulas in stronger and stronger proof systems. The situation is even more interesting here than in the SAT case, since it would not be too surprising if GraphIso $\in$ co-NP, and this would imply the existence of polynomial-size proofs for the problem in some system. In fact, GraphIso is in co-AM [5], a randomized version of co-NP.

A first example of such a lower bound was given in [36], where it was shown that a family of unsatisfiable formulas encoding pairs of non-isomorphic graphs in a natural way requires exponential-size resolution refutations. These graphs are based on the CFI construction from [8]. The lower bound can be explained as an "encoding" of the Tseitin tautologies [38] into graph isomorphism instances. This result has been extended to stronger proof systems: In [7], the authors proved linear degree lower bounds for the algebraic systems Polynomial Calculus and Positivstellensatz by studying graphs arising from Tseitin tautologies. They furthermore characterized the power of the Weisfeiler–Leman algorithm in terms of an algebraic proof system lying between degree-k Nullstellensatz and degree-k Polynomial Calculus. Moreover, it has been shown in [3, 28, 18] that the expressive power of k-WL lies between the k-th and $(k+1)$-st level of the canonical Sherali–Adams LP hierarchy [34]. By the construction in [8], no sublinear level of Sherali–Adams suffices to decide GraphIso. Again, building on the work of [8], it was shown in [30] that there exist pairs of non-isomorphic n-vertex graphs such that any Sum-of-Squares proof of non-isomorphism must have degree $\Omega(n)$. In related work [9], it was shown that no sublinear level of the Lasserre hierarchy suffices to decide GraphIso.

Very recently, a different view was considered by Schweitzer and Seebach in [33] by introducing symmetry rules into the picture. The authors proved that resolution extended with the well-known symmetry rule SRC-2 from Krishnamurthy [26] has polynomial-size

refutations for all the instances of the graph isomorphism problem for which exponential size lower bounds for (normal) resolution are known. They pointed to the search for hard instances of graph isomorphism for resolution extended with the existing symmetry rules that define the proof systems SRC-1, SRC-2, and SRC-3, a hierarchy of systems with more and more powerful symmetry rules [1, 35]. They pose the question of whether graph non-isomorphism formulas have superpolynomial resolution complexity in any of these proof systems. These are very interesting questions since finding symmetries in a formula in order to be able to apply Krishnamurthy's rules is closely related to graph isomorphism. Finding lower bounds for non-isomorphism in a system with symmetry rules can be seen as finding lower bounds for proving non-isomorphism with the help of an "isomorphism subroutine".

1.1 Our Results

We show a strong connection between the $\mathcal{L}_k$ fragment of first-order logic and the propositional resolution proof system. This is done by proving that the number of variables and the quantifier depth simultaneously needed to distinguish two graphs G and H in first-order logic exactly corresponds to the width and positive depth of a narrow resolution refutation of the unsatisfiable formula $\mathrm{ISO}(G, H)$ stating that the graphs are isomorphic (Theorem 17). Narrow resolution [17] is a slight variation of (normal) resolution that allows a distinction by cases rule, allowing to deal with the inconveniences of having long clauses in the formula. As in the case of the clause width measure [6], narrow width allows, in our case, to derive upper and lower bounds for the size of the resolution refutations of non-isomorphism. Furthermore, we show that narrow width also provides a lower bound for the clause space needed in resolution, as it is the case for the standard width measure. In particular, we prove that for any pair of non-isomorphic graphs (G, H) with n vertices each and $k \in \mathbb{N}$:

- If $G \not\equiv_{\mathcal{L}_k} H$, then there is a (normal) resolution refutation of $\mathrm{ISO}(G, H)$ of size $n^{\mathrm{O}(k)}$;
- if $G \equiv_{\mathcal{L}_k} H$, then every tree-like resolution refutation of $\mathrm{ISO}(G, H)$ has size $\geq 2^k$;
- if $G \equiv_{\mathcal{L}_k} H$, then every (normal) resolution refutation of $\mathrm{ISO}(G, H)$ has clause space $\geq k + 1$; and
- for a pair of graph colorings (λ, μ) with $(G, \lambda) \equiv_{\mathcal{L}_k} (H, \mu)$, every (normal) resolution refutation of $\mathrm{ISO}(G, H)$ has size $\exp\big(\Omega(k^2/m^2)\big)$, where $m := \sum_{v \in G} |\text{color-class}(v)|$.

The last result allows to directly derive resolution size lower bounds from Immerman's pebble game for $\mathcal{L}_k$. We use this result to prove that a version of the multipede graphs defined in [11] has exponential resolution size lower bounds. We also observe that Krishnamurthy's SRC-1 symmetry rule cannot be applied to the isomorphism formulas for asymmetric graphs and conclude that the resolution size lower bound for the multipede graphs also holds for the SRC-1 system. This provides the first example of a class of graphs whose isomorphism formulas have exponential size lower bounds for the size of resolution refutations with one of the symmetry rules, thus solving a question from [33].

1.2 Organization of This Paper

The rest of this paper is organized as follows. In Section 2, we introduce resolution complexity measures, narrow resolution, and Krishnamurthy's symmetry rules, as well as the graph isomorphism formulas and Immerman's pebble game. Then, in Section 3, we prove the connection between narrow resolution width and $\mathcal{L}_k$. This yields the upper bounds on resolution size and the lower bounds on tree-like resolution size for refuting $\mathrm{ISO}(G, H)$. The exponential lower bound for the size of SRC-1 graph isomorphism formula refutations is shown in Section 4. Finally, in Section 5, clause space lower bounds for proving graph non-isomorphism in resolution are shown.

Due to space reasons some proofs have been omitted from this version. They can be found in the full-length version of the paper [37].

2 Preliminaries

We let $\mathbb{N}$ denote the set of positive integers. For $n \in \mathbb{N}$, we let $[n] := \{k \in \mathbb{N} \mid 1 \leq k \leq n\}$.

A *literal* ℓ over a Boolean variable x is either x itself or its negation $\overline{x} := \neg x$. For a literal ℓ, we put $\overline{\ell} := \neg x$ if $\ell = x$, and $\overline{\ell} := x$ if $\ell = \neg x$; and call ℓ and $\overline{\ell}$ *complementary* literals. A *clause* $C = (\ell_1 \vee \cdots \vee \ell_k)$ is a (possibly empty) disjunction of literals ℓ_i. We let the symbol $\square$ denote the contradictory *empty clause* (the clause without any literals). A *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses. It is often advantageous to think of clauses as sets of literals and CNF formulas as sets of clauses (i. e., sets of sets). The *set of variables occurring in a clause* C will be denoted by $\mathrm{Vars}(C)$. The notion of the set of variables in a clause is extended to CNF formulas by taking unions. An *assignment/restriction* α for a CNF formula F is a function that maps some subset of $\mathrm{Vars}(F)$, denoted by $\mathrm{Dom}(\alpha)$, to $\{0,1\}$. We will consider the graph of this function and call this set also an assignment. We let $|\alpha| := |\mathrm{Dom}(\alpha)|$ be the *size* of α. We denote the empty assignment with ε. By naturally extending α by the definition $\alpha(\overline{x}) := \overline{\alpha(x)}$, we can define the result of applying α to C, which we denote by $C|_\alpha$: one deletes all occurrences of literals ℓ from C, where $\alpha(\ell) = 0$; if there is a literal $\ell \in C$ with $\alpha(\ell) = 1$, then $C|_\alpha = 1$. The notation $F|_\alpha$ denotes the formula, where all clauses containing a literal ℓ with $\alpha(\ell) = 1$ are deleted and each remaining clause C is replaced by $C|_\alpha$. If ℓ is a literal that is not assigned by α, and $a \in \{0,1\}$, then $\alpha\{\ell = a\}$ denotes the *extension of* α with $\big(\alpha\{\ell = a\}\big)(x) := \alpha(x)$ for all $x \notin \{\ell, \overline{\ell}\}$ and $\big(\alpha\{\ell = a\}\big)(\ell) = a$ as well as $\big(\alpha\{\ell = a\}\big)(\overline{\ell}) = 1 - a$.

2.1 Resolution and Complexity Measures

If $B \vee x$ and $C \vee \overline{x}$ are clauses, then the *resolution rule* allows the derivation of the clause $R := (B \vee C)$. In the resolution rule, we call $B \vee x$ and $C \vee \overline{x}$ the *parents* and R the *resolvent*.

▶ **Definition 1.** *A* resolution derivation *of a clause D from a CNF formula F (denoted by $\pi : F \vdash D$) is an ordered sequence of clauses $\pi = (C_1, \ldots, C_t)$ such that $C_t = D$, and each clause C_i, for $i \in [t]$, is*
(1) *either an* axiom clause $C_i \in F$,
(2) *or a* weakening *of a clause C_j with $j < i$, i. e., $C_i \supseteq C_j$,*
(3) *or is derived from clauses C_j and C_k with $j < k < i$ by the* resolution rule.
A derivation of the empty clause from an unsatisfiable CNF formula F is called refutation.

To every refutation π, we can associate a *refutation DAG* G_π: The clauses of the refutations label the vertices of the DAG; for every application of the resolution rule we include edges from the parents to the resolvent; and for each application of the weakening rule we include edges from the original to the weakened clauses. We say that a resolution refutation π is *tree-like* if G_π is a tree.

▶ **Definition 2.** *The* size *of a resolution refutation π, denoted* $\mathrm{Size}(\pi)$*, is defined to be the number of vertices in the underlying refutation DAG G_π.*

The width of a clause C *is defined by* $\mathrm{Width}(C) := |C|$*, whereas the* width of a formula F *is given by* $\mathrm{Width}(F) := \max_{C \in F} \mathrm{Width}(C)$*. Similarly, we put* $\mathrm{Width}(\pi) := \max_{i \in [t]} \mathrm{Width}(C_i)$ *for a refutation $\pi = (C_1, \ldots, C_t)$.*

The depth $\mathrm{Depth}(\pi)$ *of a refutation π is the length of a longest path in the underlying refutation DAG G_π.*

In the following, we will consider the one-sided version of depth, called *positive depth*, that was recently introduced in [31].

▶ **Definition 3.** *If C and R are clauses with $C \setminus R = \{\ell\}$, we say that the literal ℓ is* introduced *from R to C. The* positive depth of a clause C *in a resolution refutation π, denoted* $\mathrm{PosDepth}(C)$*, is the minimal number of negative literals introduced (while also counting re-introductions) along any (inverse) path in G_π from the empty clause to C. The* positive depth of a refutation π *is defined by* $\mathrm{PosDepth}(\pi) := \max_{C\in\pi} \mathrm{PosDepth}(C)$.

We will also refer to the clause space measure for resolution. Intuitively, the clause space of a refutation π, $\mathrm{CS}(\pi)$, is the maximum number of clauses that need to be kept in memory simultaneously when verifying the proof π. A more formal definition can be found in [14].

2.1.1 Narrow Resolution and Narrow Width

The standard definition of width is not well suited for proving size lower bounds of formulas having large width themselves (cf. [6]), like the isomorphism formulas (cf. Section 2.2). A more natural way to deal with the width concept in such formulas was introduced by Galesi and Thapen in [17] together with the concept of narrow resolution that does not take into account the width of the axioms.

▶ **Definition 4.** *A* narrow resolution derivation *of a clause D from a CNF formula F is an ordered sequence of clauses $\pi = (C_1, \ldots, C_t)$ such that $C_t = D$, and for each $i \in [t]$, the clause C_i is obtained by rule (1), (2), or (3) of a (normal) resolution derivation (Definition 1) or by the following* distinction by cases *step:*

(4) *If $(B \vee x_1 \vee \cdots \vee x_m) \in F$, and if there are clauses $C_{j_1} = (A_1 \vee \overline{x_1}), \ldots, C_{j_m} = (A_m \vee \overline{x_m})$ with $j_1 < \cdots < j_m < i$, then we can derive $C_i := (B \vee A_1 \vee \cdots \vee A_m)$ in one step.*

We write $\text{N-Width}(\pi) \leq k$ *if π is a narrow resolution derivation and* $\mathrm{Width}(C_i) \leq k$ *for all $i \in [t]$ with $C_i \notin F$.*

The definition here is a slight generalization of the original one in [17] since, in rule (4), we do not require all the A_j clauses to coincide, and we allow for a subclause B to be present in the axiom clause (note, however, that the width of each A_j and B will be counted). This modification also allows an exact characterization of the number of pebbles needed in Immerman's game in terms of the width measure in narrow resolution, as shown in Theorem 17.

▶ **Definition 5.** *For a measure $\mathcal{C} \in \{\mathrm{Size}, \mathrm{Width}, \mathrm{Depth}, \mathrm{PosDepth}, \mathrm{CS}, \text{N-Width}\}$, by taking the minimum over all refutations π of an unsatisfiable formula F, we define $\mathcal{C}(F \vdash \square) := \min_{\pi: F \vdash \square} \mathcal{C}(\pi)$ as the* size, width, depth, positive depth, clause space, and narrow width of refuting F in resolution*, respectively.*

2.1.2 Krishnamurthy's Symmetry Rules

Krishnamurthy [26] observed that symmetries arise naturally in proofs of combinatorial principles and suggested some rules to simplify such proofs.

▶ **Definition 6.** *Let L be a finite set of complementary literals. Then, a bijective mapping $f\colon L \to L$ is called a* renaming *if for every $\ell \in L$ we have $\overline{f(\ell)} = f(\overline{\ell})$. For a clause $C \subseteq L$ and a renaming f, we set $f(C) := \{f(\ell) \mid \ell \in C\}$. For a formula F with $\bigcup_{C\in F} C \subseteq L$ we put $f(F) := \{f(C) \mid C \in F\}$.*

▶ **Definition 7** (The symmetry rules, [26, 39]). *Let F be a CNF formula and C a clause that can be derived by a proof $\pi : F' \vdash C$ from a subformula $F' \subseteq F$. If there exists a renaming $f\colon \mathrm{Lits}(F) \to \mathrm{Lits}(F)$ with $f(F') \subseteq F$, then the* local symmetry rule with complementation *allows the derivation of $f(C)$ from C in one step in the extended proof system. If we have the additional restriction $F' = F$, we speak of the* global symmetry rule with complementation. *Adding the global or local rule, respectively, to the proof system resolution (i. e., we consider proofs in which each clause is inferred by resolution from two clauses listed earlier in the proof, or by the respective symmetry rule from one clause earlier in the proof) yields the proof systems* SRC-1 *and* SRC-2.

Allowing also to use so-called *dynamic symmetries*, i. e., symmetries in the clauses already resolved, and not restricting ourselves to symmetries in the original axioms, one can define the proof system *SRC-3*. We refer to [35].

2.2 Graph Isomorphism and GI Formulas

An *(undirected) graph* is a tuple $G = \big(V_G, E_G\big)$, where V_G is a finite set of *vertices* and $E_G \subseteq \binom{V_G}{2}$ is the set of *edges*. A *colored graph* (G, λ) is a graph G together with a function $\lambda\colon V \to \mathcal{C}$, called *coloring*, where $\mathcal{C}$ is some set of *colors*. We treat every uncolored graph as a monochromatic graph.

▶ **Definition 8.** *Two colored graphs (G, λ) and (H, μ) are* isomorphic*, denoted by $(G, \lambda) \cong (H, \mu)$, if there is a color- and edge-respecting bijection $\varphi\colon V(G) \to V(H)$, called* (color-preserving) isomorphism *from G to H, i. e., $\{u, v\} \in E_G \iff \big\{\varphi(u), \varphi(v)\big\} \in E_H$ and $\lambda(v) = \mu\big(\varphi(v)\big)$ holds for all $u, v \in V_G$. An* automorphism *of a colored graph (G, λ) is an isomorphism from (G, λ) to (G, λ). We denote by $\mathrm{Iso}(G, H)$ the set of isomorphisms between G and H and by $\mathrm{Aut}(G)$ the set of automorphisms of G.*

Every coloring $\lambda\colon V_G \to \mathcal{C}$ of a graph G induces a partition of V_G: for a color $c \in \mathrm{Im}(\lambda)$, we call $\lambda^{-1}(c) \subseteq V(G)$ a *color class* of G. The *color class size* of G is the cardinality of its largest color class. It is known that the GraphIso problem can be solved in polynomial time when the color classes have constant size [16].

We encode instances of the GraphIso problem as Boolean formulas. As explained below, the formulas used here are a slight modification of those in [36]. Throughout the paper, we will consider only isomorphism formulas corresponding to pairs of graphs having the same number of vertices.

▶ **Definition 9.** *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs with $V_G = \{v_1, \ldots, v_n\}$ and $V_H = \{w_1, \ldots, w_n\}$. The formula $\mathrm{ISO}(G, H)$ is defined by the following clauses:*

Type 1 clauses: *for every $i \in [n]$ the clause $(x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,n})$ indicating that vertex $v_i \in V_G$ is mapped to some vertex in V_H; and for every $j \in [n]$ the clause $(x_{1,j} \vee x_{2,j} \vee \cdots \vee x_{n,j})$ indicating that vertex $w_j \in V_H$ is the image of some vertex in V_G.*

Type 2 clauses: *for every $i, j, k \in [n]$ with $i \neq j$ the clause $(\overline{x_{i,k}} \vee \overline{x_{j,k}})$ indicating that not two different vertices are mapped to the same one; and for every $i, j, k \in [n]$ with $j \neq k$ the clause $(\overline{x_{i,j}} \vee \overline{x_{i,k}})$ indicating that the variables encode a function.*

Type 3 clauses: *for every $i, j, k, \ell \in [n]$ with $i < j$ and $k \neq \ell$ with $\{v_i, v_j\} \in E_G \Leftrightarrow \{v_k, v_\ell\} \notin E_H$, the clause $(\overline{x_{i,k}} \vee \overline{x_{j,\ell}})$ expressing the adjacency relation (an edge cannot be mapped to a non-edge and vice-versa).*

The formula $\mathrm{ISO}(G, H)$ has n^2 variables and $\mathrm{O}(n^4)$ clauses. The clauses of Type 2 and Type 3 have width 2, while the clauses of Type 1 have width n.

Clearly, these formulas are satisfiable if the corresponding graphs are isomorphic. In the original definition of the ISO(G, H) formulas [36], the second possibility of Type 1 and Type 2 clauses was not considered. The formulas with and without these clauses are equivalent under satisfiability. We include these clauses here in order to obtain an exact characterization of Immerman's pebble game. Including these clauses can only make the lower bounds for the resolution of these formulas for non-isomorphic graphs stronger. The situation is similar to that for other principles, like the Pigeon-Hole-Principle, where the formulas with the additional Type 1 and Type 2 clauses are called onto-functional-PHP formulas (see, e. g., [32]). We remark that PHP_n^{n+1} has exponential-size resolution proofs [20], but as noticed in [26, 39], polynomial-size proofs in SRC-1.

An advantage of the isomorphism formulas is that one can express colorings of the involved graphs G and H as partial assignments of the variables:

▶ **Definition 10.** *Let G, H be as in Definition 9 and let $\lambda\colon V_G \to \mathcal{C}$ and $\mu\colon V_H \to \mathcal{C}$ be two graph colorings. Set $\rho := \{x_{i,j} = 0 \mid i, j \in [n] \text{ with } \lambda(i) \neq \mu(j)\}$. Define the ISO-formula for the colored graphs as $\mathrm{ISO}_{\lambda,\mu}(G, H) := \mathrm{ISO}(G, H)|_\rho$.*

Observe that while every coloring can be represented by a restriction, a restriction is just a partial assignment and it does not always encode a coloring. A coloring can drastically reduce the number of variables in the isomorphism formula. We will later make use of this fact. It is not hard to see that we have $\mathrm{ISO}_{\lambda,\mu}(G, H) \in \mathrm{UNSAT} \iff (G, \lambda) \ncong (H, \mu)$.

▶ Remark 11. Since every pair of colorings (λ, μ) of a pair of graphs (G, H) can be encoded as a restriction ρ of the formula ISO(G, H) as explained, a lower bound on the size of a resolution refutation of the $\mathrm{ISO}_{\lambda,\mu}$-formula for colored graphs also holds for the ISO-formula of the corresponding monochromatic graphs.

It is illustrative to contrast the $\mathrm{ISO}_{\lambda,\mu}$-formulas with the ListIso problem which asks, given two graphs G and H, where each vertex $v \in V_G$ is equipped with a list $\mathfrak{L}(v) \subseteq V_H$, if there exists an isomorphism $\varphi\colon V_G \to V_H$ such that $\varphi(v) \in \mathfrak{L}(v)$ for all $v \in V_G$. This problem can also be easily expressed as a satisfiability problem by restricting the first kind of Type 1 clauses to contain only the possibilities for each vertex (and doing analogously with the second kind of Type 1 clauses). However, this restriction would not encode a graph coloring in general. Moreover, ListIso might be harder than GraphIso as it was shown in [27] (see also [25]) that this problem is NP-complete.

2.3 Immerman's Pebble Game

▶ **Definition 12** ([21, 22])**.** *For a given language $\mathcal{L}$ (of first-order logic sentences), we say that two graphs G and H are* $\mathcal{L}$-equivalent*, denoted by $G \equiv_{\mathcal{L}} H$ if for all sentences $\psi \in \mathcal{L}$ it holds that $G \vDash \psi \iff H \vDash \psi$.*

▶ **Definition 13** (k-variable fragment of first-order logic)**.** *The* k-variable fragment of first-order logic *$\mathcal{L}_k$ is the set of first-order logic formulas that use at most k different variables (possibly re-quantifying them). Furthermore, $\mathcal{L}_{k,m}$ is the subclass of $\mathcal{L}_k$ where the quantifier depth in the formulas is restricted to m.*

By allowing counting quantifiers, we can extend $\mathcal{L}_k$ to the more expressive fragment $\mathcal{C}_k$. For a graph G, we say that it has *Weisfeiler–Leman dimension* at most k if and only if $G \ncong_{\mathcal{C}_{k+1}} H$ for all graphs H non-isomorphic to G.

We next describe a pebble game that is equivalent to testing $\mathcal{L}_{k,m}$-equivalence (or $\mathcal{L}_k$-equivalence for the unrestricted game) and is a variant of an Ehrenfeucht-Fraïssé game [15, 12]. We borrow the notation from [23].

▶ **Definition 14** (Immerman's pebble game, [21]). *Let $m, k \in \mathbb{N}$. For graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ with an equal number of vertices, we define the* m-move k-pebble game of Immerman *as follows: The game is played by two players called* Player I *and* Player II *on the graphs G and H with k pairs of pebbles. The game proceeds in rounds, each of which is associated with a position consisting of pebble placements. The* position after move $r \in [m]$ *of the game is denotes by $(\vec{v}_r, \vec{w}_r) \in V_G^\ell \times V_H^\ell$ with $0 \leq \ell \leq k$. The initial position is the pair $\big((), ()\big)$ of empty tuples. We now describe a round of the game. Suppose the current position of the game is $(\vec{v}_r, \vec{w}_r) = \big((v_1, \ldots, v_\ell), (w_1, \ldots, w_\ell)\big)$.*

- *First, Player I chooses whether he wants to remove a pebble pair (only possible if $\ell > 0$) or to place a new pair of pebbles (only possible if $\ell < k$).*
  - *If he wants to remove a pair of pebbles, he chooses some $i \in [\ell]$ and the position of the game changes to $\big((v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_\ell), (w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_\ell)\big)$ and the next round begins.*
  - *Otherwise, he picks a graph $K \in \{G, H\}$ and a vertex $v \in V_K$.*
- *Player II then picks a vertex $w \in V_{\hat{K}}$, where $\hat{K} := \{G, H\} \setminus \{K\}$ is the graph not chosen by Player I. The position of the game changes to*

$$(\vec{v}_{r+1}, \vec{w}_{r+1}) := \begin{cases} \big((v_1, \ldots, v_\ell, v), (w_1, \ldots, w_\ell, w)\big) & \text{if } K = G, \\ \big((v_1, \ldots, v_\ell, w), (w_1, \ldots, w_\ell, v)\big) & \text{otherwise,} \end{cases}$$

and the next round begins.

We say that Player II survives round r *of the game if and only if $G[\vec{v}_r] \cong H[\vec{w}_r]$, i. e., the map $v_i \mapsto w_i$ (for $i \in [\ell]$) is an isomorphism of the subgraphs induced by the pebbled vertices. If any difference between the induced ordered subgraphs is exposed within at most m rounds, then we say that* Player I wins the m-move game. *This is precisely the case when there are $i, j \in [\ell]$ such that $v_i = v_j \not\Leftrightarrow w_i = w_j$ or $\{v_i, v_j\} \in E_G \not\Leftrightarrow \{w_i, w_j\} \in E_H$ or there is an $i \in [\ell]$ such that the colors of v_i and w_i are different.*

If there is no restriction on the number of rounds m being played, Player I wins the game *if he wins some round, while* Player II survives the game *if she can survive forever.*

The interpretation of a configuration $\big((v_1, \ldots, v_\ell), (w_1, \ldots, w_\ell)\big)$ is that the i-th pebble pair is placed on the vertices v_i and w_i (for $i \in [\ell]$).

3 Connection Between Narrow Resolution Width and $\mathcal{L}_k$

Immerman's pebble game can be directly translated as a Spoiler–Duplicator type game played on the $\mathrm{ISO}(G, H)$ formulas. This kind of game has often been used in proof complexity arguments. The game defined here is a version of the game for the characterization of resolution width from [2] except that now Spoiler cannot choose variables but clauses, and Duplicator has to satisfy some literal in the chosen clause. Very similar games have already been defined in [13] and [17]. The only difference is that in our game, Spoiler can only choose Type 1 clauses (instead of any clause as in [13] or even variables as in [17]). For some of our proofs, we need to define the witnessing games also on restricted isomorphism formulas $\mathrm{ISO}(G, H)|_\gamma$ for some restriction γ. In this case, we say that the *Type of an axiom $C|_\gamma$* in $\mathrm{ISO}(G, H)|_\gamma$ (1, 2, or 3) is the same as that of the original axiom C.

▶ **Definition 15** (k-witnessing game). *For $k \in \mathbb{N}$ and a restriction γ, Spoiler and Duplicator construct in rounds a partial assignment for the formula $\mathrm{ISO}(G, H)|_\gamma$. Initially, $\alpha_0 = \varepsilon$. At the beginning of round i, Spoiler chooses a subset of α_{i-1} of size at most $k - 1$ and a*

Type 1 clause $C|_\gamma$ in $\mathrm{ISO}(G,H)|_\gamma$*. Then, Duplicator extends the chosen subset to one literal in $C|_\gamma$ (we call the obtained assignment α_i), satisfying this clause and not falsifying any clause in* $\mathrm{ISO}(G,H)|_\gamma$*. If this is not possible, Duplicator loses the game.*

▶ **Observation 16.** *$G \not\equiv_{\mathcal{L}_k} H$ if and only if Spoiler wins the k-witnessing game on* $\mathrm{ISO}(G,H)$*.*

Proof. The moves of Player I in Immerman's game, placing a pebble on a vertex $v_i \in V_G$ (or a vertex $w_j \in V_H$), correspond to Spoiler choosing a Type 1 clause of the kind $(x_{i,1} \vee \cdots \vee x_{i,n})$ (respectively one of the kind $(x_{1,j} \vee \cdots \vee x_{n,j})$). Player II's answer corresponds to the literal in these clauses satisfied by Duplicator. Since Duplicator only assigns variables with 1, only Type 2 or Type 3 clauses can be falsified. Player I wins Immerman's game when two pebbles on different vertices in one graph are answered with two pebbles on the same vertex in the other graph, corresponding to a Type 2 clause being falsified, or when the pebbles contradict the local isomorphism condition, and this corresponds to a Type 3 clause being falsified in the witnessing game. ◀

Using this game, we can show an equivalence between the number of variables needed to distinguish two graphs and the width measure in narrow resolution. We also notice that the number of rounds in both games matches. Since our witnessing game is a restriction of the game in [17], the proof of the result in one direction follows similar arguments as in the result for general formulas from the mentioned paper, but the bound we obtain is slightly better.

▶ **Theorem 17.** *For $k \in \mathbb{N}$, $G \not\equiv_{\mathcal{L}_{k,m}} H$ if and only if there is a narrow width resolution refutation π of* $\mathrm{ISO}(G,H)$ *with* $\text{N-Width}(\pi) \leq k-1$ *and* $\mathrm{PosDepth}(\pi) \leq m$ *simultaneously.*

Proof. For the direction from left to right, suppose $G \not\equiv_{\mathcal{L}_{k,m}} H$. By Observation 16, there is a winning strategy for Spoiler in the k-witnessing game on $\mathrm{ISO}(G,H)$ in m moves. This strategy has to be able to decide for each reachable partial assignment α in the game what variables can be deleted from the assignment, and what Type 1 clause C to query next. Such a strategy can be represented as a graph whose vertices store the information (α, C) with $|\alpha| \leq k-1$. From such a vertex and for every literal $\ell \in C$, there is a directed edge pointing to the vertex (α'_ℓ, C_ℓ). Here, α'_ℓ is the assignment obtained from α by setting $\ell = 1$ and maybe deleting some values (according to the strategy of Spoiler after knowing the answer of Duplicator for C). Furthermore, C_ℓ is the Type 1 clause queried next or a clause falsified by α'_ℓ. In this last case, (α'_ℓ, C_ℓ) is a winning position for Spoiler and a sink in the strategy graph. The only source of the graph is the initial vertex (α_0, C_0), where $\alpha_0 = \varepsilon$ and C_0 is the first Type 1 clause queried by Spoiler. Observe that since we have supposed that Spoiler has a winning strategy, this graph is acyclic. It is not necessarily a tree.

We can construct a resolution refutation DAG of $\mathrm{ISO}(G,H)$ by following the strategy backwards, i. e., by inverting the strategy graph. For this, we associate with each vertex (α, C) the clause C_α, defined as the set of literals falsified by α. With an inductive argument, starting at the sinks, we show that C_α can be resolved by narrow resolution from the clauses associated with the successor vertices of (α, C). For the sink vertices (α, C), by the way the strategy graph and the witness game are defined, C is an axiom of width 2 falsified by α. Since C is an axiom, it does not count for the narrow width. Using weakening, we can identify C_α with this vertex. For an interior vertex (α, C) with $C = (\ell_1 \vee \cdots \vee \ell_n)$ and with successor vertices $(\beta_1, C_1), \ldots, (\beta_n, C_n)$, we can suppose by induction that there are clauses $C_{\beta_1}, \ldots C_{\beta_n}$ associated with the successor vertices. Each assignment β_i has the form $\beta_i = \alpha_i \cup \{\ell_i = 1\}$ with $\alpha_i \subseteq \alpha$ and $|\beta_i| \leq k-1$. Because of this, C and each C_{β_i} have exactly the pair of complementary literals $(\ell_i, \overline{\ell_i})$ and can be resolved. Using a narrow resolution step, we can resolve all these clauses with C in one step, obtaining a clause $C_{\alpha'}$ with $\alpha' \subseteq \alpha$, and with weakening, we obtain C_α.

Since the clause mapped to the source vertex has to be falsified by the empty assignment, this is the empty clause, and the process defines a correct narrow resolution of ISO(G, H). Notice that all the clauses in the refutation have width at most $k-1$.

The depth of the strategy graph for Spoiler in the k-witnessing game is the maximum number of rounds m needed for Spoiler to defeat Duplicator in Immerman's $\mathcal{L}_k$-game. Following a path from the empty clause towards a clause C_α being derived by a narrow resolution step from $(\ell_1 \vee \cdots \vee \ell_n)$ and $C_{\beta_1}, \ldots, C_{\beta_n}$, one can notice that this step increases the positive depth measure by one when continuing the path towards the clauses $C_{\beta_1}, \ldots, C_{\beta_n}$ (the measure stays the same when continuing towards the axiom $(\ell_1 \vee \cdots \vee \ell_n)$). The positive depth measure also increases by at most one in any ordinary resolution step. Any weakening step does not increase the positive depth. By the correspondence between the game positions (β_i, C_i) and the clauses C_{β_i} of the proof π constructed above, this shows that we have N-Width$(\pi) \leq k-1$ and PosDepth$(\pi) \leq m$ simultaneously.

For the other direction, consider a narrow resolution refutation π for ISO(G, H) of width $k-1$. We describe a strategy for Spoiler to win the k-witnessing game. Starting at the empty clause, Spoiler queries Type 1 clauses, and with the literals satisfied by Duplicator, he keeps a set S of at most k variables $x_{i,j}$ assigned with value 1 by Duplicator. For a clause $C \in \pi$ and such a set S, we say that S *contradicts* C if the following conditions happen:

1. For every negated variable $\overline{x_{i,j}}$ in C, $x_{i,j} \in S$, and
2. for every positive variable $x_{i,j}$ in C, $x_{i,j} \notin S$ and $\exists k \in [n]$ such that ($x_{i,k} \in S$ or $x_{k,j} \in S$).

Starting at the empty clause and with the set $S = \emptyset$, S determines the predecessor clause in the refutation π where Spoiler moves to. At each step, Spoiler makes a query, updates S, and always moves to the predecessor clause contradicted by the current S. Let C be Spoiler's clause at a certain stage and S the corresponding set of variables.

If C is the (normal) resolvent of two clauses on variable $x_{i,j}$, in case one of these clauses is a Type 1 axiom, Spoiler queries it. Otherwise, Spoiler queries any of the two Type 1 clauses in ISO(G, H) containing $x_{i,j}$. If Duplicator assigns value 1 to this variable, Spoiler moves to the parent clause in which this variable is negated and adds $x_{i,j}$ to S. If some other variable is given value 1 by Duplicator, Spoiler adds it to S and moves to the contradicted parent clause. In both cases, Spoiler deletes from S all the variables that are not needed for contradicting the new clause.

If C is the result of a narrow resolution step involving a Type 1 axiom D, Spoiler queries this clause. Duplicator's answer must satisfy some variable $x_{i,j} \in D$. The set S together with this variable contradicts a predecessor clause C', and this clause cannot be D unless some Type 2 axiom is falsified (see the claim below). Spoiler moves to C', and he then deletes from S all the variables that are not necessary in S for contradicting the new clause. This means keeping one variable for each negated literal in C' and at most one variable for each positive literal in C'. Because the clauses in π have narrow width at most $k-1$, Spoiler needs to keep at most k variables in S at any moment.

If C comes from a weakening step, Spoiler just needs to forget some of the variables in S.

After each new variable set by Duplicator, if some Type 2 or Type 3 axiom of ISO(G, H) is falsified, Spoiler wins the game. We claim that if, at some point, S contradicts some Type 1 axiom, then S falsifies some Type 2 axiom. Suppose that S contradicts the Type 1 clause $(x_{i,1} \vee \cdots \vee x_{i,n})$. By definition, this means that $x_{i,1}, \ldots, x_{i,n} \notin S$, and thus, again, by definition, there is a set of n indices $\{k_1, \ldots, k_n\} \subseteq [n]$ such that $x_{k_1,1}, \ldots, x_{k_n,n} \in S$. In case that $\{k_1, \ldots, k_n\} = [n]$, there exists a $j \in [n]$ with $k_j = i$. Thus, $x_{k_j,j} = x_{i,j} \in S$. But then S does not contradict the clause $(x_{i,1} \vee \cdots \vee x_{i,n})$, a contradiction. In case not all k_j's are different, there are $j, j' \in [n]$ such that $j \neq j'$ but still $k_j = k_{j'}$. Since $x_{k_j,j} \in S$

as well as $x_{k_{j'},j'} = x_{k_j,j'} \in S$, the functionality axiom $(\overline{x_{k_j,j}} \vee \overline{x_{k_j,j'}})$ is falsified by S. The case in which S contradicts a Type 1 clause of the form $(x_{1,i} \vee \cdots \vee x_{n,i})$ can be treated symmetrically.

Eventually, some axiom is reached. This axiom is contradicted by the current S. If it is a Type 2 or 3 axiom, S falsifies it (these axioms have only negated literals), and Spoiler wins. As observed, if this is a Type 1 axiom, then some Type 2 axiom is falsified, and Spoiler wins.

In the described construction of a winning strategy, Spoiler always moves to the contradicted predecessor of the clause he is currently standing on. Such a move increases the positive depth of his position. Thus he needs at most m moves to win the Immerman game, where m is the positive depth of the refutation. ◀

Not surprisingly, the result above holds also for colored graphs, that is, the number of pebbles and rounds in Immerman's game on colored graphs correspond exactly to narrow width and positive depth in resolution of the isomorphism formula under the restriction encoding the coloring. We need, in fact, a version of the result for general restrictions, not only for colorings, and therefore we have to make use of the witnessing game, which is also well defined for restrictions. The proof follows the same steps as that for the result above. We state the part of the result that we will need for our results.

▶ **Observation 18.** *For $k \in \mathbb{N}$, and for every restriction γ, Spoiler has a winning strategy for the k-witnessing game on* $\mathrm{ISO}(G,H)|_\gamma$ *if and only if* $\text{N-Width}\big(\mathrm{ISO}(G,H)|_\gamma \vdash \square\big) \leq k-1$.

The equivalence between the number of variables for graph differentiation and narrow width allows us to give upper and lower bounds for the size of resolution proofs for isomorphism formulas.

▶ **Theorem 19.** *Let $k \in \mathbb{N}$, and G and H be two graphs with n vertices each. If $G \not\equiv_{\mathcal{L}_k} H$, then there is a (normal) resolution refutation of* $\mathrm{ISO}(G,H)$ *of size* $n^{\mathrm{O}(k)}$.

Proof. By the above result, if $G \not\equiv_{\mathcal{L}_k} H$, then the narrow resolution width of $\mathrm{ISO}(G,H)$ is at most $k-1$. Since there are n^2 variables in this formula, there are at most $\sum_{i=0}^{k-1} \binom{n^2}{i} 2^i \leq 2^{k-1}\big(\frac{en^2}{k-1}\big)^{k-1}$ clauses that can appear in a $(k-1)$-narrow resolution refutation of the formula. But a narrow resolution refutation is just like a normal one in which the distinction by cases is made in just one step. This can be simulated by at most n steps (with at most $n-1$ intermediate clauses that might be wider than k) in normal resolution. Using an upper bound for the partial sum of binomial coefficients, the total number of different clauses in the refutation is thus bounded by $n^{\mathrm{O}(k)}$, and it is polynomial for constant k. ◀

Observe that this result suggests a way to automatically generate short proofs for (non)-isomorphism formulas, following the same ideas as those in the algorithm proposed in [6] and [17] for general formulas. The algorithm would generate in stages all clauses that can be derived by narrow resolution of width $1, 2, 3, \ldots$, until the empty clause is derived. By the above result, the running time of this algorithm is $n^{\mathrm{O}(k)}$.

Lower bounds for narrow width also imply lower bounds on the size of a resolution refutation for $\mathrm{ISO}(G,H)$, in the same way that width lower bounds imply size lower bounds in normal resolution, as shown by Ben-Sasson and Wigderson [6]. For this, we follow the same steps as in the mentioned paper, adapted to narrow width. The general fact that narrow width provides lower bounds for resolution size has also been proved in [17]. By concentrating on the isomorphism formulas, we obtain tighter results. The next lemma is the basis for our lower bounds. It is a version in our context of [6, Lemma 3.2] or [17, Lemma 6].

▶ **Lemma 20.** *Let γ be a restriction and let ℓ be any literal in* $\mathrm{ISO}(G,H)|_\gamma$*. If Spoiler has a winning strategy for the k-witnessing game on* $\mathrm{ISO}(G,H)|_{\gamma\{\ell=1\}}$ *as well as for the $(k-1)$-witnessing game on* $\mathrm{ISO}(G,H)|_{\gamma\{\ell=0\}}$*, then he wins the k-witnessing game on* $\mathrm{ISO}(G,H)|_\gamma$*.*

Proof. We distinguish two cases depending on whether literal ℓ is positive or negative:

Case 1: $\ell = x_{i,j}$. The formula $\mathrm{ISO}(G,H)|_{\gamma\{x_{i,j}=1\}}$ is like $\mathrm{ISO}(G,H)|_\gamma$ without the two Type 1 clauses containing literal $x_{i,j}$ and without all occurrences of the literal $\overline{x_{i,j}}$. If Spoiler selects in the game on $\mathrm{ISO}(G,H)|_\gamma$ the same sequence of Type 1 clauses as in the game on $\mathrm{ISO}(G,H)|_{\gamma\{x_{i,j}=1\}}$, Duplicator either loses the game or sets a literal $x_{a,b}$ to 1 for a clause $C = (\overline{x_{a,b}} \vee \overline{x_{i,j}}) \in \mathrm{ISO}(G,H)|_\gamma$. When this happens, Spoiler restricts the assignment to $\gamma\{x_{a,b} = 0\}$, and then simulates the strategy for $\mathrm{ISO}(G,H)|_{\gamma\{x_{i,j}=0\}}$ on $\mathrm{ISO}(G,H)|_\gamma$. If Duplicator does not assign $x_{i,j} = 1$, she loses the game eventually by the assumption. If she does, then the clause C is falsified, and she also loses. Spoiler needs to keep an assignment of size at most k at any moment.

Case 2: $\ell = \overline{x_{i,j}}$. In this case, Spoiler simulates the strategy for $\mathrm{ISO}(G,H)|_{\gamma\{x_{i,j}=0\}}$ on the formula $\mathrm{ISO}(G,H)|_\gamma$, either winning the game or forcing Duplicator to assign $x_{i,j} = 1$ (by a Type 1 clause that contains $x_{i,j}$ and which was falsified in the $\mathrm{ISO}(G,H)|_{\gamma\{x_{i,j}=0\}}$-game). Restricting then the assignment to this literal, Spoiler now plays the strategy for $\mathrm{ISO}(G,H)|_{\gamma\{x_{i,j}=1\}}$ and Duplicator loses. ◀

From this result, lower bounds as in [6] follow directly. The advantage here is that the width of the axioms of $\mathrm{ISO}(G,H)$ is not subtracted from the exponent of the lower bound results, as it is done in [6, Corollary 3.4].

▶ **Theorem 21.** *Let $k \in \mathbb{N}$, and G, H be two non-isomorphic graphs with n vertices each. If $G \equiv_{\mathcal{L}_k} H$, then the size of a tree-like resolution refutation of* $\mathrm{ISO}(G,H)$ *is at least 2^k.*

Lower bounds on narrow width also imply, as noted in [17], lower bounds on general resolution size. Using (a version for narrow width) from [6, Theorem 3.5], one can show that if G and H are two non-isomorphic graphs with n vertices each with $G \equiv_{\mathcal{L}_k} H$, then the size of a resolution refutation of $\mathrm{ISO}(G,H)$ is at least $\exp\big(\Omega(k^2/n^2)\big)$. However, since the maximum number k of variables needed for distinguishing G and H is at most the number of vertices n, this only provides trivial lower bounds. A way to avoid this problem is to consider graph colorings under which the number k is still large, but the number of variables in $\mathrm{ISO}(G,H)$ is smaller. Since such a coloring can be expressed as a restriction ρ applied to $\mathrm{Vars}\big(\mathrm{ISO}(G,H)\big)$, and using the fact that for every restriction ρ, the size of a resolution refutation of $\mathrm{ISO}(G,H)$ is at least the size of the refutation of the formula under the restriction, $\mathrm{ISO}(G,H)|_\rho$, we obtain Theorem 23 below.

▶ **Definition 22.** *Let (G,λ) and (H,μ) be two colored graphs. For a vertex $v \in V_G$, we set* $\text{color-class}(v) := \mu^{-1}\big(\lambda(v)\big)$*, i. e., the set of vertices in V_H that have the same color as v.*

If (G,λ) and (H,μ) are two colored graphs in n vertices each, $m := \sum_{v \in V_G} |\text{color-class}(v)|$ is between n and n^2.

▶ **Theorem 23.** *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two non-isomorphic graphs with n vertices each, for which there is a $k \in \mathbb{N}$ and two colorings λ, μ such that $(G,\lambda) \equiv_{\mathcal{L}_k} (H,\mu)$. Then, the size of every resolution refutation of* $\mathrm{ISO}(G,H)$ *is at least* $\exp\big(\Omega(k^2/m)\big)$*, where $m := \sum_{v \in V_G} |\text{color-class}(v)|$ is the sum of the sizes of the color classes.*

Proof Sketch. Let $\rho := \{x_{i,j} = 0 \mid i,j \in [n] \text{ with } \lambda(i) \neq \mu(j)\}$, and consider the unsatisfiable formula $\mathrm{ISO}(G,H)|_\rho$. The set of variables of this formula is $\{x_{i,j} \mid i,j \in [n] \text{ with } \lambda(i) = \mu(j)\}$ and contains exactly $m = \sum_{v \in V_G} |\text{color-class}(v)|$ variables. Since $(G,\lambda) \equiv_{\mathcal{L}_k} (H,\mu)$, by Observation 18, $\text{N-Width}\big(\mathrm{ISO}(G,H)|_\rho \vdash \square\big) \geq k$. We following the same steps of that of [6, Theorem 3.5], with the modifications needed to deal with restrictions as done in Theorem 21.

For simplicity, let us denote $\mathrm{ISO}(G,H)|_\rho$ by F and let π be a (normal) resolution refutation of minimal size s of F. We define d and a to be $d := \lceil \sqrt{2m \ln s} \rceil$ and $a := (1 - \frac{d}{2m})^{-1}$. A clause in π is called *fat* if it contains more than d literals. Let π^* be the set of fat clauses in π. We prove by induction on m that $\text{N-Width}(F \vdash \square) \leq d + \log_a(|\pi^*|)$. The result follows from this implication since $|\pi^*| \leq s$ and therefore by the way a and d are defined, $\log_a(|\pi^*|)$ is bounded by $c\sqrt{2m \ln s}$ for some constant c. The base case $m = 0$ holds trivially. For the induction case, observe that F contains at most $2m$ literals and therefore one literal ℓ appears in at least $\frac{d}{2m}|\pi^*|$ fat clauses. We consider the two refutations of the formulas $F|_{\ell=1}$ and $F|_{\ell=0}$ obtained from π by setting ℓ to 1 and to 0, respectively. Setting $\ell = 1$ removes all the clauses including literal ℓ and leaves a refutation of $F|_{\ell=1}$ with at most $(1 - \frac{d}{2m})|\pi^*| = a^{-1}|\pi^*|$ fat clauses. By induction hypothesis we have $\text{N-Width}(F|_{\ell=1} \vdash \square) \leq d + \log_a(a^{-1}|\pi^*|) = d + \log_a(|\pi^*|) - 1$. Setting $\ell = 0$ produces a refutation of the formula $F|_{\ell=0}$ with less than m variables, and again by induction on m it holds $\text{N-Width}(F|_{\ell=0} \vdash \square) \leq d + \log_a(|\pi^*|)$. By applying Lemma 20 we obtain:

$$\text{N-Width}\big(F \vdash \square\big) \leq d + \log_a(|\pi^*|) \in \mathrm{O}\left(\sqrt{m \cdot \ln\Big(\mathrm{Size}\big(F \vdash \square\big)\Big)}\right).$$

Observe that since we are dealing with narrow resolution, we do not need the width of the axioms in $\mathrm{ISO}(G,H)|_\rho$ as an additional term, as in the result from [6]. It follows that $\mathrm{Size}\big(\mathrm{ISO}(G,H)|_\rho \vdash \square\big) = \exp\big(\Omega(k^2/m)\big)$. The last fact needed is that for every restriction ρ, $\mathrm{Size}\big(\mathrm{ISO}(G,H) \vdash \square\big) \geq \mathrm{Size}\big(\mathrm{ISO}(G,H)|_\rho \vdash \square\big)$. ◀

This result can then be automatically applied to graphs in which the maximum size of a color class is small.

▶ **Corollary 24.** *Let G and H be two graphs with n vertices each, and let $k \in \mathbb{N}$ and λ, μ be colorings with constant size color classes such that $(G,\lambda) \equiv_{\mathcal{L}_k} (H,\mu)$. Then, any resolution refutation of $\mathrm{ISO}(G,H)$ has size at least $\exp\big(\Omega(k^2/n)\big)$.*

Such constant size color classes are the case for the CFI graphs [8, 36] and the variant of the multipede graphs from [11]. In both examples, the maximum size of a color class is 4, while the number of variables needed to distinguish the graphs is linear in n. Thus, for both examples, the above result gives a resolution size lower bound of $\exp\big(\Omega(n)\big)$. One can also imagine this result being useful for proving resolution size lower bounds in cases in which not all color classes of the graphs have constant size, but the sum of the class sizes is still smaller than the number of variables needed to distinguish the graphs.

4 An Exponential Lower Bound for the Size of SRC-1 proofs for Graph (Non)Isomorphism

In this section, we show that there is a family of non-isomorphic graph pairs (G_n, H_n) that has only exponentially-long proofs of $\mathrm{ISO}(G_n, H_n)$ in the SRC-1 system. Exponential size lower bounds in SRC-1 are known [39], but not for graph isomorphism formulas. Our result is proven by observing that the global symmetry rule cannot be applied to formulas corresponding to graphs having only trivial automorphisms and restricting ourselves to such graphs.

▶ **Definition 25.** *A colored graph (G, λ) is called* asymmetric *if* $\mathrm{Aut}(G) = \{\mathrm{id}\}$.

To characterize the possible symmetries in an isomorphism formula, we need the notions of graph anti-automorphism and anti-isomorphism.

▶ **Definition 26.** *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs. An* anti-isomorphism σ *from G to H is a bijection between the vertices of G and H exchanging edges and non-edges, i. e., for all $u, v \in V_G$: $\{u, v\} \in E_G \iff \{\sigma(u), \sigma(v)\} \notin E_H$. An* anti-automorphism *of a graph G is an anti-isomorphism from G to G. We denote by $\text{A-Iso}(G, H)$ the set of anti-isomorphisms between G and H and by $\text{A-Aut}(G)$ the set of anti-automorphisms of G.*

We will also need the following simple observation.

▶ **Observation 27.** *Asymmetric graphs do not have any anti-automorphisms.*

Szeider observed in [35, Lemma 10] that if a formula is asymmetric, then the size of a resolution refutation and of an SRC-1 refutation of the formula are equal. The next lemma shows that if two graphs are asymmetric, then the corresponding ISO formula is asymmetric.

▶ **Lemma 28.** *Let G and H be two graphs with $|V_G| = |V_H| =: n \geq 3$, and let $F := \mathrm{ISO}(G, H)$. Further, let $f \colon \mathrm{Lits}(F) \to \mathrm{Lits}(F)$ be a renaming of the literals in F. Then $f(F) \subseteq F$ if and only if one of the following two cases hold:*

1. *There are two permutations $\sigma, \gamma \in S_n$ such that for every $(i, j) \in [n] \times [n]$, $f(x_{i,j}) = x_{\sigma(i),\gamma(j)}$ and $(\sigma, \gamma) \in \mathrm{Aut}(G) \times \mathrm{Aut}(H)$ or $(\sigma, \gamma) \in \text{A-Aut}(G) \times \text{A-Aut}(H)$; or*
2. *there are two permutations $\sigma, \gamma \in S_n$ such that for every $(i, j) \in [n] \times [n]$, $f(x_{i,j}) = x_{\gamma(j),\sigma(i)}$ and $(\sigma, \gamma^{-1}) \in \mathrm{Iso}(G, H) \times \mathrm{Iso}(G, H)$ or $(\sigma, \gamma^{-1}) \in \text{A-Iso}(G, H) \times \text{A-Iso}(G, H)$.*

Notice that if the graphs G and H are non-isomorphic and $f(F) \subseteq F$, then we can only be dealing with Case 1 in the lemma. Moreover, by Observation 27, if the graphs G and H do not have any non-trivial automorphisms, they cannot have anti-automorphisms either. In this case, a renaming f with $f(F) \subseteq F$ cannot exist, and therefore the global symmetry rule cannot be applied. This implies that size lower bounds for the resolution of (non)isomorphism formulas for asymmetric graphs coincide with their size lower bounds for the system SRC-1.

The Cai–Fürer–Immerman construction [8] gave graphs with a large Weisfeiler–Leman dimension, more precisely with a linear lower bound on the WL-dimension. A related construction of graphs satisfying this property, known as *multipedes*, was given in [19]. However, the resulting graphs are very large in terms of the WL-dimension. Neuen and Schweitzer improved in [29] the multipede construction combining it with size reduction techniques. Using a different construction, Dawar and Khan [11] showed how to obtain graphs whose Weisfeiler–Leman dimension is linear in the number of their vertices (as with the CFI graphs) and without any non-trivial automorphisms.

▶ **Theorem 29** ([11])**.** *For $k \in \mathbb{N}$, there is (a random process that produces with high probability) a family of asymmetric pairs of non-isomorphic graphs (G_k, H_k) with $\mathrm{O}(k)$ vertices, color classes of size 4, and Weisfeiler–Leman dimension k.*

In [11], it was furthermore demonstrated by conducting experiments that the resulting graphs provide hard examples for graph isomorphism solvers, matching the hardest-known benchmarks for graph isomorphism. The following result can be seen as a theoretical insight into this phenomenon.

Corollary 24 implies that the isomorphism formulas for the pairs (G_k, H_k) of non-isomorphic graphs from the above-mentioned construction have resolution refutations of size $\exp\big(\Omega(n)\big)$, where n is the number of vertices in the graphs (linear in the WL-dimension k). Since these graphs are asymmetric, from Lemma 28, we conclude:

▶ **Theorem 30.** *There is a (non-constructive) family of non-isomorphic graph pairs* (G_n, H_n) *with* $\mathrm{O}(n)$ *vertices each, such that any refutation of* $\mathrm{ISO}(G_n, H_n)$ *requires size* $\exp\big(\Omega(n)\big)$ *in the SRC-1 proof system.*

5 Lower Bounds on Clause Space for Proving Non-Isomorphism

Atserias and Dalmau [2] gave a combinatorial characterization of resolution width and used it to show the relation $\mathrm{CS}(F \vdash \square) \geq \mathrm{Width}(F \vdash \square) - \mathrm{Width}(F) + 1$ for any $F \in \mathrm{UNSAT}$. We will show in this section that this also holds for narrow width, with the advantage that, again, in this case, we do not have to worry about the width of the axioms. From this result, we obtain clause space lower bounds for the (normal) resolution of isomorphism formulas.

▶ **Definition 31** (w-NW Family)**.** *Given an unsatisfiable CNF formula* F *and a natural number* $w \in \mathbb{N}$*, we say that a family of assignments* $\mathcal{F}$ *for* F *is a* w-NW family *if all of the following properties hold:*
(1) $\mathcal{F} \neq \varnothing$*,*
(2) $\forall \alpha \in \mathcal{F}$ *and* $\forall C \in F$*:* $C|_\alpha \neq \square$*,*
(3) $\forall \alpha \in \mathcal{F}$*:* $|\mathrm{Dom}(\alpha)| \leq w$*,*
(4) $\forall \alpha \in \mathcal{F}$ *and* $\forall \beta \subseteq \alpha$*:* $\beta \in \mathcal{F}$*,*
(5) $\forall \alpha \in \mathcal{F}$ *with* $\mathrm{Dom}(\alpha) \leq w - 1$ *and* $\forall C \in F|_\alpha$*:* $\exists \ell \in C$ *such that* $\alpha\{\ell = 1\} \in \mathcal{F}$*.*

▶ **Theorem 32.** *If* F *is an unsatisfiable CNF formula with* $\text{N-Width}(F \vdash \square) > w$*, then there exists a* $(w+1)$*-NW family for* F*.*

▶ **Theorem 33.** *If there is a* $(w+1)$*-NW family for an unsatisfiable CNF formula* F*, then* $\mathrm{CS}(F \vdash \square) \geq w + 2$*.*

Proof Sketch. This follows from an adaptation of [2, Lemma 5], by noticing that the original constant for $\mathrm{Width}(F)$ vanishes by modifying point (5) of the definition of an Atserias–Dalmau family as we did. Playing the so-called Spoiler–Duplicator game on F, as in the proof of [2, Lemma 5], Duplicator has an answer to satisfy the queried clause in one round, making it not necessary for Spoiler to query the variables in a clause until he gets a satisfying assignment. ◀

▶ **Corollary 34.** *For every* $F \in \mathrm{UNSAT}$ *we have* $\mathrm{CS}(F \vdash \square) \geq \text{N-Width}(F \vdash \square) + 1$*.*

Using Theorem 17, we obtain:

▶ **Theorem 35.** *Let* $k \in \mathbb{N}$ *and let* G *and* H *be two non-isomorphic graphs with* $G \equiv_{\mathcal{L}_k} H$*. Then* $\mathrm{CS}\big(\mathrm{ISO}(G, H) \vdash \square\big) \geq k + 1$*.*

By the CFI construction [8], for every $n \in \mathbb{N}$, there is a pair of non-isomorphic graphs (G_n, H_n) such that G_n and H_n have $\mathrm{O}(n)$ vertices but $G_n \equiv_{\mathcal{C}_n} H_n$ (and therefore also $G_n \equiv_{\mathcal{L}_n} H_n$). Hence, for these graphs, $\mathrm{CS}\big(\mathrm{ISO}(G_n, H_n) \vdash \square\big) \geq \big|\mathrm{Vars}\big(\mathrm{ISO}(G_n, H_n)\big)\big|^{1/2} + 1$.

6 Conclusions

We have given an exact characterization for the number of variables needed to distinguish two graphs in first-order logic in terms of the narrow resolution width needed for refuting the corresponding isomorphism formulas. This fact allowed us to obtain upper and lower

bounds for the size and space of (normal) resolution refutation of such formulas. The size upper bound justifies a clause length increasing algorithm for the resolution (and solving) of isomorphism formulas of the kind proposed in [6] for general formulas.

The lower bounds techniques provide a simplified method to obtain resolution size lower bounds directly from the structure of the graphs, using the $\mathcal{L}_k$-logic, and without having to deal with the isomorphism formulas directly. All the known resolution size lower bounds for isomorphism formulas can be easily derived from this result. Moreover, we have been able to use the method to obtain exponential lower bounds for isomorphism formulas in the stronger system of SRC-1, which includes a global symmetry rule, answering a question posed in [33].

The obvious open question is to prove superpolynomial size lower bounds for isomorphism formulas in the stronger systems SRC-2 and SRC-3. However, one would need different ideas for this, since, as shown recently in [33], the families of graphs based on the CFI construction, like the ones used in all known lower bounds, have polynomial-size SRC-2 refutations.

References

1 Noriko H. Arai and Alasdair Urquhart. Local symmetries in propositional logic. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2000, St Andrews, Scotland, UK, July 3-7, 2000, Proceedings*, pages 40–51, 2000. doi:10.1007/10722086_3.

2 Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, 2008. Earlier conference version in *CCC '03*. doi:10.1016/j.jcss.2007.06.025.

3 Albert Atserias and Elitza N. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. *SIAM Journal on Computing*, 42(1):112–137, 2013. Earlier conference version in *ITCS '12*. doi:10.1137/120867834.

4 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC '16)*, pages 684–697, 2016. Full-length version in *arXiv:1512.03547*. doi:10.1145/2897518.2897542.

5 László Babai and Shlomo Moran. Arthur–Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988. doi:10.1016/0022-0000(88)90028-1.

6 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001. Earlier conference versions in *STOC '99* and *CCC '99*. doi:10.1145/375827.375835.

7 Christoph Berkholz and Martin Grohe. Limitations of algebraic approaches to graph isomorphism testing. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP '15)*, pages 155–166, 2015. doi:10.1007/978-3-662-47672-7_13.

8 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. Earlier conference version in *FOCS '89*. doi:10.1007/BF01305232.

9 Paolo Codenotti, Grant Schoenebeck, and Aaron Snook. Graph isomorphism and the Lasserre hierarchy. *arXiv*, 1401.0758, 2014. arXiv:1401.0758.

10 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.

11 Anuj Dawar and Kashif Khan. Constructing hard examples for graph isomorphism. *Journal of Graph Algorithms and Applications*, 23(2):293–316, 2019. doi:10.7155/jgaa.00492.

12 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961. doi:10.4064/fm-49-2-129-141.

13 Juan Luis Esteban, Nicola Galesi, and Jochen Messner. On the complexity of resolution with bounded conjunctions. *Theoretical Computer Science*, 321(2-3):347–370, 2004. Earlier conference version in *ICALP '02*. doi:10.1016/j.tcs.2004.04.004.

14 Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001. Preliminary versions in *STACS '99* and *CSL '99*. doi:10.1006/inco.2001.2921.

15 Roland Fraïssé. Sur une nouvelle classification des systèmes de relations. *Comptes Rendus*, 230:1022–1024, 1950.

16 Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 36–41, 1980. doi:10.1109/SFCS.1980.34.

17 Nicola Galesi and Neil Thapen. Resolution and pebbling games. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT '05)*, pages 76–90, 2005. doi:10.1007/11499107_6.

18 Martin Grohe and Martin Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015. Earlier conference version in *CSL '12*. doi:10.1017/jsl.2015.28.

19 Yuri Gurevich and Saharon Shelah. On finite rigid structures. *The Journal of Symbolic Logic*, 61(2):549–562, 1996. doi:10.2307/2275675.

20 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

21 Neil Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25(1):76–98, 1982. Earlier conference version in *FOCS '80*. doi:10.1016/0022-0000(82)90011-3.

22 Neil Immerman. *Descriptive Complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

23 Sandra Kiefer. *Power and limits of the Weisfeiler–Leman algorithm*. Dissertation, RWTH Aachen University, 2020. doi:10.18154/RWTH-2020-03508.

24 Sandra Kiefer. The Weisfeiler–Leman algorithm: an exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020. doi:10.1145/3436980.3436982.

25 Pavel Klavík, Dusan Knop, and Peter Zeman. Graph isomorphism restricted by lists. *Theoretical Computer Science*, 860:51–71, 2021. Earlier conference version in *WG '20*. doi:10.1016/j.tcs.2021.01.027.

26 Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985. doi:10.1007/BF00265682.

27 Anna Lubiw. Some NP-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1):11–21, 1981. doi:10.1137/0210002.

28 Peter N. Malkin. Sherali–Adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014. doi:10.1016/j.disopt.2014.01.004.

29 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC '18)*, pages 138–150. ACM, 2018. doi:10.1145/3188745.3188900.

30 Ryan O'Donnell, John Wright, Chenggang Wu, and Yuan Zhou. Hardness of robust graph isomorphism, lasserre gaps, and asymmetry of random graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*, pages 1659–1677, 2014. doi:10.1137/1.9781611973402.120.

31 Theodoros Papamakarios and Alexander Razborov. Space characterizations of complexity measures and size-space trade-offs in propositional proof systems. Technical Report TR21-074, ECCC, 2021. URL: https://eccc.weizmann.ac.il/report/2021/074/.

32 Alexander A. Razborov. Proof complexity of pigeonhole principles. In *Proceedings of the 5th International Conference on Developments in Language Theory (DLT '01), Revised Papers*, pages 100–116, 2001. doi:10.1007/3-540-46011-X_8.

33 Pascal Schweitzer and Constantin Seebach. Resolution with symmetry rule applied to linear equations. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS '21)*, pages 58:1–58:16, 2021. doi:10.4230/LIPIcs.STACS.2021.58.

34 Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discret. Math.*, 3(3):411–430, 1990. doi:10.1137/0403036.

35 Stefan Szeider. The complexity of resolution with generalized symmetry rules. *Theory of Computing Systems*, 38(2):171–188, 2005. Earlier conference version in *STACS '03*. doi:10.1007/s00224-004-1192-0.

36 Jacobo Torán. On the resolution complexity of graph non-isomorphism. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT '13)*, pages 52–66, 2013. doi:10.1007/978-3-642-39071-5_6.

37 Jacobo Torán and Florian Wörz. Number of variables for graph identification and the resolution of GI formulas. Technical Report TR21-097, ECCC, 2021. URL: https://eccc.weizmann.ac.il/report/2021/097.

38 Grigori S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, volume 8 of *Seminars in Mathematics*, pages 115–125. Consultants Bureau, 1968.

39 Alasdair Urquhart. The symmetry rule in propositional logic. *Discret. Appl. Math.*, 96-97:177–193, 1999. doi:10.1016/S0166-218X(99)00039-6.

40 Boris Weisfeiler. *On Construction and Identification of Graphs*, volume 558 of *Lecture Notes in Mathematics*. Springer, 1976. doi:10.1007/BFb0089374.

41 Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia, Seriya 2*, 9, 1968. Translation from Russian into English by Grigory Ryabov available under https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

Anti-Unification of Unordered Goals

Gonzague Yernaux
Faculty of Computer Science, University of Namur, Belgium
Namur Digital Institute, Belgium

Wim Vanhoof
Faculty of Computer Science, University of Namur, Belgium
Namur Digital Institute, Belgium

Abstract

Anti-unification in logic programming refers to the process of capturing common syntactic structure among given goals, computing a single new goal that is more general called a generalization of the given goals. Finding an arbitrary common generalization for two goals is trivial, but looking for those common generalizations that are either as large as possible (called largest common generalizations) or as specific as possible (called most specific generalizations) is a non-trivial optimization problem, in particular when goals are considered to be *unordered* sets of atoms. In this work we provide an in-depth study of the problem by defining two different generalization relations. We formulate a characterization of what constitutes a most specific generalization in both settings. While these generalizations can be computed in polynomial time, we show that when the number of variables in the generalization needs to be minimized, the problem becomes NP-hard. We subsequently revisit an abstraction of the largest common generalization when anti-unification is based on injective variable renamings, and prove that it can be computed in polynomially bounded time.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Anti-unification, Logic programming, NP-completeness, Time complexity, Algorithms, Inductive logic programming

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.37

Related Version *Full Version*: `https://arxiv.org/abs/2107.00341`

1 Motivation and Objectives

Anti-unification refers to the process of generalizing two (or more) program objects S into a single, more general, program object that captures some of the structure that is common to all the objects in S. In a classical logic programming context, the atom $p(X, Y)$ can thus be seen as a generalization of both the atoms $p(f(A), U)$ and $p(f(g(B)), h(C))$, thanks to the variables X and Y.

Anti-unification constitutes a useful tool in various contexts ranging from program analysis techniques (including partial evaluation, refactoring, automatic theorem proving, program transformation, formal verification and test-case generation [5, 24, 11, 22, 15]) to automated reasoning [20, 21] or analogy making [18], supercompilation [27] and even plagiarism detection [28]. Many of these static techniques are executed on programs written in the form of (constraint) Horn clauses, a formalism that has been praised for its ability to capture a program's essence in a quite universal and straightforward manner [14].

In the introductive example above, the presence of variables X and Y conceptually allows concrete instances (i.e. less general objects) to harbor any value at the positions corresponding to the variable positions. The generalization process is indeed usually achieved by "forgetting" parts of the objects to generalize (either by replacing sub-objects with variables or by dropping them altogether): the less syntactic information in an object, the more general it is. Most anti-unification methods are thus steered by a *variabilization* algorithm determining how to "forget" object parts when necessary while keeping (common) parts in the generalization.

Therefore, in general one is typically interested in computing what is often called a most specific generalization (or synonymously least general generalization), that is a generalization that captures a maximal amount of shared structure. With the atoms of the example above, the common generalization $p(f(X), Y)$ is in that regard a *better* anti-unification result than $p(X, Y)$, as it exhibits more common structure (namely the use of functor f). As this example hints, "better" results are often obtained at the cost of more complex anti-unification algorithms. In that regard, computing more specific generalizations often boils down to performing some kind of optimization in the variabilization process.

In a classical approach where goals are *ordered* sequences of atoms, a goal G is more general than some other goal G' if G' can be obtained by applying on G some substitution θ, being a mapping from variables to values. G then typically harbors more variables than G', making it a less instantiated, thus more general, version of G'. In that case, G and G' are related by the θ-subsumption relation from [25], often considered to be a foundation of Inductive Logic Programming where anti-unification is used as a way to learn a general hypothesis from specific examples [20]. As the name may suggest, looking for a generalization that is common to a group of program artefacts (be it terms, atoms, goals or even predicates as a whole) is referred to as anti-unification due to it being the dual operation of unification. Both can, in fact, be applied in similar contexts. Such applications of (anti-)unification include program transformation techniques for partial deduction [13, 11], fold/unfold routines [23], invariant generation [17] and reuse of proofs [3, 24].

The study of anti-unification so far has mainly been focused on such ordered goals. However, many applications require goals to be defined as (*unordered*) sets of atoms. It is the case, for instance, when considering the most declarative semantics of logic programs [12, 16, 14]. Having a clear overview of anti-unification operators computing most specific generalizations for unordered goals (sometimes called *linear* generalizations) in logic programs is necessary for generalization-driven semantic clone detection with programs composed of constraint Horn clauses [28, 19]. Indeed, generalization operators allow to quantify a certain amount of structural similarity between different predicate definitions by highlighting what parts these have in common. In [28], this quantitative similarity measurement is used as an indication of which semantic-preserving program transformation should be applied next in order to ultimately assess whether two programs (or predicates) are semantic clones. A quite similar approach has already been taken in the case of ordered goals in [5], an obvious application of this being plagiarism detection.

Directing our interest towards unordered goals also has the advantage of broadening the traditional anti-unification theories usually rooted in a setting where logic programming is based on operational semantics, by extending the theories to the more general area of Constraint Logic Programming (CLP), unordered goals being a crucial ingredient of the CLP(X) framework. The fixpoint semantics of CLP programs are indeed typically defined with no regard to the order of appearance of the atoms in a clause's body [16]. While CLP is interesting in its own right, it is also considered a serious candidate for representing abstract *algorithmic knowledge*, rather than mere computations, in a quite universal manner [14]. In that regard, focusing on unordered goals could pave the way for performing anti-unification at the algorithmic level rather than at the level of language-specific operations.

The topic of anti-unification in the case of unordered goals has ocasionally come up in studies focussed on related fields such as *equational* anti-unification, encompassing theories specified by commutativity or associative-commutativity axioms. The topic has been treated for first-order theories [1] as well as higher-order variants [9]. The latter work applies to the first-order case as well and provides polynomial algorithms for variants of anti-unification

for unordered input. A grammar-based approach to equational anti-unification including commutative theories, called E-generalization, was introduced in [6] and refined with a working implementation in [7]. The authors of [3] elaborate a *rigid anti-unification* algorithm that can apply to unordered (and so-called *unranked*) theories by instantiating a parameter called rigidity function, a direct application of which being the computation of longest common substrings. The algorithms described in all of these works can be used to compute what we will call $\sqsubseteq$-common generalizations below in the present paper. Although none of these works develop a general (non-equational) taxonomy allowing to extend the results beyond that simple setting, nor discusses variable- or injectivity-based variants of anti-unification operators, their usages do point out other interesting (and recent) applications of anti-unification when focused on unordered goals, namely detection of recursion schemes in functional programs (as explained in [2]) and techniques for learning bugfixes from software code repositories (an example being [26]).

Anti-unification techniques that are adapted for CLP(X) have been defined in [29], but its focus is set on a polynomial abstraction procedure for a specific case where terms cannot be generalized (only variables can) and where generalization has to be carried out through injective substitutions. While [29] provides useful insights and results, it lacks a more general and in-depth study of the used generalization operator. In this work we broaden, generalize and complete the latter work by providing a detailed and systematic study of generalization operators and their characteristics in the context of CLP.

The main contributions of the present work are the following. In Section 2 we define relations close to the well-known θ-subsumption in an effort of adapting this notion to the case of unordered goals. As will be illustrated throughout the paper, our adaption of anti-unification to unordered goals makes the usual subsumption techniques unusable. In Section 3 we reframe the problem of looking for a most general/largest generalization as an optimization problem, parametrized by the *generalization operator* (or anti-unification strategy) and *variabilization function* (responsible for introducing variables in the resulting generalization) at hand. We will see that given two unordered goals as input, searching for such generalizations can be done in polynomial time. The algorithms, as well as their worst-case time complexities, are detailed throughout the development of our anti-unification framework. In Section 4 we provide an in-depth examination of several key variations of the anti-unification problem, namely variable generalization (where no terms are allowed to be generalized), injective generalization (where the generalizing substitutions need to be injective) and dataflow optimization (where the number of generalizing variables needs to be minimized) – the latter of which is proved to make the anti-unification statement NP-hard. Finally, addressing this last problem more in depth in Section 5 we revisit a tractable abstraction that was introduced in [29] but we provide for the first time a formal proof of its worst-case complexity, showing that the approximation can effectively be computed in polynomially bounded time. With the exception of this last result, the proofs of propositions, lemmas and theorems are provided in the Appendices.

2 Preliminaries

In the following, we introduce concepts and notations that will be used throughout the paper. We suppose a language of Horn clauses defined over a context, which is a 4-tuple $\langle X, \mathcal{V}, \mathcal{F}, \mathcal{Q} \rangle$, where X is a non-empty set of constant values, $\mathcal{V}$ is a set of variable names, $\mathcal{F}$ a set of function names and $\mathcal{Q}$ a set of predicate symbols. The sets $X, \mathcal{V}, \mathcal{F}$ and $\mathcal{Q}$ are all supposed to be disjoint sets. Symbols from $\mathcal{F}$ and $\mathcal{Q}$ have an associated arity (i.e. its number of arguments) and

we write f/n to represent a symbol f having arity n. Given a context $\mathcal{C} = \langle X, \mathcal{V}, \mathcal{F}, \mathcal{Q} \rangle$, we define the set of *terms* over it as $\mathcal{T}_\mathcal{C} = X \cup \mathcal{V} \cup \{f(t_1, t_2, \ldots, t_n) | f/n \in \mathcal{F} \wedge \forall i \in 1..n : t_i \in \mathcal{T}_\mathcal{C}\}$. Terms are thus ground domain constants, variables and functor-based expressions over other terms. In what follows we will use uppercase symbols to represent variables whereas lowercase symbols will be used for function and predicate symbols. The set of *atoms* over $\mathcal{C}$ is defined as $\mathcal{A}_\mathcal{C} = \{p(t_1, \ldots, t_n) \mid p/n \in \mathcal{Q} \wedge \forall i \in 1..n : t_i \in \mathcal{T}_\mathcal{C}\}$. An atom $p(t_1, \ldots, t_n)$ is understood as representing an atomic formula involving the predicate p over n arguments, the arguments being represented by terms. A *goal* G is a set of atoms, representing an (unordered) conjunction, thus $G \subseteq \mathcal{A}_\mathcal{C}$.

▶ **Example 1.** Let us consider a numerical context (e.g. $X = \mathbb{Z}$ and $\mathcal{F}$ is the set of usual functions over integers composed of addition ($+$), substraction ($-$), integer division ($/$), multiplication ($*$) and modulo ($\%$)). Supposing X and Y to represent variables, then the following are terms: 3, X, $+(3, X)$, $+(4, *(X, \%(Y, 2)))$. Given predicates $p/1$, $q/1$, $r/2$ and $c/2$, the following are atoms: $p(3)$, $q(X)$, $r(+(2, 4), +(3, X))$

In what follows we will often leave the underlying context implicit and simply talk about variables, function and predicate symbols. A *substitution* is a mapping from variables to terms and will be denoted by a Greek letter. For any substitution $\sigma : \mathcal{V} \mapsto \mathcal{T}_\mathcal{C}$, $dom(\sigma)$ represents its domain, $img(\sigma)$ its image, and for a program expression e (be it a term, an atom or a goal) and a substitution σ, we write $e\sigma$ to represent the result of substitution application, i.e. simultaneously replacing in e those variables V that are in $dom(\sigma)$ by $\sigma(V)$. A *renaming* is a special kind of substitution, mapping variables to variables only. Thus for any renaming ρ we have that $img(\rho) \subseteq \mathcal{V}$. We can now define what constitutes a generalization relation $\sqsubseteq$, which essentially defines a goal as more general than another if the latter is a potentially larger and potentially more instantiated goal than the former.

▶ **Definition 2.** *Let G and G' be goals. G is a* generalization *of G' if and only if there exists θ, a substitution such that $G\theta \subseteq G'$. We denote this fact by $G \sqsubseteq G'$ (or sometimes $G \sqsubseteq_\theta G'$ if we want to emphasize the substitution θ in question).*

▶ **Example 3.** $\{p(X, Y, Z)\}, \{q(a(X))\}$ and $\{p(t(1), Y, u(Z)), q(W)\}$ are generalizations of $\{p(t(1), t(2), u(+(4, X))), q(a(t(u(1))))\}$.

In some applications (e.g. for some usual computation domains in Constraint Logic Programming), it makes sense to use a more restricted generalization relation, in which variables are substituted by other variables rather than terms. As such, when the substitution θ in Definition 2 is a renaming, we say that G is a *variable generalization* of G', which we denote by $G \preceq G'$ (or sometimes $G \preceq_\theta G'$ to emphasize the renaming θ in question). When considering the relation $\preceq$, only variables are generalized and the function symbols are considered as being a part of the language structure itself (i.e. they are not subject to generalization). This can be advantageous, for instance in applications working with a small finite domain such as Booleans, where considering $G = \{=(A, B)\}$ to be a generalization of both $\{=(X, true)\}$ and $\{=(Y, false)\}$ can feel like ignoring too much of the goal's semantics.

Our generalization relations are variations of the classical θ-subsumption [25], adapted to goals being sets rather than ordered sequences of atoms. They share the following property with θ-subsumption.

▶ **Proposition 4.** *Relations $\sqsubseteq$ and $\preceq$ are quasi-orders.*

We will now turn our attention towards the basic concept in anti-unification, namely that of a goal being a *common generalization* of some given goals [25]. In the following, we restrict ourselves to common generalizations of *two* goals, but the concept can straightforwardly be

extended to any number of goals. As for notation, when a result or definition holds for both our relations $\preceq$ and $\sqsubseteq$, for the sake of simplicity we will sometimes use $\leqslant$ to denote both relations at once.

▶ **Definition 5.** *Let $G_1, \ldots, G_n$ be goals and $\leqslant$ a generalization relation. Then G is a* $\leqslant$-common generalization *of $\{G_1, \ldots, G_n\}$ if and only if $\forall i \in 1..n : G \leqslant G_i$.*

The definition essentially states that each $G_i (1 \leq i \leq n)$ can be generalized by G through its own substitution. Formally there exist $\theta_1, \ldots, \theta_n$ such that $\forall i \in 1..n : G \sqsubseteq_{\theta_i} G_i$. A common generalization of goals is thus, in essence, a part of their shared atomic structure, with a possible introduction of variables in certain places – the liberality of which depends on the underlying relation. Note that renamings being (restricted) substitutions, for any two goals G and G' it holds that $G \preceq_\theta G' \Rightarrow G \sqsubseteq_\theta G'$ so that if a goal is a $\preceq$-common generalization of a set of goals it is also a $\sqsubseteq$-common generalization of said goals.

▶ **Example 6.** Let $G_1 = \{p(t(X), Y), q(3, f(X))\}$ and $G_2 = \{p(5, Z), q(3, f(Z))\}$. The following is a (non-exhaustive) list of $\sqsubseteq$-common generalizations of G_1 and G_2: $\emptyset, \{p(V_1, V_2)\}$, $\{q(3, f(V_1))\}, \{p(V_1, V_2), q(V_3, V_4)\}, \{p(V_1, V_2), q(3, V_3)\}$. The following are $\preceq$-common generalizations of G_1 and G_2 as well: $\emptyset, \{q(3, f(V_1))\}$.

As a slight lexical abuse, given atoms $\{A_1, \ldots, A_n\}$ we will say that an atom A is a $\leqslant$-common generalization of $\{A_1, \ldots, A_n\}$ iff $\{A\}$ is a $\leqslant$-common generalization of $\{A_1, \ldots, A_n\}$. Note that no matter the relation and no matter the goals G_1 and G_2, at least one common generalization will always exist: the empty goal $\emptyset$. Obviously, wherever possible we are interested in more detailed representations of the common structure found in goals.

For an expression e, we use $vars(e)$ to represent the set of variables that appear in e and $\tau(e)$ to denote the multiset of all atoms and non-variable terms occurring in e. We will sometimes refer to the cardinality of $\tau(e)$ as the τ-value of e. The multiset of all atoms and terms, variables included, is denoted by $ter(e)$.

▶ **Example 7.** Let G be the goal $\{p(f(x, Y)), q(Y, X)\}$. The multiset $\tau(G)$ is equal to $\{p(f(x, Y)), f(x, Y), x, q(Y, X)\}$. G's τ-value is 4, $vars(G) = \{X, Y\}$ and $ter(G)$ is the multiset $\{p(f(x, Y)), f(x, Y), x, Y, q(Y, X), Y, X\}$.

One is typically interested in those common generalizations that are the *most specific*, i.e. that capture as much common structure as possible amongst G_1 and G_2 [25].

▶ **Definition 8.** *Given goals $G_1, \ldots, G_n$ and G such that G is a $\leqslant$-common generalization of $\{G_1, \ldots, G_n\}$, we say that G is a* $\leqslant$-most specific generalization ($\leqslant$-msg) *of $\{G_1, \ldots, G_n\}$ if $\nexists G'$, another $\leqslant$-common generalization of $\{G_1, \ldots, G_n\}$, such that $|\tau(G')| > |\tau(G)|$.*

▶ **Example 9.** Consider again the goals G_1 and G_2 from Example 6. It is easy to see that $G = \{p(V_1, V_2), q(3, f(V_3))\}$ has a higher τ-value than all the other common generalizations listed in the example; G is in fact a $\sqsubseteq$-msg of G_1 and G_2, and in this case, all other msg's of G_1 and G_2 differ from G only in a renaming of the variables V_1, V_2 and V_3. As for relation $\preceq$, the goal $\{q(3, f(V_1))\}$ as well as its variants with V_1 renamed are $\preceq$-msg's of G_1 and G_2.

A weaker yet useful measure for comparing common generalizations is the number of atoms (i.e. the cardinality) of the common generalization G.

▶ **Definition 10.** *Given goals $G_1, \ldots, G_n$ and G such that G is a $\leqslant$-common generalization of $\{G_1, \ldots, G_n\}$, we say that G is a* $\leqslant$-largest common generalization ($\leqslant$-lcg) *of $\{G_1, \ldots, G_n\}$ if $\nexists G'$, another $\leqslant$-common generalization of $\{G_1, \ldots, G_n\}$, such that $|G'| > |G|$.*

▶ **Example 11.** Let us again take a look at the goals from Example 6. Each goal of size 2 (such as $\{p(V_1, V_2), q(V_3, V_4)\}$) is a $\sqsubseteq$-lcg, seeing that no larger $\sqsubseteq$-common generalization can exist as $|G_1| = |G_2| = 2$. Regarding the $\preceq$ relation, common generalizations of size 1 (e.g. $\{q(3, f(V_1))\}$) are the largest that exist in the example since the atoms involving $p/2$ have no $\preceq$-common generalization because of the structural difference in their first argument.

Before we can dive into the process of computing common generalizations, a few more preliminary observations need to be assessed regarding relations $\sqsubseteq$ and $\preceq$. First, we state that there is no other way for a common generalization to be most-specific than to harbor as many atoms as possible.

▶ **Proposition 12.** *Any $\leqslant$-msg is a $\leqslant$-lcg and any $\preceq$-lcg is a $\preceq$-msg.*

▶ **Example 13.** Let us consider $G_1 = \{a(Y, Z), a(t(1), X)\}$ and $G_2 = \{a(t(1), E)\}$ as well as $G = \{a(t(1), V_1)\}$. It is easy to see that G (and all its variations with V_1 renamed) is the only $\preceq$-lcg (thus $\preceq$-msg), as G_2's atom can only be anti-unified with the atom in G_1 that has the same structure – and so the same τ-value. Here, G is also a $\sqsubseteq$-msg (thus a $\sqsubseteq$-lcg).

Regarding $\sqsubseteq$, the converse of the above proposition ("any $\sqsubseteq$-lcg is a $\sqsubseteq$-msg") is not true, as shown by the following example. Let us consider $G_1 = \{a(Y, Z), a(t(1), X)\}$ and $G_2 = \{a(t(1), E)\}$ as well as the following $\sqsubseteq$-lcg's: $G = \{a(V_1, V_2))\}$ and $G' = \{a(t(1), V_1)\}$. Obviously $|\tau(G')| = 3 > |\tau(G)| = 1$. In fact, G' is a $\sqsubseteq$-msg for this example.

For a set of goals $\{G_1, \ldots, G_n\}$, we have defined most specific and largest generalizations using the plural. In fact, by the definitions above and as appears clearly in our examples, $G_1, \ldots, G_n$ can have more than one $\preceq$-lcg (and equivalently $\preceq$-msg), but all are equivalent modulo a variable renaming. The same does not necessarily hold with the relation $\sqsubseteq$: there might exist more than one sensibly different $\sqsubseteq$-lcg's, depending on the degree at which the different terms are abstracted away through the generalizations process. The following example shows that a similar observation holds for $\sqsubseteq$-msg's.

▶ **Example 14.** Consider the goals $G_1 = \{p(t, u)\}$ and $G_2 = \{p(t, X), p(X, u)\}$. There are two possible structures of $\sqsubseteq$-msg's, namely $\{p(t, V_1)\}$ and $\{p(V_1, u)\}$. There is one more possible structure of $\sqsubseteq$-lcg, namely $\{p(V_1, V_2)\}$

For the sake of clarity, in the results and discussions that follow we will simplify and consider common generalizations of *two* goals, but the ideas are straightforwardly applicable to groups of more than two goals. Furthermore, when discussing the generalization process of two goals we will suppose that the goals in question share no common variable name. This hypothesis is by no means a loss of generality as renaming all variables from one goal into fresh, unused variable names can ensure this property while not altering the goal's semantics.

3 Large and Specific Generalizations

In this section we prove that msg's and lcg's as defined above can be computed with polynomial-time algorithms. First, we need the concept of a *variabilization* which is basically a function mapping couples of terms to new variables.

▶ **Definition 15.** *Given a context $\langle X, \mathcal{V}, \mathcal{F}, \mathcal{Q}\rangle$, let $V \subset \mathcal{V}$ denote a set of variables. A function $\Phi_V : \mathcal{T}^2 \mapsto \mathcal{V} \cup X$ is called a* variabilization function *if, for any $(t_1, t_2) \in \mathcal{T}^2$ it holds that if $\Phi_V(t_1, t_2) = v$, then (1) $v \notin V$, (2) $\nexists (t'_1, t'_2) \in \mathcal{T}^2 : (t'_1, t'_2) \neq (t_1, t_2) \wedge \Phi_V(t'_1, t'_2) = v$, (3) $v \in X \Leftrightarrow t_1 = t_2 \in X$ and in that case, $v = t_1 = t_2$.*

Note that a variabilization function Φ_V introduces a new variable (not present in V) for any couple of terms, except when the terms are the same constant. It can thus be seen as a way to introduce new variable names when going through the process of anti-unifying two goals. In what follows, when manipulating goals G_1 and G_2, we will use $\Phi_{vars(G_1 \cup G_2)}$ to represent an arbitrary variabilization function. If the goals at hand are clearly identified from the context, we will abbreviate the notation to Φ. In most upcoming examples we will use applications of Φ (e.g. $\Phi(X, Y), \Phi(t(X), 5), \dots$) rather than coined variable names (e.g. $V_1, V_2, \dots$) when an anti-unification operator is – ostensibly or not – at work.

Algorithm 1 shows the intuitive solution for computing a lcg with two goals G_1 and G_2 (where we suppose $|G_1| \leq |G_2|$) as input. In the algorithm, $\mathtt{au}_{\leqslant}(A_1, A_2)$ denotes the use of a function that outputs a $\leqslant$-common generalization on the atomic level for atoms A_1 and A_2 with respect to relation $\leqslant$. In our development we will call such functions *anti-unification operators*. As stated in the following observation, such operators exist for our relations.

Algorithm 1 Computing a lcg G for goals G_1 and G_2 with generalization relation $\leqslant$.

$G = \{\}, R = \{\}$
for each $(A_1 \in G_1)$ **do**
  for each $(A_2 \in G_2 \setminus R)$ **do**
    $A'_1 = \mathtt{au}_{\leqslant}(A_1, A_2)$
    if $A'_1 \neq \bot$ **then**
      $G \leftarrow G \cup A'_1$
      $R \leftarrow R \cup A_2$
      break out of the inner loop
return G

▶ **Lemma 16.** *There exist polynomial anti-unification operators to compute the $\leqslant$-lcg and/or the $\leqslant$-msg of two atoms. In particular for two atoms A_1 and A_2, there exist (1) an operator $\mathtt{au}_{\sqsubseteq}(A_1, A_2)$ computing a $\sqsubseteq$-lcg for A_1 and A_2 in $\mathcal{O}(n)$ with n the arity of A_1; (2) an operator $\mathtt{au}_{\preceq}(A_1, A_2)$ computing a $\preceq$-lcg in $\mathcal{O}(m)$ with m the maximum number of function applications in the argument terms of the atom A_1; (3) an operator $\mathtt{dau}_{\sqsubseteq}(A_1, A_2)$ computing a $\sqsubseteq$-msg with a complexity that is linear in the number of terms appearing in A_1.*

Algorithm 1 merely applies a given anti-unification operator to pairs of atoms and keeps the results (if not $\bot$) in the generalization under construction, leading to the conclusion:

▶ **Theorem 17.** *Given two goals G_1 and G_2, Algorithm 1 can compute (1) a $\sqsubseteq$-lcg in $\mathcal{O}(|G_1| \cdot |G_2| \cdot N)$ with N the maximum arity of the predicate symbols occurring in G_1 and G_2; (2) a $\preceq$-lcg in $\mathcal{O}(|G_1| \cdot |G_2| \cdot N)$ with $M = \max_{A \in G_1}\{|ter(A)|\}$.*

Note that although Algorithm 1 is able to find a $\sqsubseteq$-lcg for two goals G_1 and G_2, it can produce different lcg's depending on the order in which the atoms of G_1 and G_2 are considered. Although the $\preceq$-lcg computed by Algorithm 1 is necessarily a $\preceq$-msg (according to Proposition 12), the same observation does not hold when the underlying relation is $\sqsubseteq$ and the anti-unification operator is adapted accordingly. The fact that Algorithm 1 can miss out on a $\sqsubseteq$-msg is due to the algorithm itself not trying to match those pairs of atoms (A_1, A_2) that share as much structure as possible. Therefore, finding a $\sqsubseteq$-lcg with maximal τ-value (i.e. a $\sqsubseteq$-msg) can be seen as an optimization problem.

Indeed, applying Algorithm 1 as-is does not guarantee that the matched atoms from G_1 and G_2 are chosen in a way that optimizes the output's τ-value. The algorithm should be adapted in such a way that first, the anti-unification of A_1 and A_2 is computed for *all*

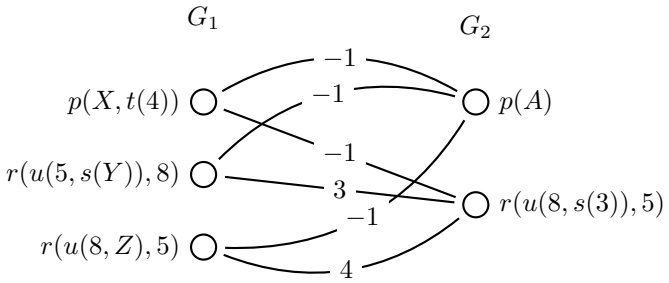


Figure 1 The bipartite graph for the assignment problem from Example 18.

$A_1 \in G_1$ and $A_2 \in G_2$; then, there must be a selection of pairs of atoms so that the resulting generalization has a maximized τ-value. This is similar to the well-known assignment problem, and can consequently be solved by existing maximization matching algorithms [8]. Indeed, with G_1 and G_2 the goals at hand, our problem can be characterized by drawing a weighted bipartite graph with as left vertexes the atoms of G_1 and as right vertexes the atoms of G_2. When considering as granted an operator $\mathtt{dau}^1_{\sqsubseteq}$ computing a $\sqsubseteq$-msg for two atoms, an edge between two vertexes A_1 and A_2 has an associated weight w indicating the potential benefit (in number of terms and predicate symbols) of anti-unifying A_1 and A_2, formally defined as

$$w(A_1, A_2) = \begin{cases} -1 & \text{if } \mathtt{dau}_{\sqsubseteq}(A_1, A_2) = \bot \\ |\tau(\mathtt{dau}_{\sqsubseteq}(A_1, A_2))| & \text{otherwise.} \end{cases}$$

Since all edges are labeled by a measurement of their τ-value, the maximum weight matching (MWM) in the bipartite graph will give the selection of pairs of atoms that, once properly anti-unified, keep the maximal structure in the generalization. Observe that by giving negative scores to atom couples that do not anti-unify, we prevent these couples from playing any part in the computed generalization.

▶ **Example 18.** Let us consider the goals $G_1 = \{p(X, t(4)), r(u(5, s(Y)), 8), r(u(8, Z), 5)\}$ and $G_2 = \{p(A), r(u(8, s(3)), 5)\}$. The corresponding assignment problem is shown in Figure 1. The MWM consists of the sole edge $(r(u(8, Z), 5), r(u(8, s(3)), 5)$, so that the resulting generalization for this simple example is $G = \{r(u(8, \Phi(Z, s(3))), 5)\}$.

▶ **Theorem 19.** *Let G_1 and G_2 be goals and $N = \max_{A \in G_1}\{|ter(A)|\}$. Then a $\sqsubseteq$-msg of G_1 and G_2 can be computed in $\mathcal{O}\big(|G_1| \cdot |G_2| \cdot N + max(|G_1|, |G_2|)^3\big)$.*

Note that the process described above finds *a* $\sqsubseteq$-msg but there is no guarantee regarding *which* $\sqsubseteq$-msg is found: as previously observed, the maximal τ-value can sometimes be reached through different atomic structures. Another inconstant parameter from one msg to the other is the number of *different* variables that are introduced in the generalization process. In fact, both aspects can sometimes be related, for example when minimizing the number of variables leads to the choice of a certain msg structure over another. A $\sqsubseteq$-most specific generalization that has *as few* different variables as possible is often seen as an even more specific generalization; the computation of such a msg is the main topic of the following section.

[1] For *deep anti-unification*.

4 Dataflow Optimization

Relations $\sqsubseteq$ and $\preceq$ are defined over substitutions that do not necessarily need to be *injective*. Indeed, a single term occurring multiple times in one of the goals can potentially be generalized by two (or more) different variables. Therefore, some most specific generalizations may contain more different variables than others depending on the underlying variabilization process. Among two common generalizations of the same pair of goals, the common generalization that has more variables than the other can be considered *less specific* as some information – namely the fact that two or more values, possibly in different atoms, are equal – has been abstracted by introducing different variables. In what follows, we will call the search of a common generalization with as few different variables as possible *dataflow optimization*. The following example illustrates the concept over the finite domain from [10].

▶ **Example 20.** Consider the domain of Booleans $\mathbb{B} = \{true, false\}$ as well as the following goals: $G_1 = \{=(X, or(Y, Z)), =(V, and(Y, Z))\}$ and $G_2 = \{=(B, or(C, D)), =(A, and(C, D)), =(E, and(F, G))\}$. Note that in G_1 the *or* and *and* operations are evaluated on the same values, represented by the multiple occurrences of the variables Y and Z. In G_2 the *or* and the *and* operation from the second atom exhibit this very same behavior (represented by the variables C and D), whereas the third atom represent an *and* operation on different values. Computing a $\preceq$-msg (and in this example, a $\sqsubseteq$-msg) for G_1 and G_2 can lead to two different generalizations, namely

$$\begin{aligned} G &= \{=(\Phi(X,B), or(\Phi(Y,C),\Phi(Z,D))), =(\Phi(V,E), and(\Phi(Y,F),\Phi(Z,G)))\} \\ G' &= \{=(\Phi(X,B), or(\Phi(Y,C),\Phi(Z,D))), =(\Phi(V,A), and(\Phi(Y,C),\Phi(Z,D)))\}. \end{aligned}$$

Clearly, both generalizations are correct msg's, but the fact that all the variables in G only occur once merely denotes that there exist six variables that together can make G true. The repetition of Y and Z in G_1 as well as their connection with C and D is a lost information, abstracted by the anti-unification process. On the other hand, G' by harboring less different variables introduces less variable abstraction, effectively depicting some dataflow logic that is common to G_1 and G_2, through the occurrence of $\Phi(Y, C)$ and $\Phi(Z, D)$ in both its atoms. On that level, G' can be considered less general than G.

Dataflow optimization thus formally boils down to finding, among a group of common generalizations for two goals G_1 and G_2, a goal G such that $|vars(G)|$ is minimal. In Example 20, we were interested in finding, among all possible msg's of G_1 and G_2, one that harbors a minimal number of variables; it makes sense, since abstracting one Boolean value with two different variables can be too liberal, depending on the applications. In that case of dataflow optimization, where the target goal must be a msg (i.e. when both structure and dataflow must be optimized), the dataflow problem is NP-complete. The same is true for lcg's. In order to show this formally, we consider a formulation in terms of decision problems.

▶ **Theorem 21.** *Let MSG-MIN (resp. LCG-MIN) denote the following decision problem: "Given goals G_1, G_2 and a constant $p \in \mathbb{N}_0$, does there exist a $\leqslant$-msg (resp. $\leqslant$-lcg) of G_1 and G_2 that has less than p different variables?" MSG-MIN and LCG-MIN are NP-complete.*

Now instead of looking to *minimize* the number of different variables in the computed generalization G, one could be interested in *forcing* to preserve all the dataflow implied in the generalized goals, not allowing to abstract away the links that appear in the goals' terms. Intuitively, this can be done by forbidding any term from one of the input goals to have more than one "corresponding term" in the other input goal. In other words, the

dataflow is considered entirely preserved if the underlying variabilization function Φ doesn't associate any term with two or more different terms at the same time. Formally, this amounts to using an *injective version* of our generalization relations. We say that a generalization relation is injective if its definition only holds for injective substitutions. For a common generalization G of goals G_1 and G_2 and for some function Φ associating fresh variable names to couples of variables, this implies when using an anti-unification algorithm (e.g. Algorithm 1) that for any two different variables $\Phi(T_1, T_2)$ and $\Phi(T_3, T_4)$ appearing in G, it holds that $T_1 \neq T_3 \neq T_2 \neq T_4 \neq T_1$. We will denote by $\sqsubseteq^\iota$ (resp. $\preceq^\iota$) the versions of $\sqsubseteq$ (resp. $\preceq$) that exhibit this property.

▶ **Example 22.** Consider the injective relation $\preceq^\iota$ as well as the goals $G_1 = \{and(A, B), or(B, C), xor(C, A)\}$ and $G_2 = \{and(X, Z), or(Y, X), xor(Z, Y)\}$. The only common generalizations are $\emptyset$, $\{and(\Phi(A, X), \Phi(B, Z))\}, \{or(\Phi(B, Y), \Phi(C, X))\}$ and $\{xor(\Phi(C, Z), \Phi(A, Y))\}$. No common generalization of size larger than 1 exists, since (at least) one of the matching substitutions is not injective. For example, the goal $G = \{and(\Phi(A, X), \Phi(B, Z)), or(\Phi(B, Y), \Phi(C, X))\}$ is not a common generalization of G_1 and G_2, since (at least) one of the substitutions mapping this goal to G_1 or G_2 is not injective. Indeed, the substitution $[\Phi(A, X) \mapsto A, \Phi(B, Z) \mapsto B, \Phi(B, Y) \mapsto B, \Phi(C, X) \mapsto C]$ maps both $\Phi(B, Z)$ and $\Phi(B, Y)$ to B; this is sufficient to reach the conclusion that G is not an injective generalization of G_1 and G_2. Note that in this case, the other potential substitution, i.e. the one mapping G on G_2, is not injective either.

The two following observations immediately result from the injective relations being more constrained versions of their non-injective counterparts.

▶ **Proposition 23.** *Relations $\sqsubseteq^\iota$ and $\preceq^\iota$ are quasi-orders.*

▶ **Proposition 24.** *Let G_1 and G_2 be goals. If $G_1 \sqsubseteq^\iota_\theta G_2$, then $G_1 \sqsubseteq_\theta G_2$. If $G_1 \preceq^\iota_\theta G_2$, then $G_1 \preceq_\theta G_2$ and $G_1 \sqsubseteq^\iota_\theta G_2$.*

With an injective generalization relation, the computing of a msg is fundamentally dissociated from that of an lcg, as an msg is not necessarily a lcg due to the injectivity constraint. However, both situations are intractable. In order to show this formally, we define the following decision problem variant.

▶ **Theorem 25.** *Let INJ denote the following decision problem: "Given an injective generalization relation $\leqslant^\iota$ along with goals G_1 and G_2 such that $|G_1| \leq |G_2|$, verify whether there exists an ad hoc injective substitution θ such that $G_1\theta \subseteq G_2$." INJ is NP-complete.*

INJ is basically the verification of whether a goal G_1 can be adequately mapped onto (a subset of) another goal G_2. If there exists a substitution θ (resp. a renaming ρ) making this possible, then G_1 is a $\sqsubseteq^\iota$- (resp. $\preceq^\iota$-)largest *and* most specific generalization of G_1 and G_2, since no larger nor structurally more specific goal than G_1 can exist for this specific situation.

Due to the inherent intractability of injective relations, it is sometimes preferable to make use of tractable abstractions rather than exact brute-force algorithms, especially if a quick and approximate (though entirely dataflow-preserving) anti-unification result suffices for the application at hand. In the next section, we give such an efficient – yet highly accurate – abstraction for the computation of $\preceq^\iota$-lcg's.

5 The k-swap Stability Abstraction

In what follows, we introduce an abstraction for the largest common generalization with respect to $\preceq^\iota$ that can be computed in polynomial time. The abstraction was already introduced in [29] but no formal proof of its complexity was given. The abstraction is based on the *k-swap stability* property, which is in turn defined in terms of *pairing generalizations*.

▶ **Definition 26.** *Let G_1 and G_2 be two renamed apart goals and G be a $\preceq^\iota$-common generalization of G_1 and G_2 such that $G \subseteq G_1$. Let ρ be any renaming such that $G\rho \subseteq G_2$. The* pairing generalization *of G, denoted $\pi(G)$, is the set of pairs $(A_1, A_2) \in G_1 \times G_2$ such that $\forall (A_1, A_2) \in \pi(G) : A_1\rho = A_2$.*

▶ **Example 27.** Considering the goals $G_1 = \{p(A), p(B), q(A)\}$ and $G_2 = \{p(X), q(Y)\}$, it is easy to see that $G = \{p(\Phi(B, X)), q(\Phi(A, Y))\}$ is a $\preceq^\iota$-common generalization of them. The corresponding pairing generalization is $\pi(G) = \{(p(B), p(X)), (q(A), q(Y))\}$.

The notion of a pairing generalization renders thus explicit the corresponding atoms from the generalized goals that contribute to the generalization. As a slight abuse of language, given a pairing generalization π of some generalization G for goals G_1 and G_2, we will simply say that π is a *pairing for G_1 and G_2*. Pairings can be used to express a notion of goal *stability* in the following sense.

▶ **Definition 28.** *Let G_1 and G_2 be two renamed apart goals and G be a $\preceq^\iota$-common generalization of G_1 and G_2 such that $G \subseteq G_1$. G is* k-swap stable *if and only if there does not exist some generalizations $\hat{G}$ and G' of G_1 and G_2 such that $\hat{G} \supset G'$ and $|\pi(G) \cap \pi(G')| \geq |\pi(G)| - k$ for some $k \in \mathtt{N}$.*

Intuitively, a generalization G is k-swap stable if it is impossible to transform G into a larger generalization $\hat{G}$ in spite of "swapping" at most k pairs in $\pi(G)$. This stability notion gives a characterization of the quality of a computed generalization. If a generalization is 0-swap stable (the weakest characterization), it cannot be extended by adding another atom but this guarantees in no way that a larger generalization could not be found. If a generalization G is k-swap stable (for $k > 0$), it means that even if we exchange up to k pairs in $\pi(G)$ by others, the generalization cannot be extended into a larger one. Consequently, if a generalization is k-swap stable for k the number of atoms in the smallest of the two goals (denoted by ∞-swap stable), it means that the computed generalization is a largest common generalization. Operationally, when naively searching for a lcg by backtracking, the fact that a computed generalization is k-swap stable means that one should backtrack by *more* than k choice points in order have a chance of finding a larger generalization.

▶ **Example 29.** Consider the goals $G_1 = \{add(X, Y, Z), even(X), odd(Z), p(Z)\}$ and $G_2 = \{add(A, B, C), add(C, B, A), even(C), odd(A), p(C)\}$. $\pi_1 = \{(add(X, Y, Z), add(A, B, C))\}$ is not 0-swap stable. Indeed, we can enlarge π_1 by adding $(p(Z), p(C))$, in order to obtain $\pi_2 = \{(add(X, Y, Z), add(A, B, C)), (p(Z), p(C))\}$. Note that π_2 is 0-swap stable, it is impossible to add another pair to π_2 and still obtain a common generalization. It is also 1-swap stable, seeing that replacing (or removing) one of the pairs doesn't lead to a pairing readily extensible to a pairing of size strictly greater than 2. However, π_2 is not 2-swap stable. Indeed, replacing the pair $(add(X, Y, Z), add(A, B, C))$ by the pair $(add(X, Y, Z), add(C, B, A))$ in π_2 and removing the now incompatible pair $(prime(Z), prime(C))$ (i.e. choosing the renaming $[X \mapsto C, Y \mapsto B, Z \mapsto A]$ instead of $[X \mapsto A, Y \mapsto B, Z \mapsto C]$) gives rise to $\pi_2' = \{(add(X, Y, Z), add(C, B, A))$, which can readily be extended into $\pi_3 = \{(add(X, Y, Z), add(C, B, A)), (even(X), even(C)), (odd(Z), odd(A))\}$ which is a pairing of size 3. The latter being ∞-swap stable, it represents a $\preceq^\iota$-lcg, namely $\hat{G} = \{add(\Phi(X, C), \Phi(Y, B), \Phi(Z, A)), even(\Phi(X, C)), odd(\Phi(Z, A))\}$

An algorithm has been introduced in [29] that builds up a k-swap stable generalization using the process suggested in Example 29. Its practical performance has been assessed on different test cases. The tests indicate that the k-swap stability property represents a well-suited

approximation of the concept of $\preceq^{\iota}$-lcg. Indeed, in all test cases the size of the k-swap stable generalization was at least 90% of the size of an lcg for the same anti-unification problem, while the computational time was radically reduced – especially as the size of the input goals grows[2]. However, in [29] only pragmatical aspects have been explored; the theoretical foundations of the k-swap technique were not detailed, and no actual time complexity upper bound has been demonstrated. We fill this gap in the remainder of this section. First, we introduce the algorithm, then we formally prove that its time complexity is polynomially bounded. Before introducing the algorithm, which is essentially composed of two sub-algorithms, we give some notations that will facilitate their formulation. First, we define an operator that allows to combine two pairings into a single pairing.

▶ **Definition 30.** *Let G_1 and G_2 be two renamed apart goals. The* enforcement operator *is defined as the function $\triangleleft : (G_1 \times G_2)^2 \mapsto (G_1 \times G_2)$ such that for two pairing generalizations π and π' for G_1 and G_2, $\pi \triangleleft \pi' = \pi' \cup M$ where M is the largest subset of π such that $\pi' \cup M$ represents a $\preceq^{\iota}$-common generalization of G_1 and G_2.*

In other words, $\pi \triangleleft \pi'$ is the mapping obtained from $\pi \cup \pi'$ by eliminating those pairs of atoms (A, A') from π that are *incompatible* with some $(B, B') \in \pi'$ either because they concern the same atom(s) or because the involved renamings cannot be combined into a single injective renaming.

▶ **Example 31.** Consider $\pi = \{(p(X,Y), p(A,B)), (q(X), q(A))\}$ as a pairing for two goals G_1 and G_2. Suppose $\pi' = \{(r(Y), r(C))\}$ is also a pairing for G_1 and G_2. Enforcing π' into π gives $\pi \triangleleft \pi' = \{(q(X), q(A)), (r(Y), r(C))\}$. Indeed, this can be seen as forcing Y to be mapped on C; therefore the resulting pairing generalization can no longer contain $(p(X,Y), p(A,B))$ as the latter maps Y on B.

For π_1 and π_2 pairings we will also denote by $comp_{\pi_1}(\pi_2)$ the subset of π_2 of which each element can be added to π_1 such that the result is a pairing (i.e. there is no injectivity conflict in the associated renaming). Finally, we use $gen(G_1, G_2)$ to represent those atoms from G_1 and G_2 that are variants of each other, formally $gen(G_1, G_2) = \{(A, A') \mid A \in G_1, A' \in G_2$ and $A\rho = A'$ for some renaming $\rho\}$. The first algorithm is depicted in Algorithm 2. The algorithm represents the construction of a k-swap stable generalization of goals G_1 and G_2. At each round, the process tries to transform the current generalization π (which initially is empty) into a larger generalization by forcing a new pair of atoms (A, A') from $gen(G_1, G_2)$ in π, which is only accepted if doing so requires to swap no more than k elements in π. More precisely, the algorithm selects a subset of π (namely π_s) that can be swapped with a subset π_c of the remaining mappings from $gen(G_1, G_2) \setminus \pi$ such that the result of replacing π_s by π_c in π and adding (A, A') constitutes a pairing. Note how condition 1 in the algorithm expresses that π_s must include at least those elements from π that are not compatible with (A, A'). The search continues until no such (A, A') can be added.

The main operation of Algorithm 2, namely the selection of π_s and π_c, is detailed in Algorithm 3 which aims to select the parts of the pairings to be swapped in order to enlarge the resulting pairing under construction (π) by the couple (A, A'). To that purpose π_s is initialized with the part of π that is incompatible with the pair of atoms (A, A') that we

[2] For example, with k fixed to 4, anti-unifying goals harboring 15 to 22 atoms, each of arity between 1 and 3, comes on average down from more than 7 minutes (using bruteforce) to 272 milliseconds (using the algorithms presented in this section), while the size of the computed generalization is on average 95% of the size of a lcg. More detailed test results are exposed in [29].

Algorithm 2 Computing a k-swap stable generalization G for goals G_1 and G_2.

$\pi \leftarrow \emptyset$
repeat
  $found \leftarrow false$
  for all (A, A') in $gen(G_1, G_2) \setminus \pi$ **do**
    select $\pi_s \subseteq \pi$ and $\pi_c \subseteq gen(G_1, G_2) \setminus (\pi \cup \{(A, A')\})$ such that:
      (1) $\pi_s \supseteq \pi \setminus \pi \triangleleft \{(A, A')\}$
      (2) $|\pi_s| \leq k$
      (3) $|\pi_c| = |\pi_s|$
      (4) $\pi \setminus \pi_s \cup \pi_c \cup \{A, A'\}$ is a pairing generalization of G_1 and G_2
    if such π_c and π_s are found **then**
      $\pi \leftarrow \pi \setminus \pi_s \cup \pi_c \cup \{(A, A')\}$
      $found \leftarrow true$
      break out of the **for** loop
until $\neg found$
$G \leftarrow dom(\pi)$

wish to enforce into the generalization. Its replacement mapping π_c is initially empty and the algorithm subsequently searches to construct a sufficiently large π_c (the inner while loop). During this search, S represents the set of candidates, i.e. couples from $gen(G_1, G_2)$ that are not (yet) associated to the generalization. In order to explore different possibilities with backtracking, the while loop manipulates a stack GS that records alternatives for π_c with the corresponding set S for further exploration.

If the search for π_c was without a satisfying result (i.e. no π_c is found equal in size to π_s), the algorithm continues by removing another couple from π (thereby effectively enlarging π_s). The rationale behind this action is that there might be a couple in π that is "blocking" the couples in S from addition to π. In order to achieve the removal of such potentially blocking couples, an arbitrary couple from $\pi \setminus \pi_s$ is selected, and alternatives are recorded in a queue (BS). Note the use of a queue (and its associated operations *enter* and *exit*) as opposed to the stack GS. The process is repeated until either $|\pi_c| = |\pi_s|$ in what case we have found a suitable k-swap, or until $|\pi_s| > k$ in what case we have not, and the algorithm returns $\perp$.

While the algorithms have been proven to correctly compute a k-swap stable generalization [29], no result on their complexity has yet been formally established.

▶ **Theorem 32.** *For a given and constant value of k, the combination of Algorithms 2 and 3 computes a k-swap stable common generalization of input goals G_1 and G_2 in polynomial time $\mathcal{O}((\alpha M)^{k+1})$, with $0 \leq M \leq |gen(G_1, G_2)|$ and $0 \leq \alpha \leq min(|G_1|, |G_2|)$.*

Proof. In order to search for a suited π_c to be swapped with a certain π_s, Algorithm 3 must try to add $|\pi_s|$ couples to $\pi \setminus \pi_s$ among the couples in S that are compatible with it. To simplify notation, let $i = |\pi_s|$ and $n = |comp_{\pi \setminus \pi_s \cup \pi_c}(S)|$. Note that at any moment $i \leq k$. The attempt of Algorithm 3 to find π_c is essentially a search of a combination of i couples among n; that is $\binom{n}{i}$ possibilities to explore. We have $\binom{n}{i} = \frac{n!}{i!(n-i)!}$ which reduces to a polynomial of degree n^i:

$$\frac{n!}{i!(n-i)!} = \frac{n \cdot (n-1) \cdot \cdots \cdot (n-(i+1)) \cdot (n-i) \cdot (n-(i-1)) \cdot \cdots \cdot 1}{i! \cdot (n-i) \cdot (n-(i-1)) \cdot \cdots \cdot 1} = \frac{n \cdot (n-1) \cdot \cdots \cdot (n-(i+1))}{i!} \approx \mathcal{O}(n^i)$$

If no suiting π_c is found during such a search, then π_s gets enlarged, having its size m increased by (at least) one unit. In the worst case, the size i of π_s is, at the start of Algorithm 3, equal to 1. It then gets incremented by one, until it reaches k (each time more

Algorithm 3 Selecting π_s and π_c for a given (A, A').

$GS \leftarrow \{\}$, $BS \leftarrow \{\}, \pi_c \leftarrow \{\}$
$\pi_s \leftarrow \pi \setminus \pi \triangleleft \{(A, A')\}$
$S \leftarrow gen(G_1, G_2) \setminus \pi \triangleleft \{(A, A')\}$
while $|\pi_c| < |\pi_s|$ and $|\pi_s| \leq k$ **do**
  while $|\pi_c| < |\pi_s|$ and $\neg(comp_{\pi \setminus \pi_s \cup \pi_c}(S) = \{\}$ and $GS = \{\})$ **do**
    for all p in $comp_{\pi \setminus \pi_s \cup \pi_c}(S)$ **do**
      $push(GS, (\pi_c \cup p, S \setminus \{p\}))$
    $(\pi_c, S) \leftarrow pop(GS)$
  if $|\pi_c| < |\pi_s|$ **then**
    for all p in $\pi \setminus \pi_s$ **do**
      $enter(BS, \pi_s \cup \{p\})$
    if $BS \neq \{\}$ **then**
      $\pi_s \leftarrow exit(BS)$
      $\pi_c \leftarrow \{\}$
      $S \leftarrow gen(G_1, G_2) \setminus (\pi \cup \{(A, A')\})$
    else
      return $\bot$
if $|\pi_c| = |\pi_s|$ **then**
  return π_s, π_c
r**return** $\bot$

atoms from π being considered to be part of π_s). Let p denote the size of the pairing π under construction, that is $p = |\pi|$. As k is constant, if backtracking is exhaustive there are $\sum_{i=1}^{k} \binom{p}{i}$ possibilities for π_s pairings that are explored this way. Each of these π_s pairings leads to the search for a corresponding π_c pairing. As such, the overall search carried out by Algorithm 3 takes a number of iterations that is in the worst case represented by

$$\sum_{i=1}^{k} \binom{p}{i} \cdot \binom{n}{i} \quad \approx \quad \sum_{i=1}^{k} \mathcal{O}(p^i) \cdot \mathcal{O}(n^i) \quad \approx \quad \mathcal{O}((p \cdot n)^k)$$

Given that n is bound by the number of compatible couples of atoms from $G_1 \times G_2$, we will denote the worst-case time complexity of Algorithm 3 by $\mathcal{O}((p \cdot M)^k)$ with $M \leq |gen(G_1, G_2)|$ and p the length of the pairing under construction π.

Turning our attention to Algorithm 2 it is clear that the size of pairing π is incremented by 1 in each iteration of the *repeat*-loop, since $found$ must be true for a new iteration to occur. As such, in the worst-case scenario there can be as many iterations as there are atoms in the smallest goal amongst G_1 and G_2, seeing that a generalization size cannot exceed that of the goals it generalizes. We will denote this number by $\alpha = \min(|G_1|, |G_2|)$. As for the inner loop of Algorithm 2, it can browse through up to $|gen(G_1, G_2)| - p$ candidates for choosing the couple (A, A') that will be enforced in the pairing π. This gives us at most $\sum_{p=1}^{\alpha} (|gen(G_1, G_2)| - p) \quad \approx \quad \sum_{p=1}^{\alpha} \mathcal{O}(M - p)$ iterations of Algorithm 2.

Algorithm 3 being called at each inner loop iteration of Algorithm 2, we can represent the time complexity of the combined algorithms by $\sum_{p=1}^{\alpha} \mathcal{O}(M - p) \cdot \mathcal{O}((p \cdot M)^k) \approx \sum_{p=1}^{\alpha} \left((M - p) \cdot p^k \cdot M^k\right)$ which can be rewritten as $M^{k+1} \cdot \left(\sum_{p=1}^{\alpha} p^k\right) - M^k \cdot \left(\sum_{p=1}^{\alpha} p^{k+1}\right)$.

Since $\sum_{p=1}^{\alpha} p^k \approx \mathcal{O}(\alpha^{k+1})$ and $\sum_{p=1}^{\alpha} p^{k+1} \approx \mathcal{O}(\alpha^{k+2})$, we can conclude the total complexity to be of the order $\mathcal{O}((\alpha \cdot M)^{k+1}) - \mathcal{O}(\alpha^{k+2} \cdot M^k)$ which proves the result. ◀

Whenever there is a need to compute numerous anti-unifications of unordered goals with limited time resources, the k-swap stability abstraction allows to keep the search space tractable while outputting goals that are, on average, close in size to that of a lcg. Such situations can e.g. arise in static analysis techniques for large Horn clause programs, such as the assessment of structural similarity between algorithms expressed in CLP [28].

6 Conclusions and Future Work

In this work, we have systematically studied different key notions and results concerning anti-unification of unordered goals, i.e. sets of atoms. We have defined different anti-unification operators and we have studied several desirable characteristics for a common generalization, namely optimal cardinality (lcg), highest τ-value (msg) and variable dataflow optimizations. For each case we have provided detailed worst-case time complexity results and proofs. An interesting case arises when one wants to minimize the number of generalization variables or constrain the generalization relations so as they are built on injective substitutions. In both cases, computing a relevant generalization becomes an NP-complete problem, results that we have formally established. In addition, we have proven that an interesting abstraction – namely k-swap stability which was introduced in earlier work – can be computed in polynomially bounded time, a result that was only conjectured in earlier work.

Our discussion of dataflow optimization in Section 4 essentially corresponds to a reframing of what authors of related work sometimes call the *merging* operation in rule-based anti-unification approaches as in [4]. Indeed, if the "store" manipulated by these approaches contains two anti-unification problems with variables generalizing the same terms, then one can "merge" the two variables to produce their most specific generalization. If the merging is exhaustive, this technique results in a generalization with as few different variables as possible. In this work we isolated dataflow optimization from that specific use case and discussed it as an anti-unification problem in its own right.

While anti-unification of goals in logic programming is not in itself a new subject, to the best of our knowledge our work is the first systematic treatment of the problem in the case where the goals are not sequences but unordered sets. Our work is motivated by the need for a practical (i.e. tractable) generalization algorithm in this context. The current work provides the theoretical basis behind these abstractions, and our concept of k-swap stability is a first attempt that is worth exploring in work on clone detection such as [28].

Other topics for further work include adapting the k-swap stable abstraction from the $\preceq^{\iota}$ relation to dealing with the $\sqsubseteq^{\iota}$ relation. A different yet related topic in need of further research is the question about what anti-unification relation is best suited for what applications. For example, in our own work centered around clone detection in Constraint Logic Programming, anti-unification is seen as a way to measure the distance amongst predicates in order to guide successive syntactic transformations. Which generalization relation is best suited to be applied at a given moment and whether this depends on the underlying constraint context remain open questions that we plan to investigate in the future.

References

1 María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A Modular Order-Sorted Equational Generalization Algorithm. *Information and Computation*, 235:98–136, 2014. Special issue on Functional and (Constraint) Logic Programming. doi:10.1016/j.ic.2014.01.006.

2 Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in haskell functions via anti-unification. *Future Generation Computer Systems*, 79:669–686, 2018. doi:10.1016/j.future.2017.07.024.

3 Alexander Baumgartner and Temur Kutsia. Unranked second-order anti-unification. *Information and Computation*, 255:262–286, 2017. WoLLIC 2014. doi:10.1016/j.ic.2017.01.005.

4 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *Journal of Automated Reasoning*, 58(2):293–310, February 2017. doi:10.1007/s10817-016-9383-3.

5 Peter E. Bulychev, Egor V. Kostylev, and Vladimir A. Zakharov. Anti-unification Algorithms and Their Applications in Program Analysis. In Amir Pnueli, Irina Virbitskaite, and Andrei Voronkov, editors, *Perspectives of Systems Informatics*, pages 413–423, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

6 Jochen Burghardt. E-generalization using grammars. *Artificial Intelligence*, 165(1):1–35, 2005. doi:10.1016/j.artint.2005.01.008.

7 Jochen Burghardt. An improved algorithm for e-generalization. *arXiv*, 2017. arXiv:1709.00744.

8 Dirk G. Cattrysse and Luk N. [Van Wassenhove]. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260–272, 1992. doi:10.1016/0377-2217(92)90077-M.

9 David M. Cerna and Temur Kutsia. Higher-order pattern generalization modulo equational theories. *Mathematical Structures in Computer Science*, 30(6):627–663, 2020. doi:10.1017/S0960129520000110.

10 Philippe Codognet and Daniel Diaz. Boolean Constraint Solving Using CLP(FD). In *International Logic Programming Symposium*, page 15 pages, Vancouver, British Columbia, Canada, 1993.

11 Danny De Schreye, Robert Glück, Jesper Jørgensen, Michael Leuschel, Bern Martens, and Morten Heine Sørensen. Conjunctive partial deduction: foundations, control, algorithms, and experiments. *The Journal of Logic Programming*, 41(2):231–277, 1999. doi:10.1016/S0743-1066(99)00030-8.

12 Melvin Fitting. Fixpoint Semantics for Logic Programming A Survey. *Theoretical Computer Science*, 278(1):25–51, 2002. Mathematical Foundations of Programming Semantics 1996. doi:10.1016/S0304-3975(00)00330-3.

13 J. P. Gallagher. Tutorial on Specialisation of Logic Programs. In *Proceedings of the 1993 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*, PEPM '93, pages 88–98, New York, NY, USA, 1993. ACM. doi:10.1145/154630.154640.

14 Graeme Gange, Jorge A. Navas, Peter Schachte, Harald Sondergaard, and Peter Stuckey. Horn Clauses as an Intermediate Representation for Program Analysis and Transformation. *Theory and Practice of Logic Programming*, 15, July 2015. doi:10.1017/S1471068415000204.

15 Peter Idestam-Almquist. Generalization of Clauses under Implication. *Journal of Artificial Intelligence Research*, November 1995. doi:10.1613/jair.194.

16 Joxan Jaffar, Michael Maher, Kim Marriott, and Peter Stuckey. The Semantics of Constraint Logic Programs. *The Journal of Logic Programming*, 37(1):1–46, 1998. doi:10.1016/S0743-1066(98)10002-X.

17 Laura Ildikó Kovács and Tudor Jebelean. An Algorithm for Automated Generation of Invariants for Loops with Conditionals. In *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), 25-29 September 2005, Timisoara, Romania*, pages 245–249, 2005. doi:10.1109/SYNASC.2005.19.

18 Ulf Krumnack, Angela Schwering, Helmar Gust, and Kai-Uwe Kühnberger. Restricted Higher-Order Anti-Unification for Analogy Making. In Mehmet A. Orgun and John Thornton, editors, *AI 2007: Advances in Artificial Intelligence*, pages 273–282, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

19 Frédéric Mesnard, Étienne Payet, and Wim Vanhoof. Towards a Framework for Algorithm Recognition in Binary Code. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016*, pages 202–213, 2016. doi:10.1145/2967973.2968600.

20 Stephen Muggleton and Luc de Raedt. Inductive Logic Programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629–679, 1994. Special Issue: Ten Years of Logic Programming. doi:10.1016/0743-1066(94)90035-3.

21 Stephen Muggleton and Cao Feng. Efficient Induction of Logic Programs. In *New Generation Computing*. Academic Press, 1990.

22 S. H. Nienhuys-Cheng and R. de Wolf. Least Generalizations and Greatest Specializations of Sets of Clauses. *arXiv e-prints*, page cs/9605102, April 1996. arXiv:cs/9605102.

23 Alberto Pettorossi and Maurizio Proietti. Program Specialization via Algorithmic Unfold/Fold Transformations. *ACM Comput. Surv.*, 30(3es):6, 1998. doi:10.1145/289121.289127.

24 F. Pfenning. Unification and Anti-Unification in the Calculus of Constructions. In *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 74–85, July 1991. doi:10.1109/LICS.1991.151632.

25 Gordon D. Plotkin. A Note on Inductive Generalization. *Machine Intelligence*, 5:153–163, 1970.

26 Reudismam Rolim, Gustavo Soares, Rohit Gheyi, Titus Barik, and Loris D'Antoni. Learning quick fixes from code repositories, 2018. arXiv:1803.03806.

27 Morten H. Sørensen and Robert Glück. An Algorithm of Generalization in Positive Supercompilation. In *Proceedings of ILPS'95, the International Logic Programming Symposium*, pages 465–479. MIT Press, 1995.

28 Wim Vanhoof and Gonzague Yernaux. Generalization-Driven Semantic Clone Detection in CLP. In Maurizio Gabbrielli, editor, *Logic-Based Program Synthesis and Transformation*, pages 228–242, Cham, 2020. Springer International Publishing.

29 Gonzague Yernaux and Wim Vanhoof. Anti-unification in Constraint Logic Programming. *Theory and Practice of Logic Programming*, 19(5-6):773–789, 2019. doi:10.1017/S1471068419000188.