32nd International Symposium on Temporal Representation and Reasoning

TIME 2025, August 27–29, 2025, Queen Mary University of London, UK

Edited by

Thierry Vidal

Przemysław Andrzej Wałęga

Editors

Thierry Vidal
Technological University of Tarbes, France
thierry.vidal@uttop.fr

Przemysław Andrzej Wałęga
Queen Mary University of London, UK
p.walega@qmul.ac.uk

ACM Classification 2012
Computing methodologies → Temporal reasoning; Information systems → Spatial-temporal systems; Theory of computation → Modal and temporal logics; Information systems → Temporal data; Mathematics of computing → Time series analysis; Theory of computation → Timed and hybrid models; Computing methodologies → Discrete-event simulation

ISBN 978-3-95977-401-7

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/978-3-95977-401-7.

Publication date
October, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.TIME.2025.0

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

32nd International Symposium on Temporal Representation and Reasoning (TIME 2025).
Editors: Thierry Vidal and Przemysław Andrzej Wałęga
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Short Papers

Preface

"If you knew Time as well as I do," said the Hatter, "you wouldn't talk about wasting it."
— Lewis Carroll, Alice's Adventures in Wonderland

We are delighted to welcome you to the 32nd International Symposium on Temporal Representation and Reasoning (TIME 2025), hosted at Queen Mary University of London, UK.

TIME has been for more than twenty years the only yearly multidisciplinary international event dedicated to the topic of time in computer science. The purpose of the symposium is to bring together active researchers in different scientific fields involving temporal and spatio-temporal data, information and/or knowledge management. Such a concern arises in a number of different though often related research domains, namely Artificial Intelligence (both symbolic approaches based on explicit Logic or Constraint-based models, and numerical data-based approaches such as Deep Learning and Large Language Models), Databases and Data Mining, or System Specification and Verification. In an effort to broaden the symposium program we have introduced in TIME 2025 three types of submissions:

- **Original long papers** (12 pages), presenting unpublished theoretical (algorithms, models, proofs) or applied (systems, evaluations, applications) work;
- **Survey papers** (12 pages), offering concise overviews of established research areas; and
- **Extended abstracts** (4 pages), initially intended to stimulate discussion and include work-in-progress, project reviews, PhD summaries, or summaries of papers published elsewhere.

This year, we received 27 submissions. Following a rigorous single blind peer-review process, 19 papers were accepted for presentation: 11 as original 12-page long papers, which were allocated a 30 minutes talk, and 8 as 4-page **short papers** (either as submitted extended abstracts, or initial original long papers which received a more balanced assessment), which were allocated a 20 minutes talk. Unfortunately this year submitted survey papers did not meet our quality requirements, next editions will have to decide if that type of submission will be maintained.

The program also features two invited talks:

- **Michael Zakharyaschev** (Birkbeck, University of London, UK): *Separation and interpolation problems related to linear temporal logic LTL*
- **Michael Wooldridge** (University of Oxford, UK): *From Synthesis to Rational Verification*

Moreover, which is another novelty in TIME 2025, the program includes an invited tutorial:

- **Nicola Gigante** (Free University of Bozen-Bolzano, Italy): *An Introduction to First-Order Linear Temporal Logic*

The conference program also includes social events themed around the notion of time. We will host a welcome reception in the heart of London, near Big Ben – an iconic symbol of both the city and the passage of time. In addition, we organise an excursion to the Royal Observatory in Greenwich, home of the Prime Meridian and the historical centre of timekeeping. There, participants will have the opportunity to explore the origins of modern time measurement and stand on the zero longitude line.

32nd International Symposium on Temporal Representation and Reasoning (TIME 2025).
Editors: Thierry Vidal and Przemysław Andrzej Wałęga
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We are deeply grateful to the members of the programme committee for their diligence and commitment to maintaining the scientific quality of the symposium, as well as to all authors for the care taken in their submissions and final versions of their articles, and in their oral presentations, including our invited speakers. In view of the high quality of these contributions, we have decided this year to award two prizes:

- The recipient of the **best paper award** is *On the Complexity of the Realisability Problem for Visit Events in Trajectory Sample Databases* which authors are **Arthur Jansen** and **Bart Kuijpers**.
- The recipient of the **best reviewer award** is **Emanuel Sallinger** from the Technical University of Wien.

We also thank the local organizing team for their dedication in making this event a success, and express our appreciation to LIPIcs for publishing the proceedings and supporting open-access dissemination of scientific knowledge.

We hope that TIME 2025 provides a stimulating and collaborative environment for all participants, and contributes meaningfully to the advancement of temporal representation and reasoning.

TIME 2025 Program Committee Chairs
Thierry Vidal, Technical University of Tarbes, France
Przemysław Wałęga, Queen Mary University of London, UK

TIME Steering Committee

32nd International Symposium on Temporal Representation and Reasoning (TIME 2025).
Editors: Thierry Vidal and Przemysław Andrzej Wałęga

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

Program Committee Members

32nd International Symposium on Temporal Representation and Reasoning (TIME 2025).
Editors: Thierry Vidal and Przemysław Andrzej Wałęga
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Yakoub Salhi
Artois University
France
salhi@cril.fr

Emanuel Sallinger
Technical University, Wien
Austria
emanuel.sallinger@tuwien.ac.at

Guido Sciavicco
University of Ferrara
Italy
scvgdu@unife.it

Michael Sioutis
University of Montpellier
France
michael.sioutis@lirmm.fr

Kathleen Stewart
University of Maryland
US
stewartk@umd.edu

Ajdin Sumic
ONERA, Toulouse
France
aidin.sumic@onera.fr

Paolo Terenziani
University of Piemonte Orientale, Alessandria
Italy
paolo.terenziani@unipmn.it

Matteo Zavatteri
University of Padua
Italy
matteo.zavatteri@unipd.it

Local Organizing Committee

Fredrik Dahlqvist
Niki Omidvari
Yongxin Yang
Queen Mary University of London
United Kingdom

Antonio Di Stasio
City, University of London
United Kingdom

David Tena Cucala
RHUL, University of London
United Kingdom

Raymond Hu
Marc Roth

32nd International Symposium on Temporal Representation and Reasoning (TIME 2025).
Editors: Thierry Vidal and Przemysław Andrzej Wałęga
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Rashmie Abeysinghe (13)
The University of Texas Health Science Center at Houston, TX, USA

Eric Alsmann (3)
Theoretical Computer Science / Formal Methods, University of Kassel, Germany

Alexander Artikis (18)
University of Piraeus, Greece;
NCSR "Demokritos", Athens, Greece

Mathieu Barthet (20)
Aix-Marseille Univ CNRS PRISM, France;
Centre for Digital Music, Queen Mary University of London, United Kingdom

Pietro Bellodi (4)
Department of Mathematics and Computer Science, University of Ferrara, Italy

Luigi Bellomarini (15)
Bank of Italy, Rome, Italy

Nassim Belmecheri (14)
Simula Research Laboratory, Oslo, Norway

Jiang Bian (13)
Indiana University Bloomington, Bloomington, IN, USA

Livia Blasi (15)
TU Wien, Vienna, Austria;
Bank of Italy, Rome, Italy

Florian Bruse (5)
TUM School of Computation, Information and Technology, Technical University of Munich, Munich, Germany

Pietro Casavecchia (4)
Department of Mathematics and Computer Science, University of Ferrara, Italy

Hongyu Chen (13)
University of Florida, Gainesville, FL, USA

Mark Chevallier (6)
School of Engineering, University of Edinburgh, UK

Licong Cui (13)
The University of Texas Health Science Center at Houston, TX, USA

Fabio Aurelio D'Asaro (21)
Dip. di Studi Umanistici,
Università del Salento, Lecce, Italy

Giuseppe De Giacomo (7)
University of Oxford, UK

Antonio Di Stasio (7)
City St George's, University of London, UK

Luke Dickens (21)
Dept. of Information Studies,
University College London, UK

Riccardo Dondi (8)
Università degli Studi di Bergamo, Italy

Curtis E. Dyreson (9)
Department of Computer Science,
Utah State University, Logan, UT, USA

Jacques D. Fleuriot (6)
School of Informatics,
University of Edinburgh, UK

Luca Geatti (10)
University of Udine, Italy

Nicola Gigante (2)
Free University of Bozen-Bolzano, Italy

Xing He (13)
Indiana University Bloomington,
Bloomington, IN, USA

Yan Huang (13)
The University of Texas Health Science Center at Houston, TX, USA

Luke Hunsberger (11)
Vassar College, Poughkeepsie, NY, USA

Arthur Jansen (12, 16, 17)
Hasselt University, Databases and Theoretical Computer Science Group and Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Andreas Kouvaras (18)
University of Piraeus, Greece

Bart Kuijpers (12, 16, 17)
Hasselt University, Databases and Theoretical Computer Science Group and Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Martin Lange (3)
Theoretical Computer Science / Formal Methods, University of Kassel, Germany

32nd International Symposium on Temporal Representation and Reasoning (TIME 2025).
Editors: Thierry Vidal and Przemysław Andrzej Wałęga
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Stefan Lattner (20)
Sony CSL, Paris, France

Pengze Li (13)
Mayo Clinic in Florida, Jacksonville, FL, USA

Xiaojin Li (13)
The University of Texas Health Science Center at Houston, TX, USA

Periklis Mantenoglou (18)
Örebro University, Sweden

Rares-Ioan Mateiu (8)
Department of Computer Science, University of Bucharest, Romania

Gregor Meehan (20)
Centre for Digital Music, Queen Mary University of London, United Kingdom

Mauro Milella (19)
Department of Mathematics and Computer Science, University of Ferrara, Italy

Markus Nissl (15)
TU Wien, Vienna, Austria

Giovanni Pagliarini (19)
Department of Mathematics and Computer Science, University of Ferrara, Italy

Alberto Paparella (4)
Department of Mathematics and Computer Science, University of Ferrara, Italy

Gianmarco Parretti (7)
La Sapienza University of Rome, Italy

Johan Pauwels (20)
Centre for Digital Music, Queen Mary University of London, United Kingdom

Stefano Pessotto (10)
University of Udine, Italy

Alexandru Popa (8)
Department of Computer Science, University of Bucharest, Romania

Roberto Posenato (11)
University of Verona, Italy

Emanuel Sallinger (15)
TU Wien, Vienna, Austria;
University of Oxford, Oxford, UK

Bishal Sarkar (9)
Department of Computer Science, Utah State University, Logan, UT, USA

Richard Schmoetten (6)
School of Informatics, University of Edinburgh, UK

Guido Sciavicco (4, 19)
Department of Mathematics and Computer Science, University of Ferrara, Italy

Filip Smola (6)
School of Informatics, University of Edinburgh, UK

Ionel Eduard Stan (4, 19)
Department of Informatics, Systems, and Communications, University of Milano-Bicocca, Italy

Zenan Sun (13)
The University of Texas Health Science Center at Houston, TX, USA

Cui Tao (13)
Mayo Clinic in Florida, Jacksonville, FL, USA

Shiqiang Tao (13)
The University of Texas Health Science Center at Houston, TX, USA

Stefano Tonetta (10)
Fondazione Bruno Kessler, Italy

Alexander Williams (20)
Centre for Digital Music, Queen Mary University of London, United Kingdom

Lyris Xu (21)
Dept. of Information Studies, University College London, UK

Michael Zakharyaschev (1)
School of Computing and Mathematical Sciences, Birkbeck, University of London, UK

Guo-Qiang Zhang (13)
The University of Texas Health Science Center at Houston, TX, USA

Interpolation and Separation Problems for Linear Temporal Logics

Michael Zakharyaschev
School of Computing and Mathematical Sciences, Birkbeck, University of London, UK

Abstract

The talk gives a survey of recent results on two types of problems for linear temporal logics: one is related to the existence of a Craig interpolant for a given implication in a temporal logic, the other to the existence of a temporal query separating two given sets of temporal data instances.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Linear temporal logic, Craig interpolation, query-by-example

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.1

Category Invited Talk

1 Extended Abstract

This talk discusses two types of problems that have recently been posed and investigated in the context of linear time temporal logics.

The first problem is related to Craig interpolation. Recall that a logic L has the *Craig interpolation property* (CIP, for short) if, for any two formulas φ and ψ in the language of L, whenever $\varphi \to \psi$ is valid in L, then there exists a formula χ, constructed from the common non-logical symbols of φ and ψ, such that both $\varphi \to \chi$ and $\chi \to \psi$ are valid in L. Such a formula χ is called an *interpolant of φ and ψ* in L. Classical and intuitionistic propositional and first-order logics, standard modal and description logics as well as numerous other logics enjoy the CIP. The vast majority of temporal logics do not, including all logics with the operators $\Diamond$ (some time in the future) and $\Box$ (always in the future) interpreted over any class of connected Kripke frames of unbounded depth, their Priorean extensions with the past-time operators, and full linear temporal logic LTL over any interesting time line.

A different, non-uniform approach to Craig interpolation in logics L without the CIP was suggested in [4] by posing the following *interpolant existence problem* (IEP): given any formulas φ and ψ in the language of L, decide whether there exists an interpolant χ for φ and ψ in L. For L with the CIP, interpolant existence reduces to validity; for L without the CIP, the IEP is typically harder as far as computational complexity is concerned.

The first part of the talk gives a survey of the recent results on the IEP in temporal logics obtained in [5, 6]. In particular, it discusses how a semantic criterion of interpolant existence in terms of descriptive frames can be used to design a coNP-algorithm that is capable of deciding the IEP for any given finitely axiomatisable temporal logic with the operators $\Diamond$ and $\Box$. In this case, the IEP is as complex as the decision problem for the logic. For LTL over finite strict linear orders, the talk discusses how the IEP can be reduced to the following separation problem: given two regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$, decide whether there is a first-order definable (= star-free) language $\mathcal{L}$ separating $\mathcal{L}_1$ and $\mathcal{L}_2$ in the sense that $\mathcal{L}_1 \subseteq \mathcal{L}$ and $\mathcal{L}_2 \cap \mathcal{L} = \emptyset$. A classical result of [7] shows that deciding first-order separability of two regular languages is in 2ExpTime in the size of the DFAs specifying the given languages. Thus, the IEP for LTL is decidable in 4ExpTime, being PSpace-hard. The same bounds hold for LTL over $(\mathbb{N}, <)$. In both cases, the exact complexity remains open.

The second part of the talk is concerned with the scenario where temporal models over finite strict linear orders represent timestamped (qualitative) data. We discuss the reverse engineering (or query-by-example) setting that deals with pairs (E^+, E^-) of finite sets of "positive" and "negative" data examples. An LTL-formula (query) $\boldsymbol{q}$ *separates* (E^+, E^-) if $\mathfrak{M}, 0 \models \boldsymbol{q}$ for all $\mathfrak{M} \in E^+$, and $\mathfrak{M}, 0 \not\models \boldsymbol{q}$ for all $\mathfrak{M} \in E^-$. Based on the results from [1, 2, 3], we give a survey of recent developments in the study of the following problems:

Given any pair (E^+, E^-) of positive and negative examples and a class $\mathcal{Q}$ of LTL-queries,

1. How hard is it to decide whether (E^+, E^-) is separable by a query in $\mathcal{Q}$?
2. How hard is it to compute a $\mathcal{Q}$-separator of (E^+, E^-), which is (i) shortest/longest or (ii) most specific (logically strongest)/most general (weakest) $\mathcal{Q}$-separator?
3. How hard is it to decide whether there is a unique most specific/general $\mathcal{Q}$-separator?
4. Given a $\mathcal{Q}$-separator $\boldsymbol{q}$ of (E^+, E^-), how hard is it to decide whether $\boldsymbol{q}$ is (i) shortest/longest, (ii) most general/specific, (iii) unique, etc.?
5. Given any $\boldsymbol{q} \in \mathcal{Q}$, how hard is it to decide whether there is (E^+, E^-) that *uniquely characterises* $\boldsymbol{q}$ in $\mathcal{Q}$ in the sense that $\boldsymbol{q}' \in \mathcal{Q}$ separates (E^+, E^-) iff $\boldsymbol{q}' \equiv \boldsymbol{q}$?
6. Does there exist a *polynomial-size* unique characterisation of a given $\boldsymbol{q} \in \mathcal{Q}$?

Interesting and practically relevant classes $\mathcal{Q}$ consist of conjunctive LTL formulas. We also outline generalisations of the problems above to combinations of LTL and description logics.

References

1 Marie Fortin, Boris Konev, Vladislav Ryzhikov, Yury Savateev, Frank Wolter, and Michael Zakharyaschev. Unique characterisability and learnability of temporal instance queries. In Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer, editors, *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel, July 31 - August 5, 2022*, 2022. URL: https://proceedings.kr.org/2022/17/.

2 Jean Christoph Jung, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Temporalising unique characterisability and learnability of ontology-mediated queries (extended abstract). In Oliver Kutz, Carsten Lutz, and Ana Ozaki, editors, *Proceedings of the 36th International Workshop on Description Logics (DL 2023) co-located with the 20th International Conference on Principles of Knowledge Representation and Reasoning and the 21st International Workshop on Non-Monotonic Reasoning (KR 2023 and NMR 2023)., Rhodes, Greece, September 2-4, 2023*, volume 3515 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3515/abstract-13.pdf.

3 Jean Christoph Jung, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Extremal separation problems for temporal instance queries. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 3448–3456. ijcai.org, 2024. URL: https://www.ijcai.org/proceedings/2024/382.

4 Jean Christoph Jung and Frank Wolter. Living without beth and craig: Definitions and interpolants in the guarded and two-variable fragments. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470585.

5 Agi Kurucz, Frank Wolter, and Michael Zakharyaschev. A non-uniform view of Craig interpolation in modal logics with linear frames. *CoRR*, abs/2312.05929, 2023. doi:10.48550/arXiv.2312.05929.

6 Agi Kurucz, Frank Wolter, and Michael Zakharyaschev. From interpolating formulas to separating languages and back again, 2026. To appear, preprints accessible from https://cibd.bitbucket.io/taci/.

7 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. *Log. Methods Comput. Sci.*, 12(1), 2016. doi:10.2168/LMCS-12(1:5)2016.

An Introduction to First-Order Linear Temporal Logic

Nicola Gigante
Free University of Bozen-Bolzano, Italy

Abstract

Linear temporal logic (LTL), most commonly defined as a propositional modal logic, is the de-facto standard language for specifying temporal properties of systems in formal verification, artificial intelligence, and other fields. First-order linear temporal logic (FOLTL) lifts LTL to the setting of first-order logic, obtaining a remarkably flexible and expressive formalism. First-order modal and temporal logics have a long history, but recent years have seen a rise of interest in (well-behaved fragments of) FOLTL for the specification of complex infinite-state systems. This tutorial is a gentle introduction to the field of first-order temporal logics, starting from classic results and exploring recent directions.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Temporal logic, first-order logic, knowledge-representation, infinite-state systems

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.2

Category Invited Talk

1 Introduction

Linear Temporal Logic (LTL) [16] is one of the most common formalisms to express temporal properties of systems in many fields including *formal verification* and *artificial intelligence*. In its classic form, LTL is a propositional modal logic interpreted over *infinite* linear orders or *words*, although recently interest has risen in the artificial intelligence field for LTL_f [3], *i.e.* LTL interpreted over *finite* words.

The success of LTL stems from its intuitive syntax and semantics and the existence of many efficient techniques for *reasoning* about the logic. Indeed, although *satisfiability* of LTL formulas is PSPACE-complete [21], many efficient techniques are known [9, 12, 22], and the same can be said for *model checking* [15].

However, in many scenarios, the *propositional* nature of LTL poses some limits to its applicability. Consider the following classic example:

$$\mathsf{G}(req \rightarrow \mathsf{F}grant)$$

The above formula states that at any given moment (G), if a *request* is received, then *eventually* (F) an answer is *granted*. This kind of specification is quite common, but it is rarely sufficient to express it in the above way. That is because the above formula makes no connection between the answer that is granted and the request that is being answered. For example, the formula is also satisfied by a single answer after many requests.

What one would really like to express is that *each* request gets *its own* answer. One may wonder what the *identity* of requests consist in. Let us suppose requests have unique identifiers that last for all the execution of the system, and suppose that, instead of propositions, we can use the $req(r)$ predicate to tell that the request r has been requested, and $grant(r)$ to tell that r has been answered. Then, we can write the following:

$$\forall r \,.\, \mathsf{G}(req(r) \rightarrow \mathsf{F}grant(r))$$

The above specification is a *sentence* in *first-order linear temporal logic* (FOLTL). FOLTL combines the usual *temporal operators* from LTL with classic *first-order logic*, obtaining a remarkably expressive and sophisticated language. In what follows we introduce the semantics of this language (Section 2), its major computational trade-offs (Section 3), and we describe some current trends (Section 4).

2 Semantics

In first-order logic, once the *signature* Σ is fixed (*i.e.* the set of constant, predicate and function symbols used), sentences (*i.e.* formulas with no free variables) are interpreted over Σ-structures that interpret the signature's symbols. In FOLTL, sentences are interpreted over *sequences* of Σ-structures, representing the evolution over time of the symbols' interpretation. Classically, FOLTL has been studied over many different kind of sequences or *linear orders* of Σ-structures: finite or infinite, discrete or *dense*, *etc.*, with most classic results holding independently of the nature of the underlying linear order [5]. Here, for simplicity, let us assume *discrete* linear orders, either finite or infinite.

When formally defining the semantics of the *satisfaction* of FOLTL formulas, one immediately encounters a non-trivial question about the meaning of *existence* of the objects of first-order quantification. To see what this means, suppose we are modeling the human resources of a company, and we care about the feelings of our employees. We may write a sentence like the following.

$$\psi := \forall x.\mathsf{G}happy(x) \rightarrow \mathsf{G}\forall x.happy(x)$$

The above sentence is an instance of a scheme called the *Barcan formula* [14]. In this case, it is saying that, in the company, if everyone is happy every day, then every day everyone is happy. One may consider ψ to be obviously a *valid* FOLTL sentence or not, depending on two different views:

1. the sentence is valid because the two mentions of *"everyone"* in the above phrase refer to the same set of people, *i.e.* the domain of the two universal quantifiers is the same; or
2. the sentence is not valid because who is *"everyone"* today may include or not include people that will be absent or present tomorrow, *i.e.* the domain of the two universal quantifiers may differ.

Depending on our choice of point of view, we can identify a different *semantics* for FOLTL:

1. *eternalist* or *constant-domain* semantics: every and all objects in the quantification domain always exist at any point in time, as the domain *does not change*;
2. *presentist* or *varying-domain* semantics: objects in the domain pop up into existence at some time and cease to exist at some later time, as the domain *varies over time*.

In the eternalist semantics, the Barcan sentence is valid and we can swap the universal quantifier with the *always* temporal operator (or, equivalently, the existential quantifier with the *eventually* operator). In the presentist semantics, we cannot do that. The choice of semantics affects the semantics of purely first-order sentences as well. Consider for example the following sentence:

$$\forall x.happy(x) \rightarrow happy(\mathsf{Nik})$$

This is an instance of the *universal instantiation* axiom which is *valid* in classical first-order logic for any formula in the place of $happy(x)$. However, in the varying-domain semantics the sentence is not valid anymore, because the constant Nik may refer to somebody that is not *currently* in the domain (*e.g.* because he still has to be hired) and therefore *may* be not happy. An axiomatization of FOLTL in both semantics can be found in McArthur's book [14].

Note that, in modeling terms, the *constant-domains* semantics is the most general: one can simulate the varying-domain semantics in a constant-domain sentence by introducing an *existence predicate* $exists(x)$ guarding all occurrences of quantifiers.

3 Computational trade-offs

Depending on the signature Σ and the Σ-*theory* we consider, FOLTL can be extremely expressive. Of course, this expressiveness is payed in terms of *tractability*, as both *satisfiability* and *validity* are *undecidable* and *not even semi-decidable* [5]. This can be seen easily, for example, by reducing the *recurrent tiling* problem [23] to FOLTL satisfiability.

Unfortunately, research has shown that *decidable* fragments of FOLTL are rare and far apart [5]. A classic example of decidable fragment of FOLTL is the *monodic* fragment. A sentence is *monodic* if any *temporal* subformula has at most *one free variable*. For example:

$$\forall x[p(x) \rightarrow Gp(x)]$$

is a monodic sentence, as is the *Barcan* sentence discussed above. Instead:

$$\forall xy[r(x, y) \rightarrow Gr(x, y)]$$

is *not* monodic. The *monodic* fragment is made of *relational* monodic sentences *with no* equality symbol, and can be proved to be decidable over any kind of linear order through a reduction to Büchi's decidability result for monadic second-order logic [5] or, for discrete linear orders, via *first-order automata* [7].

It is clear that the major restriction of the monodic fragment is the inability of transferring *relational* information across time. As this is a major limitation for the modeling of many scenarios, one may wonder whether more expressiveness can be recovered by accepting *semi-decidability*. After all, many decades of research in the automated reasoning community have proven that semi-decidable problems *can* be addressed in practice is suitably effective semi-decision procedures are found. An example is that of *constrained Horn clauses*, a semi-decidable fragment of first-order logic effectively solved in practice by *property-directed reachability* techniques [11].

In this vein, one may prove (*e.g.*, again via first-order automata [7]) that if one starts from a combination of *decidable* first-order logic fragment and theory (*e.g.* the *two-variable fragment* [10] or many decidable theories employed in *satisfiability modulo theories* (SMT) [2]), then FOLTL over *finite words* is *semi-decidable*.

4 Recent trends

Recent trends go in the mentioned direction of accepting computational trade-offs in exchange of more expressive power, unlocking the usage of (well-behaved fragments of) FOLTL in the specification of infinite-state systems.

An example of work in this direction is $\mathsf{LTL_f}$ *modulo theories* ($\mathsf{LTL_f^{MT}}$) [6, 8], a recently-introduced fragment of FOLTL interpreted over *finite* words that, although semi-decidable, has a decision procedure effectively implementable in terms of modern SMT solvers.

$\mathsf{LTL_f^{MT}}$ poses semantic and syntactic restrictions to FOLTL. Semantically, it is interpreted, in the *constant-domain* semantics, over finite linear orders where the interpretation of predicates and function symbols is *rigid*, *i.e.* arbitrary but fixed in time, and only the

interpretation of *constants* is allowed to change. Syntactically, it forbids the alternation of quantifiers and temporal operators but allows the usage of a *lookahead* operator that, applied to a constant, designates the value of the constant at the next time step. For example:

$$(\rhd\, a = a + 1)\ \mathsf{U}\ (a = 42)$$

is a $\mathsf{LTL}_\mathsf{f}^\mathsf{MT}$ sentence saying that the constant a increments by one at each time step until it reaches the value 42. One may rewrite such a sentence in pure FOLTL as follows[1]:

$$(\exists x.[\mathsf{X}(x = a) \wedge x = a + 1])\ \mathsf{U}\ (a = 42)$$

The $\mathsf{LTL}_\mathsf{f}^\mathsf{MT}$ logic has been defined over finite words because its semi-decidability (under the conditions mentioned in the previous section) cannot be proven in general if infinite words are involved. However, some *decidability* conditions have been identified [8] which hold for infinite words as well (*i.e.* LTL^MT). These results complement the classic ones on the monodic fragment: $\mathsf{LTL}_\mathsf{f}^\mathsf{MT}$ *can* be translated into monodic FOLTL sentences, but the classic decidability results mentioned above hold for *rigid constants* and *non-rigid predicates*, which is exactly the opposite semantic setting than $\mathsf{LTL}_\mathsf{f}^\mathsf{MT}$, in addition to the fact that $\mathsf{LTL}_\mathsf{f}^\mathsf{MT}$ allows the equality symbol.

The structure of the logic has been designed in order to have its satisfiability problem being easily reduced into a sequence of SMT calls that can be solved by standard solvers, obtained by an SMT encoding of a suitable extension of Reynolds' tree-shaped tableau for LTL [9, 18]. Performance are promising in practice, although more work is still needed to exploit the full potential of the approach.

Recently, interest has sparked about a task that goes beyond satisfiability and validity, *i.e.* *reactive synthesis*. This is the task of synthesizing a *controller* that can ensure the satisfaction of a temporal formula independently from the actions of an external antagonistic environment. Reactive synthesis for propositional LTL and LTL_f is already a hard problem (2EXPTIME-complete [4, 17]), and is of course undecidable for FOLTL and for $\mathsf{LTL}_\mathsf{f}^\mathsf{MT}$ as well.

However, progress has been made on addressing the problem in practice. In particular, *equirealizable* Boolean abstractions have been found for LTL^MT *without lookaheads* [20] which can be given to propositional LTL synthesizers, whose produced strategies can be mapped back to a strategy for the original LTL^MT sentence. Then, the approach has been extended to LTL^MT with lookaheads by a counterexample-guided abstraction refinement procedure [19], although unlocking full potential of this technique requires manual intervention in the loop.

5 Conclusions

The field of first-order temporal logics intersects with first-order modal logics, knowledge representation, temporal *description logics* [1, 13], satisfiability modulo theories, and many other corners of computer science, in a fascinating and intricate web of connections.

Because of its discouraging computational behavior, FOLTL has been studied during the decades in mostly theoretical terms and with a focus on modeling and knowledge representation rather than reasoning. Nevertheless, recent work has highlighted the possibility of dealing in practice with reasoning tasks over expressive fragments of FOLTL, including *reactive synthesis*. Progress in this field is encouraging and this short abstract also wants to be a call to action for the community to invest resources in this promising direction.

[1] The semantics given in [6] needs to be slightly tweaked for this translation to work in general.

References

1 Alessandro Artale and Enrico Franconi. Temporal description logics. In Michael Fisher, Dov M. Gabbay, and Lluís Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 375–388. Elsevier, 2005. doi:10.1016/S1574-6526(05)80014-8.

2 Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-825.

3 Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of 23rd IJCAI*, pages 854–860, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

4 Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1558–1564. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/223.

5 D.M. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. Fragments of first-order temporal logics. In *Many-Dimensional Modal Logics*, volume 148 of *Studies in Logic and the Foundations of Mathematics*, pages 465–545. Elsevier, 2003. doi:10.1016/S0049-237X(03)80012-5.

6 Luca Geatti, Alessandro Gianola, and Nicola Gigante. Linear temporal logic modulo theories over finite traces. In *Proceedings of the 31st IJCAI*, pages 2641–2647, 2022. doi:10.24963/ijcai.2022/366.

7 Luca Geatti, Alessandro Gianola, and Nicola Gigante. First-order automata. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 14940–14948. AAAI Press, 2025. doi:10.1609/AAAI.V39I14.33638.

8 Luca Geatti, Alessandro Gianola, Nicola Gigante, and Sarah Winkler. Decidable fragments of $\mathrm{ltl}_{\mathrm{f}}$ modulo theories. In *Proceedings of the 26th ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 811–818. IOS Press, 2023. doi:10.3233/FAIA230348.

9 Luca Geatti, Nicola Gigante, Angelo Montanari, and Gabriele Venturato. SAT meets tableaux for linear temporal logic satisfiability. *J. Autom. Reason.*, 68(2):6, 2024. doi:10.1007/S10817-023-09691-1.

10 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bull. Symb. Log.*, 3(1):53–69, 1997. doi:10.2307/421196.

11 Arie Gurfinkel and Nikolaj S. Bjørner. The science, art, and magic of constrained horn clauses. In *Proceedings of the 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 6–10. IEEE, 2019. doi:10.1109/SYNASC49474.2019.00010.

12 Jianwen Li, Yinbo Yao, Geguang Pu, Lijun Zhang, and Jifeng He. Aalta: an LTL satisfiability checker over infinite/finite traces. In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey, editors, *Proc. of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 731–734. ACM, 2014. doi:10.1145/2635868.2661669.

13 Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporal description logics: A survey. In Stéphane Demri and Christian S. Jensen, editors, *15th International Symposium on Temporal Representation and Reasoning, TIME 2008, Université du Québec à Montréal, Canada, 16-18 June 2008*, pages 3–14. IEEE Computer Society, 2008. doi:10.1109/TIME.2008.14.

14 Robert P. McArthur. *Tense Logic*. Springer, 1976. doi:10.1007/978-94-017-3219-2.

15 Kenneth L. McMillan. Interpolation and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 421–446. Springer, 2018. doi:10.1007/978-3-319-10575-8_14.

16 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

17 Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989. doi:10.1007/BFB0035790.

18 Mark Reynolds. A New Rule for LTL Tableaux. In *Proc. of the 7th International Symposium on Games, Automata, Logics and Formal Verification*, volume 226 of *EPTCS*, pages 287–301, 2016. doi:10.4204/EPTCS.226.20.

19 Andoni Rodriguez, Felipe Gorostiaga, and Cesar Sanchez. Counter example guided reactive synthesis for ltl modulo theories. In *Proceedings of the 37th International Conference on Computer Aided Verification (CAV 2025)*, page to appear, 2025.

20 Andoni Rodríguez and César Sánchez. Boolean abstractions for realizability modulo theories. In *Proceedings of the 35th CAV*, volume 13966 of *Lecture Notes in Computer Science*, pages 305–328. Springer, 2023. doi:10.1007/978-3-031-37709-9_15.

21 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.

22 M. Suda and C. Weidenbach. A PLTL-Prover Based on Labelled Superposition with Partial Model Guidance. In *Proc. of the 6th International Joint Conference on Automated Reasoning*, volume 7364 of *LNCS*, pages 537–543. Springer, 2012. doi:10.1007/978-3-642-31365-3_42.

23 Peter van Emde Boas et al. The convenience of tilings. *Lecture Notes in Pure and Applied Mathematics*, pages 331–363, 1997.

Metric Linear-Time Temporal Logic with Strict First-Time Semantics

Eric Alsmann
Theoretical Computer Science / Formal Methods, University of Kassel, Germany

Martin Lange
Theoretical Computer Science / Formal Methods, University of Kassel, Germany

Abstract

We introduce strict first-time semantics for the Until operator from linear-time temporal logic which makes assertions not just about some future moment but about the first time in the future that its argument should hold. We investigate Metric Linear-Time Temporal Logic under this interpretation in terms of expressive power, relative succinctness and computational complexity. While the expressiveness does not exceed that of pure LTL, there are properties definable in this logic which can only be expressed in LTL with exponentially larger formulas. Yet, we show that the complexity of the satisfiability problem remains PSPACE-complete which is in contrast to the EXPSPACE-completeness of Metric LTL. The motivation for this logic originates in a study of the expressive power of State Space Models, a recently proposed alternative to the popular transformer architectures in machine learning.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Automata over infinite objects

Keywords and phrases linear-time temporal logic, metric temporal logic, computational complexity, Streett automata

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.3

1 Introduction

The linear-time temporal logic LTL is a well-known formalism for specifying properties of runs of reactive systems [14]. Its model-theoretic and computational properties are well researched and understood, for instance its satisfiability problem being PSPACE-complete [17] and its expressiveness coinciding with that of First-Order Logic [6], resp. star-free regular expressions over ω-words or over finite words [7].

The relative weakness in expressive power has led to the study of several extensions of LTL, some of which genuinely extend its expressiveness, typically to that of full (ω-)regularity, for instance the Linear-Time μ-Calculus [3, 19], the industry standard PSL [5], Quantified LTL [16], etc. Others extend LTL only pragmatically by providing further constructs without extending the expressive power altogether, but providing means to express certain properties more easily.

One such variant is the Metric Linear-Time Temporal Logic MTL [1]. The term "metric temporal logic" does not uniquely identify one particular formalism, not even within the linear-time framework. It describes temporal logics in which the temporal operators are extended so that they do not simply make assertions about the future (or past) moments in a linear sequence of events, but additionally constrain their distance to the current moment. For example $p\,\mathtt{U}_{[3,5]}\,q$ does not only demand that q holds at some point in the future with p being continuously true up to that point, but this future point also has to occur within three to five time units from now on. The exact semantics then depends on the underlying models etc. Metric temporal logic has been developed for reasoning about real-time systems [9], but

it can also be used for systems evolving in discrete steps. Here we use MTL only as a logic over discrete time, resp. (untimed) ω-words. Consequently, the intervals adorning the metric operators are always interpreted over the natural numbers.

It is not hard to see that MTL does not exceed LTL in expressiveness: temporal operators with metric constraints can easily be unravelled. The property above can be expressed as "*p holds now and in the next two steps; afterwards we have q or p followed by q or p followed by another p, followed by q*" using only the next-time operator (in this case). A general translation from MTL to LTL is an easy exercise, and it incurs a blow-up that is polynomial in the value of the involved interval bounds. Hence, when such bounds are encoded unarily, MTL satisfiability is also PSpace-complete. The more reasonable assumption of binary encodings then yields an exponential translation into LTL and therefore an ExpSpace upper bound on its satisfiability problem. This is tight, satisfiability for MTL with binarily encoded interval bounds is in fact ExpSpace-complete [10].

The lower bound uses a standard reduction from the word problem for exponentially space bounded Turing Machines (TM). The key to the proof is then MTL's ability to express "something holds after 2^n steps" which, using binary encodings, can be done with a formula of size polynomial in n. This is used to express, for instance, that a symbol b is seen at that distance, i.e. at the same cell on the tape in the next configuration, formalising a potential writing operation of the TM. Clearly, the tape may contain more occurrences of that symbol, so the one seen at distance 2^n is usually not the first one to be seen in the future.

This paper is motivated by the study of the expressive power of a recent machine learning model, so-called State-Space Models (SSM) [13]. They have been proposed as an alternative to the well-known transformer architectures underlying prominent Large Language Model tools, promising more efficient evaluation of long input strings. Here we do not go further into the details of SSMs; they merely serve as a motivation for studying a particular variant of MTL which is geared towards the expressive capabilities of SSMs. To motivate this variant, we recall the well-known unfolding principles for temporal operators.

▶ **Example 1.** Suppose that, in the context of some decision procedure, we were to establish the truth of the LTL formula $\mathtt{G}(p\,\mathtt{U}\,q)$ at position i of some ω-sequence of events. By unfolding the $\mathtt{G}$-operator, this typically amounts to establishing truth of both (i) $p\,\mathtt{U}\,q$ and (ii) $\mathtt{X}\,\mathtt{G}(p\,\mathtt{U}\,q)$ at moment i. To solve (i) we can either prove q or defer it to a later point by unfolding the Until-formula and proving p as well as (iii) $\mathtt{X}(p\,\mathtt{U}\,q)$ at moment i. Now, (ii) and (iii) both start with a Next-operator, so they imply proving $\mathtt{G}(p\,\mathtt{U}\,q)$ and (iii') $p\,\mathtt{U}\,q$ at moment $i+1$. The former can be handled as done at moment i; in particular it means proving $p\,\mathtt{U}\,q$ at moment $i+1$ which is already the task identified as (iii'). In a sense, there are two reasons for the need to prove $p\,\mathtt{U}\,q$ at moment $i+1$, and they just collapse to a single task.

Now suppose that the original formula was one of MTL, namely $\mathtt{G}(p\,\mathtt{U}_{[3,5]}\,q)$. Here the reasoning can be done in the same way, but when unfolding the metric Until we obtain an index shift: $p\,\mathtt{U}_{[3,5]}\,q$ is established at moment i when $\mathtt{X}(p\,\mathtt{U}_{[2,4]}\,q)$ is established there. Hence, when carrying out this kind of reasoning we obtain two tasks regarding moment $i+1$ that do not collapse into one, namely to prove $p\,\mathtt{U}_{[2,4]}\,q$ as well as $p\,\mathtt{U}_{[3,5]}\,q$ there.

This can also be seen as an indication for MTL's higher complexity: truth of a formula at moment i does not just depend on the truth of its subformulas at moments i and $i+1$, but also on variants of the subformulas obtained through index shifts.

The mechanisms in SSMs naturally suggest a comparison of their expressiveness with formulas of *metric* linear-time temporal logic (on finite words), since SSMs can easily track the distance between occurrences of events in a word, they seem to be unable to cope with the blow-up in formulas that need to be tracked through unfolding.

In this paper we initiate the study of a variant of MTL– called *Metric Temporal Logic with Strict First-Time Semantics* here, MTL^1– which is defined because of a certain similarity to the mechanics in SSMs. It features the Metric Until operator $\varphi \,\mathtt{U}^1_{[x,y]}\, \psi$ which does not just demand that some future moment at distance between x and y satisfies ψ, but this also has to be the *first time* (from the current moment on) that ψ is satisfied. Note that this remedies the problem with a potential blow-up indicated by the example above: while $(p \,\mathtt{U}_{[2,4]}\, q) \wedge (p \,\mathtt{U}_{[3,5]}\, q)$ cannot be simplified into a formula using only a single metric Until operator, $(p \,\mathtt{U}^1_{[2,4]}\, q) \wedge (p \,\mathtt{U}^1_{[3,5]}\, q)$ is indeed equivalent to $p \,\mathtt{U}^1_{[3,4]}\, q$, suggesting that the strict first-time semantics makes formulas easier to handle computationally. It is also not hard to see that the proof of ExpSpace-hardness for MTL breaks down under the strict first-time semantics, as suggested above. So apart from the obvious question about MTL^1's expressiveness in relationship to the other variants of LTL, its complexity somewhere between PSpace and ExpSpace is to be pinpointed as well. An ExpSpace upper bound is obtained easily because of a simple linear translation into MTL, based on principles like $\varphi \,\mathtt{U}^1_{[x,y]}\, \psi \equiv (\varphi \wedge \neg\psi) \,\mathtt{U}_{[x,y]}\, \psi$.

This paper is organised as follows. In Section 2 we formally recall known concepts needed for this study, in particular LTL and its extension MTL. We also recall the definition of Streett automata [18] which will serve as a main tool in a decision procedure for MTL^1 which is formally introduced in Section 3. There, we also investigate its expressiveness and succinctness relative to LTL. In Section 4 we construct a singly exponential translation from MTL^1 into Streett automata thus pinpointing the former's complexity as being PSpace-complete only. In Section 5 we conclude with remarks on further work in this area.

2 Preliminaries

We recall the necessary definitions from temporal logics and automata theory.

Linear-time temporal logic. Let $\mathcal{P} = \{p, q, \ldots\}$ be a non-empty, countable set of atomic propositions. Formulas of the linear-time temporal logic LTL are given by the following grammar.

$$\varphi \;::=\; q \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathtt{X}\,\varphi \mid \varphi \,\mathtt{U}\, \varphi$$

where $q \in \mathcal{P}$. Further Boolean operators like $\mathtt{tt}, \mathtt{ff}, \vee, \rightarrow, \leftrightarrow$ are introduced as abbreviations in the usual way, and so are the temporal operators $\mathtt{F}\,\varphi := \mathtt{tt}\,\mathtt{U}\,\varphi$, $\mathtt{G}\,\varphi := \neg\,\mathtt{F}\,\neg\varphi$, $\varphi\,\mathtt{R}\,\psi := \neg(\neg\varphi\,\mathtt{U}\,\neg\psi)$.

The set $\mathit{Sub}(\varphi)$ of subformulas of φ is defined in the usual way. The size of a formula φ is measured in terms of the number of its subformulas: $|\varphi| := |\mathit{Sub}(\varphi)|$. For our purposes here, it makes no difference whether formulas with abbreviated operators are written out in the original syntax or the abbreviated operators are seen as first-class citizens. The size in the former case is only larger by a constant factor compared to the latter case.

Formulas of LTL are interpreted over traces of labelled transition systems which are just ω-words over the alphabet $2^{\mathcal{P}}$. Note that any formula can contain only finitely many propositional symbols. Hence, the alphabet underlying any given formula can always be assumed to be finite. Let $w = a_0 a_1 \ldots \in (2^{\mathcal{P}})^\omega$ and $i \in \mathbb{N}$. Satisfaction of an LTL formula φ in w at position i is explained inductively as follows.

$$\begin{aligned} w, i \models q \quad & \text{iff} \quad q \in a_i \\ w, i \models \varphi \wedge \psi \quad & \text{iff} \quad w, i \models \varphi \text{ and } w, i \models \psi \end{aligned}$$

$$
\begin{array}{lll}
w,i \models \neg\varphi & \text{iff} & w,i \not\models \varphi \\
w,i \models \mathtt{X}\,\varphi & \text{iff} & w,i+1 \models \varphi \\
w,i \models \varphi\,\mathtt{U}\,\psi & \text{iff} & \text{there is } j \geq i \text{ s.t. } w,j \models \psi \text{ and } w,h \models \varphi \text{ for all } h \text{ with } i \leq h < j
\end{array}
$$

A word w satisfies a formula φ, written $w \models \varphi$, if φ is satisfied in its initial position, i.e. $w,0 \models \varphi$. The *language* of a formula φ is the set of all words satisfying it: $L(\varphi) := \{w \mid w \models \varphi\}$. Two formulas are equivalent, written $\varphi \equiv \psi$, when their languages coincide, i.e. $L(\varphi) = L(\psi)$. Note that this is the case if and only if they are satisfied by exactly the same positions in the same words, i.e. $w,i \models \varphi$ iff $w,i \models \psi$. The "if"-direction is trivial because two formulas that are satisfied by the same words at arbitrary positions are clearly satisfied by the same words at initial positions. The "only if"-direction holds because LTL has future-only operators, so we have $a_0a_1\ldots,i \models \varphi$ iff $a_ia_{i+1}\ldots,0 \models \chi$ for any φ. Thus, any position in a word is the initial position of another word, namely the suffix starting at this position.

A formula is *valid*, written $\models \varphi$, if $L(\varphi) = (2^{\mathcal{P}})^\omega$, i.e. it is satisfied by any word. The satisfiability problem – given a formula φ, decide whether $L(\varphi) \neq \emptyset$ – is well-known to be decidable. We write $\mathrm{SAT}(\mathcal{L})$ for the satisfiability problem for logic $\mathcal{L}$. We assume familiarity with standard complexity classes, in particular the space complexity classes NLogSpace, PSpace and ExpSpace. For further details we refer to standard textbooks in complexity theory, e.g. [2].

▶ **Proposition 2** ([17]). *SAT(LTL) is PSpace-complete.*

Metric linear-time temporal logic. The term "metric" temporal logic usually refers to extensions of ordinary temporal logics (like LTL) with modifications of the temporal operators that impose restrictions on the moments that witness their satisfaction in case of an Until or a Finally, resp. relax the future moments under consideration of a Generally, resp. Release formula. The logic MTL is obtained by extending the syntax of LTL with a metric version of Until, written $\varphi\,\mathtt{U}_I\,\psi$ for a non-empty interval I over the natural numbers.

We use standard notation for closed, half-open and open intervals like $[x,y], (x,y], [x,y)$ and (x,y) for $x,y \in \mathbb{N} \cup \{\infty\}$ with $x \leq y$. Because of the discrete nature of $\mathbb{N}$ and the lack of a direct predecessor of ∞ in this structure, we can restrict our attention to intervals of the form $[x,y]$ for $x,y \in \mathbb{N}$, and of the form $[x,\infty)$ for $x \in \mathbb{N}$. Note that $(x,y] = [x+1,y]$ etc.

Intervals of the form $[0,y]$ or $[x,\infty)$ for some $x,y \in \mathbb{N}$ are called *simple* and written more compactly as ${\leq}y$, resp. ${\geq}x$.

The semantics of formulas of MTL is extended accordingly for words $w = a_0a_1\ldots \in (2^{\mathcal{P}})^\omega$ and $i \in \mathbb{N}$ by

$$
\begin{array}{lll}
w,i \models \varphi\,\mathtt{U}_I\,\psi & \text{iff} & \text{there is } j \geq i \text{ s.t. } j-i \in I \text{ and } w,j \models \psi \text{ and} \\
& & w,h \models \varphi \text{ for all } h \text{ with } i \leq h < j
\end{array}
$$

Hence, $\varphi\,\mathtt{U}_I\,\psi$ note only requires a position i in w to satisfy $\varphi\,\mathtt{U}\,\psi$ in the usual sense that some future moment satisfies ψ and all future moments before that satisfy φ. It additionally requires that future moment to be found at a distance which falls into the interval I.

This clearly extends the non-metric version of LTL because $\varphi\,\mathtt{U}\,\psi \equiv \varphi\,\mathtt{U}_{\geq 0}\,\psi$. The metric extension is then applied to the other derived temporal operators $\mathtt{F}, \mathtt{G}$ and $\mathtt{R}$ accordingly.

Because of the use of natural numbers as interval bounds in temporal operators, number of subformulas is not a realistic measure of the representation size of a formula anymore. Note that $\mathtt{F}_{[x,y]}\,p$, stating that p occurs in a word at some distance between x and y would

have constant size regardless of the values of x and y. We assume that such interval bounds are given in binary encoding and therefore measure the size of a formula more appropriately as follows. Let $m(\varphi) :=$

$$\max\{k \in \mathbb{N} \mid \exists \psi_1, \psi_2 \in \mathit{Sub}(\varphi), \exists h \in \mathbb{N} \text{ s.t. } \psi_1 \,\mathtt{U}_I\, \psi_2 \in \mathit{Sub}(\varphi) \text{ for } I = [h,k] \text{ or } I = [k,\infty)\}$$

denote the largest integer constant that is used as an interval bound in a subformula of φ. Then $|\varphi| := |\mathit{Sub}(\varphi)| \cdot \lceil \log m(\varphi) \rceil$.

▶ **Proposition 3** ([10]). *SAT(MTL) is ExpSpace-complete.*

Finite automata on ω-words. Let $\mathcal{P}$ be a finite set of atomic propositions. A nondeterministic Streett automaton (NSA) is an $\mathcal{A} = (Q, \mathcal{P}, q_I, \delta, \mathcal{F})$ where Q is a finite set of states, $q_I \in Q$ is the designated initial state, $\delta \subseteq Q \times \mathbb{B}(\mathcal{P}) \times Q$ is a finite set of transition triples (p, β, q) with $\mathbb{B}(\mathcal{P})$ denoting the set of Boolean formulas over $\mathcal{P}$ (with the usual Boolean operators $\wedge$, $\vee$, $\neg$, $\ldots$). Finally, $\mathcal{F} = \{(F_1, G_1), \ldots, (F_m, G_m)\}$ with $F_i, G_i \subseteq Q$ for all $i = 1, \ldots, m$ is the acceptance condition.

Note that here such finite automata have a symbolically represented transition table instead of the general $\delta \subseteq Q \times \Sigma \times Q$ for a finite alphabet Σ. The reason for this choice is their use in a decision procedure for a temporal logic whose formulas are naturally interpreted over ω-words whose letters are finite sets of propositions, i.e. $\Sigma = 2^{\mathcal{P}}$. Instead of enumerating all such possible sets and explaining the automaton's one-step behaviour for each step, the symbolic representation here is more convenient for such special cases. There are straightforward translations for the transition relations between symbolic and explicit representations, and for a fixed set of propositions, they are polynomial. Hence, we can safely use Streett automata in this form and still appeal to known results about them, even though they were perhaps formulated for the form with an explicit alphabet.

A run of the NSA $\mathcal{A} = (Q, \mathcal{P}, q_I, \delta, \mathcal{F})$ with $\mathcal{F} = \{(F_1, G_1), \ldots, (F_m, G_m)\}$ on a word $w = a_0 a_1 \ldots \in (2^{\mathcal{P}})^\omega$ is a $\rho = q_0, q_1, \ldots \in Q^\omega$ such that $q_0 = q_I$ and for all $i \in \mathbb{N}$ there is some $(p, \beta, q) \in \delta$ such that $q_i = p$, $a_i \models \beta$ and $q = q_{i+1}$. Here, satisfaction of a Boolean formula β over $\mathcal{P}$ by a set $a \subseteq \mathcal{P}$, $a \models \beta$, is explained in the usual way: β evaluates to true when all $q \in a$ are set to true and all $q \notin a$ are set to false.

We write $\mathit{Inf}(\rho)$ to denote the set of all states that occur infinitely often in ρ. By finiteness of Q, we always have $\mathit{Inf}(\rho) \neq \emptyset$.

The run ρ is accepting if it satisfies the Streett condition $\mathcal{F}$ in the sense that for all $i = 1, \ldots, m$: if $\mathit{Inf}(\rho) \cap F_i \neq \emptyset$ then $\mathit{Inf}(\rho) \cap G_i \neq \emptyset$. As usual, $L(\mathcal{A})$ is the language of the NSA $\mathcal{A}$, and it consists of all words on which $\mathcal{A}$ has an accepting run.

So both NSA and formulas of linear-time temporal logics defines languages of ω-words which is why these different formalisms can be compared to one another in terms of expressiveness, and satisfiability of a formula corresponds to non-emptiness of the language of an automaton. Algorithms for non-emptiness problems for ω-automata are routinely used to obtain decision procedures for linear-time temporal logics [21, 20, 4]. It has been shown that non-emptiness for Streett automata can be decided in polynomial time [8, 12] and this requires an explicit construction. A polynomial (equivalence-preserving) translation into Büchi automata is not possible, let alone one computable in logarithmic space. This would immediately transfer the upper bound of NLogSpace to Streett automata. Nevertheless, it does hold, too; it simply needs to be shown directly.

▶ **Theorem 4.** *The non-emptiness problem for NSA is decidable in NLogSpace.*

Proof. The key observation is the following. The language of an NSA $\mathcal{A} = (Q, \mathcal{P}, q_I, \delta, \mathcal{F})$ with $n := |Q|$ and $\mathcal{F} = \{(F_1, G_1), \ldots, (F_k, G_k)\}$ is non-empty iff there is an ultimately periodic word $w = uv^\omega \in L(\mathcal{A})$. This follows from finiteness of n and k by the pigeon hole principle. An accepting run on an arbitrary word must eventually traverse a state q for the second time such that in between, for every $i = 1, \ldots, k$, either no state from F_i or some state from G_i has been seen.

This gives rise to a nondeterministic algorithm for deciding non-emptiness. It guesses, step-by-step, the states of a run and also nondeterministically remembers some state q occurring in this simulation. It then maintains $2k$ bits to remember, for each $i = 1, \ldots, k$, whether some state in F_i and some state in G_i has been seen. It accepts, when q occurs again, and the bits indicate that the Streett condition has been met in between.

In order to terminate and reject on computation paths with unsuccessful guesses, it counts the number of steps done in this simulation. It is not hard to see that the first occurrence of q can be required to occur after at most n steps. The second occurrence can be expected to occur after no more than a further $2nk$ steps, for otherwise the run contains parts that could be skipped. Hence, the space needed for the counter is at most logarithmic in $|\mathcal{A}|$. ◀

3 Metric LTL with First-Time Semantics

Syntax and Semantics. We introduce *Metric Linear-Time Temporal Logic with Strict First-Time Semantics* (MTL^1) which, instead of metric Until formulas of the form $\varphi \,\mathtt{U}_I\, \psi$, features a special modification $\varphi \,\mathtt{U}^1_I\, \psi$ that is interpreted under *strict first-time semantics*. Intuitively, it does not just demand that *some* occurrence of ψ in the future happens at a distance that falls into the interval I. Instead, it requires this to be the *first* time in the future that this happens.

▶ **Definition 5.** Let $\mathcal{P} = \{p, q, \ldots\}$ be a non-empty, countable set of atomic propositions as usual. Formulas of the linear-time temporal logic MTL^1 are built according to the following grammar.

$$\varphi \;::=\; q \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathtt{X}\,\varphi \mid \varphi \,\mathtt{U}^1_I\, \varphi$$

where $q \in \mathcal{P}$ and I is an interval over $\mathbb{N}$ as discussed above.

We also introduce its fragment sMTL^1– *Simple* MTL^1– which only features simple intervals $\leq k$ or $\geq k$ in its formulas. Other Boolean and temporal operators, in particular the strict first-time variants $\mathtt{F}^1_I$, $\mathtt{G}^1_I$ and $\mathtt{R}^1_I$ are introduced as abbreviations in the usual way. The set $Sub(\varphi)$ of subformulas of an MTL^1 formula φ is defined as usual by induction over the syntax tree of the formula.

The intuition behind the restriction to first-time occurrences is made formal as follows.

▶ **Definition 6.** Let $\mathcal{P}$ be given as above and $w = a_0 a_1 \ldots \in (2^{\mathcal{P}})^\omega$. Satisfaction of an MTL^1 formula φ at a position i of the word w is explained inductively over the structure of φ as for LTL, apart from the following case.

$$w, i \models \varphi \,\mathtt{U}^1_I\, \psi \quad \text{iff} \quad \begin{array}{l} \text{there is } j \geq i \text{ s.t. } j - i \in I, \text{ and } w, j \models \psi \text{ and} \\ \qquad w, h \models \varphi \wedge \neg\psi \text{ for all } h \text{ s.t. } i \leq h < j \end{array}$$

Expressiveness. LTL trivially embeds into sMTL^1 which trivially embeds into MTL^1. The reason for this is the fact whenever an Until-formula gets satisfied *somewhere* then there is also a first time that this happens. A more interesting question concerns the other direction, and therefore also the connection to MTL. Since LTL can trivially be embedded into MTL, we immediately obtain a translation from MTL^1 into MTL, from one from MTL^1 into LTL.

We remark that the translations introduced in the following are not polynomial for formula length, measure in terms of length of string representations, but only for formula size, measured in terms of number of subformulas, as they often require the duplication of subformulas.

The first observation about expressiveness is the expressive equivalence between MTL^1 and sMTL^1. This is perhaps a little bit surprising as this principle does not apply to MTL.

▶ **Theorem 7.** *For every $\varphi \in MTL^1$ there is a $\widehat{\varphi} \in sMTL^1$ of size $\mathcal{O}(|\varphi|)$ such that $\widehat{\varphi} \equiv \varphi$.*

Proof. We can define $\widehat{\varphi}$ inductively. The only non-trivial case is that of an Until formula. Then we have

$$\widehat{\varphi\, \mathtt{U}^1_{[x,y]}\, \psi} \;:=\; (\widehat{\varphi}\, \mathtt{U}^1_{\geq x}\, \widehat{\psi}) \wedge (\widehat{\varphi}\, \mathtt{U}^1_{\leq y}\, \widehat{\psi})$$

for $x, y \in \mathbb{N}$. Clearly, this increases the number of subformulas at most by factor 3. Correctness of this translation is straightforward by inspection of the semantics. ◀

Because of Thm. 7 we can restrict our attention to sMTL^1 formulas down below as this simplifies the technical details of various constructions slightly.

One reason for considering numerical values in formulas to be represented in binary (as opposed to unary), apart from this being natural, is the fact that the expressive power of sMTL^1– and that of MTL in fact – does not exceed that of LTL. However, translations back into LTL are polynomial in the value of interval bounds only, i.e. they are in fact exponential in the size of a formula.

▶ **Theorem 8.** *For every $sMTL^1$ formula φ there is an LTL formula $\widehat{\varphi}$ such that $\widehat{\varphi} \equiv \varphi$ and $|\widehat{\varphi}| = 2^{\mathcal{O}(|\varphi|)}$.*

Proof. We define a translation $\widehat{\cdot}\colon \mathrm{sMTL}^1 \to \mathrm{LTL}$ as follows.

$$\widehat{q} := q \qquad \widehat{\varphi \wedge \psi} := \widehat{\varphi} \wedge \widehat{\psi} \qquad \widehat{\neg\varphi} := \neg\widehat{\varphi} \qquad \widehat{\mathtt{X}\,\varphi} := \mathtt{X}\,\widehat{\varphi}$$

$$\widehat{\varphi\, \mathtt{U}^1_{\leq k}\, \psi} := \widehat{\psi} \vec{\vee} \underbrace{(\widehat{\varphi} \wedge \mathtt{X}(\widehat{\psi} \vec{\vee} \ldots \vec{\vee}\, \mathtt{X}(\widehat{\psi} \vec{\vee} (\widehat{\varphi} \wedge \mathtt{X}\,\widehat{\psi}) \ldots)))}_{k \text{ occurrences of } \mathtt{X}}$$

$$\widehat{\varphi\, \mathtt{U}^1_{\geq k}\, \psi} := \underbrace{\widehat{\varphi} \wedge \neg\widehat{\psi} \wedge \mathtt{X}(\widehat{\varphi} \wedge \neg\widehat{\psi} \wedge \mathtt{X}(\ldots \mathtt{X}(\widehat{\varphi} \wedge \neg\widehat{\psi} \wedge \mathtt{X}(\varphi\, \mathtt{U}\, \psi)) \ldots))}_{k \text{ occurrences of } \mathtt{X}}$$

The translation of an operator $\mathtt{U}^1_{\sim k}$ clearly produces a formula whose size is linear in the value k, i.e. exponential in the representation size of k. Replacing the abbreviated biased disjunctions by plain Boolean formulas only incurs a further polynomial blow-up because formula size is measured in terms of number of subformulas.

Correctness of the translation is proved by a straightforward induction on the structure of φ, showing that for all $w \in (2^{\mathcal{P}})^\omega$ and all $i \in \mathbb{N}$, we have $w, i \models \widehat{\varphi}$ iff $w, i \models \varphi$. ◀

Succinctness. The exponential blow-up predicted by Thm. 8 may seem like a downside at first sight. However, it should be read as the possibility that certain properties, which are definable in LTL, can be defined in sMTL^1 with much shorter formulas. The same

phenomenon is of course known from LTL. It is easy to show that every family of LTL formulas equivalent to the MTL formulas $\varphi_n := \mathtt{F}_{\geq 2^n}\, q$ require size $\mathcal{O}(2^n)$. The proof can be re-used entirely to show that sMTL^1 is also exponentially more succinct than LTL. Note that $|\,\mathtt{F}^1_{\geq 2^n}\, q| = \mathcal{O}(n)$.

▶ **Theorem 9.** *There is a family of LTL-definable languages $(L_n)_{n\geq 1}$ over a singleton $\mathcal{P}$ such that each L_n is expressible in sMTL^1 by a formula of size $\mathcal{O}(n)$ but every family $(\varphi_n)_{n\geq 1}$ of LTL formulas with $L(\varphi_n) = L_n$ is such that $|\varphi_n| = \Omega(2^n)$.*

Proof. Consider the sMTL^1 formulas $\varphi_n := \mathtt{F}^1_{\geq 2^n}\, q$ for $n \geq 1$. Clearly, $|\varphi_n| = \mathcal{O}(n)$. By a standard induction on the structure of LTL formulas we can show that formulas of size $< 2^n$ cannot distinguish between the two words $w_n = \emptyset^{2^n-1}\{q\}\emptyset^\omega$ and $w'_n = \emptyset^{2^n}\{q\}\emptyset^\omega$. Since $w_n \models \varphi_n$ but $w'_n \not\models \varphi_n$ for all $n \geq 1$, we get that any presumed LTL-formula equivalent to φ_n needs to have size 2^n at least. ◀

4 An Automata-Theoretic Decision Procedure

We give an automata-theoretic decision procedure for MTL^1. Decidability of its satisfiability problem is not a surprise in the light of Thm. 8, stating that sMTL^1– and therefore also MTL^1 according to Thm. 7 – can be translated into LTL at an exponential blow-up. This is unavoidable according to Thm. 9. Then it is perhaps rather surprising that the complexity of MTL^1 is asymptotically no worse than that of LTL. We give an upper bound of PSpace, based on a translation into Streett automata. According to Thm. 8, it suffices to do so for sMTL^1.

Temporal formulas and their unfoldings. For convenience, we work with sMTL^1 formulas in *negation normal form* (NNF), i.e. those that are built from literals $q, \neg q$ using the Boolean operators $\wedge, \vee$ and the temporal operators $\mathtt{X}, \mathtt{U}^1$ and $\mathtt{R}^1$ where $\varphi\, \mathtt{R}^1_{\sim k}\, \psi := \neg(\neg\varphi\, \mathtt{U}^1_{\sim k}\, \neg\psi)$. The following is a standard observation about the ability to push negations inwards in formulas to obtain NNF.

▶ **Lemma 10.** *For every sMTL^1 formula φ there is an sMTL^1 formula $\overline{\varphi}$ in NNF of size $\mathcal{O}(|\varphi|)$ such that $\overline{\varphi} \equiv \varphi$.*

The construction of a Streett automaton recognising $L(\varphi)$ for some sMTL^1 formula φ in NNF then follows the same principles as the standard construction of a Büchi automaton for an LTL formula. Temporal operators are typically handled by unfolding, not just in automata-theoretic decision procedures. The term denotes the two equivalences

$$\varphi\, \mathtt{U}\, \psi \equiv \psi \vee (\varphi \wedge \mathtt{X}(\varphi\, \mathtt{U}\, \psi)) \quad \text{and} \quad \varphi\, \mathtt{R}\, \psi \equiv \psi \wedge (\varphi \vee \mathtt{X}(\varphi\, \mathtt{R}\, \psi))\ .$$

These can be extended to the temporal operators in sMTL^1 as follows. The proof is just by close inspection of the semantics of MTL^1.

▶ **Lemma 11.** *Let $\varphi, \psi \in$ sMTL^1, $k \geq 0$. We have*

$$\begin{aligned}
\varphi\, \mathtt{U}^1_{\leq 0}\, \psi &\equiv \psi & \qquad \varphi\, \mathtt{R}^1_{\leq 0}\, \psi &\equiv \psi \\
\varphi\, \mathtt{U}^1_{\geq 0}\, \psi &\equiv \psi \vee (\varphi \wedge \mathtt{X}(\varphi\, \mathtt{U}^1_{\geq 0}\, \psi)) & \varphi\, \mathtt{R}^1_{\geq 0}\, \psi &\equiv \psi \wedge (\varphi \vee \mathtt{X}(\varphi\, \mathtt{R}^1_{\geq 0}\, \psi)) \\
\varphi\, \mathtt{U}^1_{\leq k+1}\, \psi &\equiv \psi \vee (\varphi \wedge \mathtt{X}(\varphi\, \mathtt{U}^1_{\leq k}\, \psi)) & \varphi\, \mathtt{R}^1_{\leq k+1}\, \psi &\equiv \psi \wedge (\varphi \vee \mathtt{X}(\varphi\, \mathtt{R}^1_{\leq k}\, \psi)) \\
\varphi\, \mathtt{U}^1_{\geq k+1}\, \psi &\equiv \neg\psi \wedge \varphi \wedge \mathtt{X}(\varphi\, \mathtt{U}^1_{\geq k}\, \psi) & \varphi\, \mathtt{R}^1_{\geq k+1}\, \psi &\equiv \neg\psi \vee \varphi \vee \mathtt{X}(\varphi\, \mathtt{R}^1_{\geq k}\, \psi)
\end{aligned}$$

A formula χ of the form on one of the left-hand sides in these equations is called a *temporal formula* and we use $unf(\chi)$ to denote the corresponding right-hand side. Temporal formulas of the form $\varphi\,\mathtt{U}^1_{\geq k}\,\psi$, i.e. Until-formulas whose metric parameter is a half-open interval to the right, are called *critical*.

Later on we will need a second observation about temporal formulas. The proof is straightforward from an inspection of the semantics of MTL^1. The lemma already holds for MTL in fact.

▶ **Lemma 12.** *Let φ, ψ be sMTL^1 formulas and $k, \ell \in \mathbb{N}$ such that $k \leq \ell$. Then we have*

$$\models (\varphi\,\mathtt{U}^1_{\leq k}\,\psi) \to (\varphi\,\mathtt{U}^1_{\leq \ell}\,\psi) \qquad \models (\varphi\,\mathtt{R}^1_{\leq \ell}\,\psi) \to (\varphi\,\mathtt{R}^1_{\leq k}\,\psi)$$
$$\models (\varphi\,\mathtt{R}^1_{\geq k}\,\psi) \to (\varphi\,\mathtt{R}^1_{\geq \ell}\,\psi) \qquad \models (\varphi\,\mathtt{U}^1_{\geq \ell}\,\psi) \to (\varphi\,\mathtt{U}^1_{\geq k}\,\psi)$$

The Fischer-Ladner closure of a formula χ is a collection of all subformulas and perhaps others derived from them that may play a role in determining the truth of χ at some point in a word.

▶ **Definition 13.** *Let $\chi \in$ sMTL^1 be in NNF. Its* Fischer-Ladner closure *is the least set $FL(\chi)$ that contains χ and is closed under the following operations.*

- *If $\varphi \wedge \psi \in FL(\chi)$ or $\varphi \vee \psi \in FL(\chi)$ then $\{\varphi, \psi\} \subseteq FL(\varphi)$.*
- *If $\mathtt{X}\,\varphi \in FL(\chi)$ then $\varphi \in FL(\chi)$.*
- *If $\varphi \in FL(\chi)$ for a temporal φ, then $unf(\varphi) \in FL(\chi)$.*

The key concept in an automata construction for sMTL^1 formulas is that of a Hintikka set – a set of formulas that is closed under propositional logic consequence. We need to refine the standard definition slightly in order to obtain the intended complexity bound in the end.

▶ **Definition 14.** Let $\chi \in$ sMTL^1 be in NNF. A $\Phi \subseteq FL(\chi)$ is called a *Hintikka set* for χ if it satisfies the following conditions.

- If $\varphi \wedge \psi \in \Phi$ then $\{\varphi, \psi\} \subseteq \Phi$.
- If $\varphi \vee \psi \in \Phi$ then $\{\varphi, \psi\} \cap \Phi \neq \emptyset$.
- If $\varphi \in \Phi$ for a temporal φ then $unf(\varphi) \in \Phi$.

Φ is called *(propositionally) consistent* if there is no $q \in \mathcal{P}$ such that $\{q, \neg q\} \subseteq \Phi$. It is called *lean* if for all φ, ψ there is at most one temporal formula $\varphi\,\mathtt{U}^1_{\leq k}\,\psi \in \Phi$ for any $k \geq 0$, and likewise for temporal formulas of the three other forms with operators $\mathtt{U}^1_{\geq k}$, $\mathtt{R}^1_{\leq k}$ and $\mathtt{R}^1_{\geq k}$. We write $\mathcal{H}(\chi)$, resp. $\mathcal{H}_{\mathsf{ln}}(\chi)$ for the set of all lean, propositionally consistent Hintikka sets for χ, resp. those that are additionally lean.

While the Fischer-Ladner closure of an sMTL^1 formula χ is generally exponential in $|\chi|$ and there are, thus, doubly exponentially many Hintikka sets, there are only singly exponentially many lean Hintikka sets.

▶ **Lemma 15.** *Let $\chi \in$ sMTL^1 be in NNF. Then $|\mathcal{H}_{\mathsf{ln}}(\chi)| = 2^{\mathcal{O}(|\chi|^2)}$.*

Proof. Note that there are at most $\mathcal{O}(|\chi|)$ many formula schemes, i.e. members of $FL(\chi)$ modulo concrete metric parameters. A lean set can then be seen as a mapping for each such formula scheme to a value in $\{\bot, 0, \ldots, m(\chi)\}$, indicating (non-)inclusion in the set and giving a concrete parameter value for a temporal formula. Since $m(\chi) \in 2^{\mathcal{O}(|\chi|)}$ due to binary encoding, there are at most $(2^{\mathcal{O}(|\chi|)})^{\mathcal{O}(|\chi|)} = 2^{\mathcal{O}(|\chi|^2)}$ many lean (Hintikka) sets. ◀

Streett automata for sMTL^1 formulas. We are now in a position to define a Streett automaton of singly exponential size that recognises exactly the models of an sMTL^1 formula in NNF.

Fix an sMTL^1 formula χ in NNF over some finite $\mathcal{P}$. We will need three constructions on (lean) Hintikka sets for χ. The first one collects all literals in a lean Hintikka set.

$$Now(\Phi) \;:=\; \bigwedge_{q\in\Phi} q \wedge \bigwedge_{\neg q\in\Phi} \neg q$$

It can be seen as the extraction of a propositional formula determining the letter at a position in a word where all formulas in Φ are supposed to be true. On the other hand,

$$Nxts(\Phi) \;:=\; \{ln(\Psi) \mid \Psi \in \mathcal{H}(\chi) \text{ and } \forall\, \mathtt{X}\,\psi \in \Phi : \psi \in \Psi\}$$

collects all "leanifications" of Hintikka sets that are potential successors to Φ in the sense that they contain all formula ψ which Φ needs to be true at the next position. The leanification $ln(\Psi)$ of Ψ is obtained by successively replacing temporal formulas as follows until no further steps are applicable.

- If $\{\varphi\,\mathtt{U}^1_{\leq k}\,\psi, \varphi\,\mathtt{U}^1_{\leq \ell}\,\psi\} \subseteq \Psi$ or $\{\varphi\,\mathtt{R}^1_{\geq k}\,\psi, \varphi\,\mathtt{R}^1_{\geq \ell}\,\psi\} \subseteq \Psi$ for some φ, ψ and $k < \ell$ then replace all occurrences of the latter that do not occur under the scope of a $\mathtt{X}$-operator by the former.
- If $\{\varphi\,\mathtt{U}^1_{\geq k}\,\psi, \varphi\,\mathtt{U}^1_{\geq \ell}\,\psi\} \subseteq \Psi$ or $\{\varphi\,\mathtt{R}^1_{\leq k}\,\psi, \varphi\,\mathtt{R}^1_{\leq \ell}\,\psi\} \subseteq \Psi$ for some φ, ψ and $k < \ell$ then replace all occurrences of the former that do not occur under the scope of a $\mathtt{X}$-operator by the latter.

It should be clear that $Nxts(\Phi)$ is indeed a set of lean Hintikka sets for χ.

The important observation about the leanification process is the preservation of the semantics in a strong sense.

▶ **Lemma 16.** *Let $\chi \in sMTL^1$ be in NNF and $\Phi \in \mathcal{H}(\chi)$. For every $\varphi \in \Phi$ there is $\varphi' \in ln(\Phi)$ such that φ' differs from φ only in the values of metric parameters and $\models \varphi' \to \varphi$.*

Proof. This is proved in a straight-forward induction on the structure of φ. The only non-trivial cases are those of temporal formulas. These are covered by Lemma 12. Since leanification also replaces subformulas, we also need the fact that χ (and all its subformulas) are given in NNF. Hence, if $\models \varphi' \to \varphi$ then also $\models \psi \to \psi[\varphi/\varphi']$ for any ψ, i.e. all formulas are monotonic. ◀

The acceptance condition of the NSA $\mathcal{A}_\chi$ is determined by the set of all critical temporal formulas that can occur in lean Hintikka sets for χ. However, unfolding decreases interval bounds by one, so the number of critical temporal formulas in $FL(\chi)$ is exponential: if χ contains the subformula $\varphi\,\mathtt{U}^1_{\geq k}\,\psi$ for some $k \in \mathbb{N}$, then $FL(\chi)$ contains $\varphi\,\mathtt{U}^1_{\geq h}\,\psi$ for all $h \leq k$, i.e. exponentially many in $|\chi|$ because of binary encodings of interval bounds.

The exact interval bounds are irrelevant for the acceptance condition. It is only used to ensure that no critical temporal formula $\varphi\,\mathtt{U}^1_{\geq k}\,\psi$ gets unfolded forever without ever satisfying its right argument ψ. We introduce the notion of a critical formula *scheme* $\varphi\,\mathtt{U}^1_{\geq *}\,\psi$ and write $(\varphi\,\mathtt{U}^1_{\geq *}\,\psi) \in \Phi$ if there is some $k \in \mathbb{N}$ such that $(\varphi\,\mathtt{U}^1_{\geq k}\,\psi) \in \Phi$.

Note that there is no need to treat the other three kinds of temporal formulas in the same way. A temporal Release-formula is a greatest fixpoint, and unfolding it infinitely often is a legitimate way of determining its truth. A non-critical, temporal Until-formula of the form $\varphi\,\mathtt{U}^1_{\leq k}\,\psi$ cannot get unfolded infinitely often because each unfolding step decreases the metric parameter in it until it eventually becomes 0, and the formula is replaced by ψ anyway. Note

that this is not the case for critical Until-formulas. Unfolding them still decreases the metric parameter. However, the leanification process can increase it again, whereas leanification for non-critical Until-formulas can only decrease it further.

Let $\{\gamma_1, \dots, \gamma_n\}$ be the set of all schemes of critical temporal Until-formulas in $FL(\chi)$, i.e. $\gamma_i = \alpha_i \,\mathtt{U}^1_{\geq *}\, \beta_i$ for some α_i, β_i. We define the NSA $\mathcal{A}_\chi$ as $(\mathcal{H}_{\mathsf{ln}}(\chi), \mathcal{P}, I, \delta, \mathcal{F})$ where

- $I := \{\Phi \in \mathcal{H}_{\mathsf{ln}}(\chi) \mid \chi \in \Phi\}$,
- $\delta := \{(\Phi, \mathit{Now}(\Phi), \Psi) \mid \Psi \in \mathit{Nxts}(\Phi)\}$,
- $\mathcal{F} := \{(F_1, G_1), \dots, (F_n, G_n)\}$ with $F_i := \{\Phi \mid \gamma_i \in \Phi\}$ and $G_i := \{\Phi \mid \beta_i \in \Phi\}$. Note that β_i may contain other temporal formulas, so β_i is to be understood as a scheme potentially, and $\beta_i \in \Phi$ means that some formula deviating from β_i in metric parameters only is contained in Φ.

The next two lemmas are devoted to the soundness and completeness of the construction.

▶ **Lemma 17.** *Let $\chi \in sMTL^1$ over $\mathcal{P}$ be in NNF and $\mathcal{A}_\chi$ as above. Then $L(\mathcal{A}_\chi) \subseteq L(\chi)$.*

Proof. Let $w = a_0 a_1 \dots (2^{\mathcal{P}})^\omega$ be such that there is an accepting run $\rho = \Phi_0, \Phi_1, \dots$ of $\mathcal{A}_\chi$ on w. By the construction of $\mathcal{A}_\chi$ we have (I) $\chi \in \Phi_0$ and, for all $i \geq 0$, (II) $a_i \models \mathit{Now}(\Phi_i)$ and (III) there is $\Psi_{i+1} \in \mathcal{H}(\chi)$ such that $\psi \in \Psi_{i+1}$ for all $\mathtt{X}\,\psi \in \Phi_i$ and $\Phi_{i+1} = \mathit{ln}(\Psi_{i+1}) \in \mathit{Nxts}(\Phi_i)$.

We show by induction on the structure of formulas φ that for all $i \in \mathbb{N}$ and all $\varphi \in \Phi_i$ we have $w, i \models \varphi$. For literals q or $\neg q$ this follows immediately from (II). For conjunctions and disjunctions this follows by the hypothesis for both conjuncts, resp. one disjunct and the fact that each Φ_i is a lean Hintikka set that behaves like a Hintikka set in this case, i.e. it contains both conjuncts of a conjunction etc. This is the case because leanification replaces either none or all occurrences that are not under the scope of a $\mathtt{X}$-operator. Hence, a replacement takes place in a conjunction iff it takes place in both conjuncts etc.

Suppose φ is of the form $\mathtt{X}\,\psi$. Because of (III), there is a $\Psi_{i+1} \in \mathcal{H}(\chi)$ with $\psi \in \Psi_{i+1}$ with $\Phi_{i+1} = \mathit{ln}(\Psi_{i+1})$. According to Lemma 16, there is $\psi' \in \Phi_{i+1}$ that is structurally not greater than ψ. Hence, we can apply the induction hypothesis to it and obtain $w, i+1 \models \psi'$. According to Lemma 16, we then also have $w, i+1 \models \psi$ and therefore $w, i \models \varphi$.

Suppose φ is of the form $\psi_1 \,\mathtt{U}^1_{\leq k}\, \psi_2$. By inspection of the unfolding rule (Lemma 11) and successive applications of the principles (II) and (III) with the same kind of reasoning using Lemma 16, we get some $k' \leq k$ and a sequence $\Psi_{i+1}, \dots, \Psi_{i+k'}$ of Hintikka sets such that $\Phi_{i+j} = \mathit{ln}(\Psi_{i+j})$, $\psi' \in \Psi_{i+k'}$ for some ψ' with $\models \psi'_2 \to \psi_2$ and $\psi''_h \in \Phi_{i+h}$ for some $\psi''_0, \dots, \psi''_{k'-1}$ such that $\models \psi''_h \to \psi_1$. Applying the induction hypothesis to ψ'_2 at position $i+k'$ and for $\psi''_0, \dots, \psi''_{k'-1}$ at positions $i, \dots, i+k'-1$ shows that these are satisfied at the respective positions in w. Lemma 16 then yields $w, i+k' \models \psi_2$ and $w, i+h \models \psi_1$ for all $h = 0, \dots, k'-1$. Thus, $w, i \models \varphi$.

Suppose φ is of the form $\psi_1 \,\mathtt{U}^1_{\geq k}\, \psi_2$. Note that it is a critical temporal formula in this case. We can apply the same reasoning as in the previous case using Lemmas 11 and 16. However, here the leanification process may replace metric parameters by larger ones. Hence, this alone does not guarantee the existence of a $k' \geq k$ such that $\psi_2 \in \Phi_{i+k'}$. This is where $\mathcal{A}_\chi$'s acceptance condition comes into place. Since ρ is accepting, it must either contain finitely many lean Hintikka sets containing φ or infinitely many containing ψ_2. The latter case immediately implies the existence of such a $k' \geq k$. The former case does so, too, by inspection of Lemma 11 and the construction of Hintikka sets. It is only possible to have (the scheme) φ finitely often only when some $\Phi_{i+k'}$ contains $\psi_1 \,\mathtt{U}^1_{\geq k}\, \psi_2$ and therefore also ψ_2. The rest of this case is handled as the previous one.

The remaining two cases of temporal Release-formulas are also handled in a way that is analogous to those of the Until-formulas.

At last, (I) says that $\chi \in \Phi_0$. By the reasoning above we then have $w, 0 \models \chi$, i.e. $w \in L(\chi)$ which completes the claim. ◀

▶ **Lemma 18.** *Let $\chi \in sMTL^1$ over $\mathcal{P}$ be in NNF and $\mathcal{A}_\chi$ as above. Then $L(\chi) \subseteq L(\mathcal{A}_\chi)$.*

Proof. Suppose $w = a_0 a_1 \ldots \in L(\chi)$. We need to construct an accepting run $\Phi_0, \Phi_1, \ldots$ of $\mathcal{A}_\chi$ on w. To this end, we construct a sequence $\Phi'_0, \Phi'_1, \ldots$ via $\Phi'_i := \{\varphi \in FL(\chi) \mid w, i \models \varphi\}$. It is not hard to see that each Φ'_i is indeed a propositionally consistent Hintikka set. Moreover, if $\mathtt{X}\varphi \in \Phi_i$ then $\varphi \in \Phi_{i+1}$. This therefore determines a sequence of lean Hintikka sets by leanification: $\Phi_i := ln(\Phi'_i)$ for all $i \geq 0$, forming a run $\rho = \Phi_0, \Phi_1, \ldots$.

It remains to be seen that it is indeed an accepting run. Take some critical Until-formula scheme $\gamma = \alpha \,\mathtt{U}^1_{\geq *}\, \beta$ and suppose that ρ contains an infinite subsequence $\Phi_{i_1}, \Phi_{i_2}, \ldots$ with $\gamma_j := (\alpha \,\mathtt{U}^1_{\geq k_j}\, \beta) \in \Phi_{i_j}$ for all $j \geq 1$. By construction, we have $w, i_j \models \gamma_j$. By the semantics of MTL^1 there are $k'_1, k'_2, \ldots$ with $k'_j \geq k_j$ such that $w, i_j + k'_j \models \beta$ and, again, by construction $\beta \in \Phi'_{i_j + k'_j}$. Since $i_1 < i_2 < \ldots$, the set $\{i_j + k'_j \mid j \geq 1\}$ is infinite. Hence, the sequence $\Phi'_0, \Phi'_1, \ldots$ has an infinite subsequence in which every Hintikka set contains β. Then the run ρ has an infinite subsequence in which every lean Hintikka set either contains β itself or an instantiation of the scheme β. In any case, the run ρ satisfies the Streett pair associated with γ. Since this is the case for any critical γ, ρ is indeed accepting and so we have $w \in L(\mathcal{A}_\chi)$. ◀

Putting all of the above together we obtain that MTL^1 is not just decidable but that its satisfiability problem is no worse than that of ordinary LTL (and therefore exponentially easier than that of MTL), even though there is an exponential succinctness gap between MTL^1 and LTL.

▶ **Theorem 19.** *SAT(MTL^1) is PSpace-complete.*

Proof. The lower bound is straightforwardly inherited from LTL. For the upper bound, note that every MTL^1 formula χ can be translated into an equivalent sMTL^1 formula in NNF at a linear blow-up only (Thm. 7 and Lemma 10). This can in turn be translated into an equivalent NSA (Lemmas 17 and 18) of exponential size in $|\chi|$ (Lemma 15). Language equivalence entails particularly that its language is non-empty iff χ is satisfiable. Non-emptiness for NSA can be decided in NLogSpace (Thm. 4) which is NPSpace measured in the size of χ. Savitch's Theorem [15] then gives a PSpace upper bound. ◀

5 Conclusion

We have investigated the expressiveness and computational complexity of MTL^1, a variant of Metric Linear-Time Temporal logic MTL in wich the metric parameters do not constrain the occurrence of some event but their first occurrence (in an Until formula). The resulting logic is still exponentially more succinct than LTL. Unlike full MTL whose satisfiability problem is ExpSpace-complete, we obtained a PSpace upper bound for MTL^1 by the construction of equivalent Streett automata of exponential size.

The main motivation for the study of this logic is given by links to State-Space Models in machine learning. Further work will elaborate on the connections between these formalisms. On the side of temporal logics, there is obvious further work in terms generalisations of the strict first-time semantics to a strict n-th time semantics, constraining further moments in which an Until-formula gets satisfied. We suspect that for every fixed n, the resulting logic MTL^n remains PSpace-complete.

There are also obvious connections to Counting LTL [11], a variant of LTL with a counting operator. The first-time semantics is clearly expressible using counting operators by stating that the number of positions beforehand is zero. The relative succinctness between the two formalisms remains to be investigated, as is the case for MTL and MTL[1].

References

1 R. Alur and T. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993. doi:10.1006/INCO.1993.1025.

2 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge Univ. Press, 2006.

3 B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Proc. Coll. on Temporal Logic in Specification*, volume 398 of *LNCS*, pages 62–73. Springer, 1989. doi:10.1007/3-540-51803-7_22.

4 S. Demri, V. Goranko, and M. Lange. *Temporal Logics in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

5 C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Series on Integrated Circuits and Systems. Springer, 2006. doi:10.1007/978-0-387-36123-9.

6 D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th Symp. on Principles of Programming Languages, POPL'80*, pages 163–173. ACM, 1980. doi:10.1145/567446.567462.

7 G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. 23rd Int. Joint Conf. on A.I., IJCAI'13*, pages 854–860. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

8 M. Rauch Henzinger and J. A. Telle. Faster algorithms for the nonemptiness of streett automata and for communication protocol pruning. In *Proc. 5th Scand. Workshop on Algorithm Theory, SWAT'96*, volume 1097 of *LNCS*, pages 16–27. Springer, 1996. doi:10.1007/3-540-61422-2_117.

9 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real Time Syst.*, 2(4):255–299, 1990. doi:10.1007/BF01995674.

10 F. Laroussinie, N. Markey, and Ph. Schnoebelen. Efficient timed model checking for discrete-time systems. *Theoretical Computer Science*, 353(1):249–271, 2006. doi:10.1016/j.tcs.2005.11.020.

11 F. Laroussinie, A. Meyer, and E. Petonnet. Counting LTL. In *Proc. 17th Int. Symp. on Temporal Representation and Reasoning, TIME'10*, pages 51–58. IEEE, 2010. doi:10.1109/TIME.2010.20.

12 T. Latvala and K. Heljanko. Coping with strong fairness. *Fundam. Informaticae*, 43(1-4):175–193, 2000. doi:10.3233/FI-2000-43123409.

13 W. Merrill, J. Petty, and A. Sabharwal. The illusion of state in state-space models. In *Proc. 41st Int. Conf. on Machine Learning, ICML'24*, volume 235, pages 35492–35506. JMLR.org, 2024.

14 A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. on Foundations of Computer Science, FOCS'77*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.

15 W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

16 A. P. Sistla. *Theoretical Issues in the Design of Distributed and Concurrent Systems*. PhD thesis, Harvard Univ., Cambridge, MA, 1983.

17 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

18 R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982. doi:10.1016/S0019-9958(82)91258-X.

19 M. Y. Vardi. A temporal fixpoint calculus. In *Proc. Conf. on Principles of Programming Languages, POPL'88*, pages 250–259. ACM, 1988. doi:10.1145/73560.73582.

20 M. Y. Vardi. *An Automata-Theoretic Approach to Linear Temporal Logic*, volume 1043 of *LNCS*, pages 238–266. Springer, New York, NY, USA, 1996. doi:10.1007/3-540-60915-6_6.

21 M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994. doi:10.1006/INCO.1994.1092.

Assessing the (In)Ability of LLMs to Reason in Interval Temporal Logic

Pietro Bellodi
Department of Mathematics and Computer Science, University of Ferrara, Italy

Pietro Casavecchia
Department of Mathematics and Computer Science, University of Ferrara, Italy

Alberto Paparella
Department of Mathematics and Computer Science, University of Ferrara, Italy

Guido Sciavicco
Department of Mathematics and Computer Science, University of Ferrara, Italy

Ionel Eduard Stan
Department of Informatics, Systems, and Communications, University of Milano-Bicocca, Italy

Abstract

The logical reasoning skills of Large Language Models (LLMs) is poorly understood and often overstated. Current evaluation suites rely on algebraic or commonsense puzzles that mix reasoning with symbolic manipulation and/or provide static datasets that quickly saturate or leak into pretraining corpora. In purely logical terms, the most relevant reasoning skill is the meta-mathematical task of valid formula recognition, which is at the foundation of higher-level reasoning tasks (including deduction and minimization of assertions, to name just a few). In the current landscape of LLMs benchmarking, puzzles are most often stated in propositional or first-order logic, with a few exceptions for point-based temporal logic, such as LTL; yet, in the real world, event-based temporal statements are prevalent, and they are more naturally expressed in interval-based temporal logic. Interval temporal logic offers a much richer (w.r.t. point-based temporal logic, for example) variety of problems, and not only do different languages present different expressive powers, but also the computational complexity of the validity problem can vary widely. In this paper, we tackle the problem of assessing the ability of LLMs to reason about interval-based statements in the form of validity recognition. We explore whether their accuracy is sensible to the underlying language, the computational complexity of the associated validity problem, and the intrinsic hardness of the problem in terms of formula length and modal depth of the problem. We benchmark several frontier LLMs (Gemma 3 27b It, Llama 4 Maverick, DeepSeek Chat V3 release 0324, Qwen 3 32b, and Qwen 3 235b) and show that, despite apparently impressive performance on algebraic or commonsense benchmarks, they falter on logically rigorous tasks.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Theory and algorithms for application domains

Keywords and phrases Large Language Models, Benchmarking, Interval Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.4

Supplementary Material *Software*: `https://github.com/aclai-lab/TIME2025-LLM`
archived at `swh:1:dir:de0d3840df7e24429dd48c845a7e784aa32e4da2`

Acknowledgements We acknowledge the support of the FIRD project Methodological Developments in Modal Symbolic Geometric Learning, funded by the University of Ferrara.

1 Introduction

Large Language Models (LLMs) have achieved remarkable success across a wide range of natural language tasks in recent years. Models like GPT-3 [7] and its successors demonstrated emergent capabilities in reasoning and problem-solving when prompted appropriately [33]. Notably, benchmarks such as GSM8K [10] and MATH [16] spurred progress in arithmetic

and mathematical reasoning by encouraging multi-step chain-of-thought solutions. However, the question of whether LLMs can *reliably perform logical reasoning*, in the rigorous sense of formal logic, remains underexplored and challenging [27, 25]. Logical consistency and deductive inference are critical for advanced AI reasoning, yet even state-of-the-art models often stumble on tasks requiring strict logical entailment, especially in the presence of negation or complex rules [25, 21].

There is growing evidence that current LLMs struggle with basic logical deductive questions that humans find trivial. For example, a recent evaluation called SimpleBench showed that non-expert humans significantly outperform frontier LLMs on a set of 200 straightforward reasoning questions involving spatial-temporal reasoning and logical trick questions [2]. Similarly, general intelligence tests like the Abstraction and Reasoning Corpus (ARC) [8], which requires solving abstract visual puzzles independent of prior knowledge, remain far from solved by machines, underscoring the gaps in core reasoning abilities. These observations highlight the need for systematic evaluation of LLMs' logical reasoning skills, beyond the realms of arithmetic or commonsense reasoning.

In this work we consider the problem of benchmarking the ability of LLMs to reason about temporal events and their relationships. Our approach automatically generates instances from first principles, using logic tautologies and formal inference rules, so that each example's label (valid or invalid formula) is guaranteed correct by construction. By leveraging the well-defined semantics of formal logics, we avoid the ambiguities and potential errors of human-crafted logical puzzles, providing a *reliable ground truth* for model evaluation. In our specific testbed we focus on Halpern-Shoham interval temporal logic (HS) [14], which can be considered a standard logical setting for event-based reasoning at the qualitative level. Reasoning in HS can be considered a hard problem; depending on the specific subset of operators that occur in a formula, establishing its validity status is a problem whose complexity ranges from NP-complete, to PSPACE-complete, to EXPSPACE-complete, to non primitive recursive (NPR)-complete, to undecidable. In the particular case of linear models based on the set of natural numbers, the status of every possible syntactic fragment is reported in [5].

Our approach fundamentally *differs from existing work* in three ways. First, we target *logical validity in a formal sense*, rather than numerical or symbolic manipulation. Many prior reasoning benchmarks for LLMs (e.g., math word problems in GSM8K/MATH, or code execution tasks) involve algebraic reasoning or pattern matching that, while complex, do not probe a model's ability to apply abstract logical rules or handle operators like negation and implication in a principled way. In contrast, our evaluation specifically stresses logical consistency and the handling of negation, which has been identified as a stumbling block for LLMs' reasoning [25]. Second, we focus on event-based temporal reasoning, so far neglected in the context of LLMs benchmarking. Third, we explore LLMs reasoning ability in relation to problem length (i.e., number of involved symbols), problem abstraction level (i.e., modal depth), and intrinsic problem complexity (i.e., computational class to which it belongs). Fourth, we employ an algorithmic test generator that produces valid formulas, rather than a fixed set of predetermined ones, avoiding the risk of future leaking into training datasets. We validate our approach by conducting an extensive evaluation of several leading LLMs on the generated logical reasoning tasks. In particular, we benchmark a suite of state-of-the-art models, namely Gemma 3 27b It, Llama 4 Maverick, DeepSeek Chat V3 release 0324, Qwen 3 32b, and Qwen 3 235b. Through systematic experiments on these diverse systems, we analyze their performance on problems that require genuine logical reasoning. As we show later, even the best models *struggle on logically challenging instances*, confirming findings from recent studies that current LLMs have not attained robust logical competence [21, 25, 30].

In a sense, a test of reasoning capabilities as we designed it is a test of *general intelligence* level. We can safely assume that an LLM *does not know* what interval-based temporal logic is in the strict sense (the amount of existing and available material on this topic is exponentially lower than, for example, linear time point-based temporal logic); for the test purposes such a notion is therefore *explained* to the LLM (via prompting); finally the ability of the LLM to reason about what was explained are *tested*. A positive result from a test so designed would not be a proof that a model possesses general intelligence; a negative one would be a proof that the model does not.

2 Related Work

The quest to measure and improve reasoning in LLMs has led to a variety of benchmarks targeting different reasoning facets.

On the numerical side, the Grade School Math 8K dataset (GSM8K) [10] and the MATH competition dataset [16] have become standard tests for arithmetic and mathematical problem solving. These datasets require multi-step reasoning and have spurred innovations like chain-of-thought prompting [17, 33] to elicit latent reasoning steps from models. More generally, BIG-bench and related efforts compiled diverse reasoning tasks, like commonsense and symbolic, to probe emerging capabilities of large models [28]. However, *formal logical reasoning* was not a primary focus in these early benchmarks. Notably, ARC challenge [8] targeted abstract pattern reasoning via visual puzzles to evaluate general intelligence; LLMs have struggled with ARC-style tasks unless augmented with specialized tools, reflecting a gap in out-of-distribution reasoning. Recently, the SimpleBench evaluation explicitly highlighted fundamental reasoning gaps in frontier models: on a suite of basic logic, spatial, and trick questions, human participants achieved over 80% accuracy while even top-tier LLMs (e.g., o3, Claude Sonnet, Gemini, Grok 3, and DeepSeek R1) remained far lower (30–50% range), often failing on problems requiring careful logical consistency. These findings motivate the development of dedicated logical reasoning benchmarks.

A different line of work has emerged to directly evaluate (and train) models on logical deduction tasks using controlled datasets. A pioneering example is RuleTaker [9], which generated synthetic natural language facts and rules and quizzed models on deductive conclusions. Remarkably, transformers fine-tuned on RuleTaker showed the ability to correctly answer many queries, even generalizing to some deeper inference chains, hinting at the possibility of learned logical reasoning. Subsequent efforts extended this approach: LogicNLI [31] expanded the scope to first-order logic, and ProofWriter [29] augmented the RuleTaker paradigm by asking models not only for yes/no answers but also to generate explicit natural language proofs for their conclusions. ProofWriter demonstrated that models could produce plausible step-by-step derivations for synthetic logic puzzles, though evaluation of proof correctness remained difficult. Saparov and He proposed PrOntoQA [27], another synthetic QA dataset that encodes formal deductive reasoning problems, used it to formally analyze how models reason, and founding that LLMs tend to be superficial reasoners often jumping to conclusions that are logically invalid if they appear superficially plausible. In addition to fully synthetic data, there have been expert-crafted datasets to test logical reasoning. A notable benchmark is FOLIO [15], introduced by Han et al., which consists of logically complex natural language puzzles written by experts and annotated with first-order logic forms. FOLIO problems are open-domain and diverse, covering, among others, quantifiers and implications, intended to require genuine logical deduction from the given premises. Models like GPT-3 and PaLM were reported to perform poorly on FOLIO, indicating that pretraining alone does not equip LLMs

with robust logic skills [15]. Another example is the AR-LSAT corpus [35], which contains analytical reasoning questions from law school admission tests; these are high-level logical puzzles in natural language, and zero-shot GPT-4 still finds many of them challenging [34]. One last contribution is [18], that proposes an automatic test generator (ATG), similar to our proposal, but limited to propositional logic. Overall, these works illustrate a spectrum from purely synthetic logic exercises to realistic logical reasoning problems. Across the board, *negation and multi-step inference* emerge as common pain points for current models [15, 25].

More recently, researchers have started assembling more systematic benchmarks to thoroughly probe LLMs' logical reasoning across multiple phenomena. Parmar et al. introduce LogicBench [25], a collection of 25 distinct logical reasoning patterns expressed in natural language. Each LogicBench question focuses on a single inference rule (e.g. modus ponens, modus tollens, syllogism, transitivity, etc.) either in propositional, first-order, or non-monotonic logic, presented as a small textual scenario with a yes/no question. This controlled setup allows one to pinpoint which specific forms of inference a model handles or fails. Testing GPT-4, ChatGPT, Gemini, Llama-2 and others, they found that existing LLMs do not fare well on LogicBench – especially on instances involving more complex reasoning or embedded negations, performance was near chance. Our work is closely aligned with this goal of systematic evaluation, though we approach it by generating formula-based entailment instances from formal semantics (as opposed to natural language templated questions). Another recent benchmark, LTLBench [30], specifically targets temporal logic reasoning. Tang and Belle developed a pipeline using random graph generation and an LTL model checker to create 2,000 temporal reasoning challenges, and evaluated six LLMs on them. Their results showed that while some LLMs exhibit basic competence on simple temporal queries, they struggle as the complexity increases (e.g. more events or nested temporal operators) and substantially underperform compared to what would be required for sound temporal reasoning. LTLBench demonstrates the feasibility of combining formal verification tools with LLM evaluation – an approach our work generalizes and extends to other logics. In a similar vein, Morishita et al. present the Formal Logic Deduction benchmark (FLD) [21], generated from a complete set of first-order logic deduction rules. They report that even GPT-4 solves only about half of the problems in FLD, underlining that pure logical deduction (even when posed in natural language) remains a serious challenge for LLMs.

Beyond temporal logic, reasoning about space and structured knowledge are important dimensions of logical evaluation. Spatial relation formalisms such as RCC5 and RCC8 [26] provide a calculus for qualitative spatial reasoning (e.g. relations like disjoint, overlap, and containment between regions). These have not yet been widely used to evaluate LLMs, but they present an attractive next step: one could generate spatial scenarios and queries in natural language underpinned by RCC constraints, and test if LLMs can infer implied spatial relations. We posit that our methodology can be applied here by generating spatial logic formulas with known entailments. Similarly, Description Logics (DL) underpin knowledge graphs and ontologies in the Semantic Web, enabling rigorous inference of subclass relations, instance membership, etc. [4]. Traditional AI systems employ DL reasoners (like FaCT++ or HermiT) to perform these inferences reliably. In contrast, an LLM might be used to answer ontology queries or complete a knowledge graph, but concerns arise about whether it can honor the formal logical constraints (e.g. avoid asserting mutually inconsistent facts). There is ongoing research in combining LLMs with symbolic reasoners to ensure logical consistency in knowledge-intensive applications. These neuro-symbolic approaches typically involve translating natural language to a logical form, using a logic reasoner to derive conclusions or check consistency, and then translating back to text [24, 23].

A consistent observation across these benchmarks is that negation and non-monotonic reasoning are weak spots for LLMs. For instance, LogicBench finds that models often misunderstand statements with negated conditions or conclusions [25]; similarly, PrOntoQA analysis noted that models are apt to assume a fact is true unless explicitly contradicted, even if logically it should be undetermined [27]. This tendency relates to the shallow heuristics LLMs might pick up from text, which break down for logical constructs like negation that require careful semantic interpretation. The broader implication is that purely neural models alone may lack the guarantees of logical soundness that symbolic reasoning provides. By developing benchmarks grounded in formal logic (as we do in this paper), we contribute toward bridging this gap. A robust evaluation methodology for LLMs logical reasoning is not only academically interesting but also practically vital as these models begin to be deployed in areas like legal reasoning, safety-critical decision making, and knowledge graph completion, where logical correctness is paramount. Our work specifically addresses this by including a variety of entailment cases with negated formulas and ensuring that only logically valid inferences count as correct. We thereby force models to confront the full truth-functional meaning of negation and other operators. Another aspect is combinatorial complexity: multi-step logical reasoning (combining several premises) taxes the models' limited reasoning depth and working memory. Datasets like ProofWriter and FLD explicitly vary the number of inference steps, and performance drops as steps increase [29, 21]. In our evaluation, we similarly consider entailments that may require reasoning across multiple temporal steps or combining multiple logical conditions. This allows us to examine whether models can perform reasoning beyond one-hop inference in a formal setting.

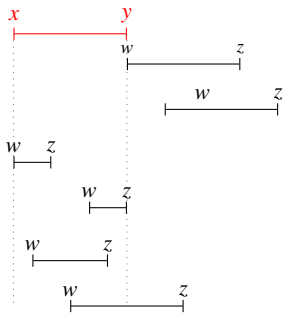
3 Interval Temporal Logic

While several different interval temporal logics have been proposed in the recent literature [13], *Halpern and Shoham's Modal Logic for Time Intervals (HS)* [14] is certainly the formalism that has received the most attention. Let $\mathbb{D} = \langle D, < \rangle$ be a linear order with *domain* D; in the following, we shall use D and $\mathbb{D}$ interchangeably. A *strict interval* over $\mathbb{D}$ is an ordered pair $[x, y]$, where $x, y \in \mathbb{D}$ and $x < y$. If we exclude the identity relation, there are 12 different binary ordering relations between two strict intervals on a linear order, often called *Allen's interval relations* [3]: the six relations R_A (*adjacent to*, also known as *after*), R_L (*later than*), R_B (*begins*, also known as *starts*), R_E (*ends*, also known as *finishes*), R_D (*during*) and R_O (*overlaps*), depicted in Tab. 1, and their *inverses*, that is, $R_{\overline{X}} = (R_X)^{-1}$, for each $X \in \{A, L, B, E, D, O\}$. We interpret interval structures as Kripke structures, with Allen's relations playing the role of accessibility relations. Thus, we associate an *existential modality* $\langle X \rangle$ with each Allen's relation R_X. Moreover, for each $X \in \{A, L, B, E, D, O\}$, the *transpose* of modality $\langle X \rangle$ is the modality $\langle \overline{X} \rangle$ corresponding to the inverse relation $R_{\overline{X}}$ of R_X. Now, let $\mathcal{X} = \{A, \overline{A}, L, \overline{L}, B, \overline{B}, E, \overline{E}, D, \overline{D}, O, \overline{O}\}$; well-formed HS formulas are built from a set of *propositional letters* $\mathcal{P}$, the classical connectives $\vee$ and $\neg$, and a modality for each Allen's interval relation, as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle X \rangle \varphi,$$

where $p \in \mathcal{P}$ and $X \in \mathcal{X}$. The other propositional connectives and constants (i.e., $\psi_1 \wedge \psi_2 \equiv \neg(\neg\psi_1 \vee \neg\psi_2), \psi_1 \rightarrow \psi_2 \equiv \neg\psi_1 \vee \psi_2$ and $\top = p \vee \neg p$), as well as, for each $X \in \mathcal{X}$, the *universal modality* $[X]$ (e.g., $[A]\varphi \equiv \neg\langle A \rangle \neg\varphi$), can be derived in the standard way. The set of all subformulas of a given HS formula φ is denoted by $sub(\varphi)$.

Table 1 Allen's interval relations and HS modalities.

HS modality	Definition w.r.t. the interval structure	Example
$\langle A \rangle$ (adjacent)	$[x,y]R_A[w,z] \Leftrightarrow y = w$	
$\langle L \rangle$ (later)	$[x,y]R_L[w,z] \Leftrightarrow y < w$	
$\langle B \rangle$ (begins)	$[x,y]R_B[w,z] \Leftrightarrow x = w \wedge z < y$	
$\langle E \rangle$ (ends)	$[x,y]R_E[w,z] \Leftrightarrow y = z \wedge x < w$	
$\langle D \rangle$ (during)	$[x,y]R_D[w,z] \Leftrightarrow x < w \wedge z < y$	
$\langle O \rangle$ (overlaps)	$[x,y]R_O[w,z] \Leftrightarrow x < w < y < z$	

The strict semantics of HS is given in terms of *interval models* of the type $M = \langle \mathbb{I}(\mathbb{D}), V \rangle$, where $\mathbb{D}$ is a linear order, $\mathbb{I}(\mathbb{D})$ is the set of all strict intervals over $\mathbb{D}$, and V is a *valuation function* $V : \mathcal{P} \to 2^{\mathbb{I}(\mathbb{D})}$ which assigns to every atomic proposition $p \in \mathcal{P}$ the set of intervals $V(p)$ on which p holds. The truth of a formula φ on a given interval $[x,y]$ in an interval model M, denoted by $M, [x,y] \Vdash \varphi$, is defined by structural induction on the complexity of formulas, as follows:

$$
\begin{array}{lll}
M, [x,y] \Vdash p & \text{if and only if} & [x,y] \in V(p), \text{ for each } p \in \mathcal{P}, \\
M, [x,y] \Vdash \neg\psi & \text{if and only if} & M, [x,y] \nVdash \psi, \\
M, [x,y] \Vdash \psi_1 \vee \psi_2 & \text{if and only if} & M, [x,y] \Vdash \psi_1 \text{ or } M, [x,y] \Vdash \psi_2, \\
M, [x,y] \Vdash \langle X \rangle \psi & \text{if and only if} & \text{there exists } [w,z] \text{ s.t. } [x,y]R_X[w,z] \text{ and } M, [w,z] \Vdash \psi,
\end{array}
$$

where $X \in \mathcal{X}$. Given a model $M = \langle \mathbb{I}(\mathbb{D}), V \rangle$ and a formula φ, we say that M *satisfies* φ if there exists an interval $[x,y] \in \mathbb{I}(\mathbb{D})$ such that $M, [x,y] \Vdash \varphi$. A formula φ is *satisfiable* if there exists an interval model that satisfies it. Moreover, a formula φ is *valid* if it is satisfiable at every interval of every (interval) model or, equivalently, if its negation $\neg\varphi$ is *unsatisfiable*.

By setting $\mathbb{D} = \mathbb{N}$, we limit our attention to interval models based on the set of natural numbers. This is not the only scenario that has been studied in the context of HS, but it is a very common one; it is the interval counterpart to the typical interpretation of LTL on the same domain. The satisfiability problem for HS is undecidable, and a great amount of effort has been devoted to the search of well-behaved syntactic fragments of it. The result of such an effort, in the case of natural numbers, is summarized in [5], and pictured in Fig. 1.

Interval temporal logic is an important tool in formal reasoning about temporal events. It is applied in several areas of artificial intelligence and machine learning (see, e.g., [19, 20], and being able to correctly reason in such a language can be of relevance. In the past, sound and complete tableau systems have been introduced in prototypical form in [6, 11, 12, 22] for variants, fragments, and generalizations of HS; however, the problem of reasoning in HS is still open in practical terms. While reasoning tasks can vary, it is known that most of them can be reduced to validity recognition, which is therefore representative of the reasoning challenges that a specific logical system poses. The question we pose is whether LLMs are able to establish if a given HS-formula is valid, and if their accuracy is sensible to the intrinsic difficulty of the problem. Such difficulty can be measured in several ways, including the length of the formula, its modal depth, and the computational complexity class to which the smallest fragment that contains the formula belongs to.

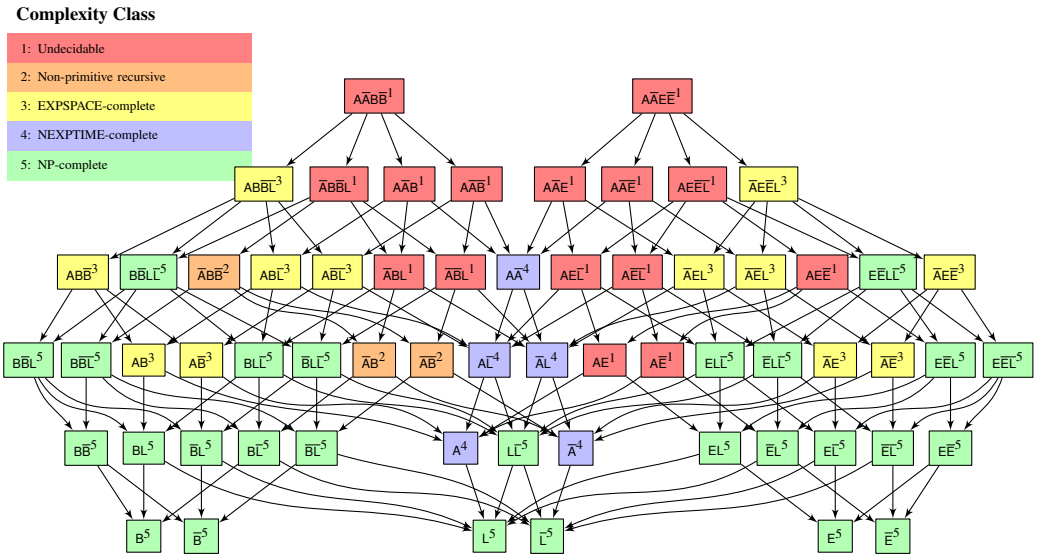


Figure 1 Relative expressive power and computational complexity of fragments of HS interpreted on models based on $\mathbb{N}$; unreported fragments are undecidable.

4 Benchmark Generation

The key point of our problem generation approach is the observation that *reasoning corresponds to validity recognition*. By their own nature, LLMs convey the idea of *natural language* reasoning, that is, the idea of reaching some logical conclusion from some set of premises. In turn, this reflects the concept of *logical* reasoning. However, while in most cases existing approaches to LLMs benchmarking relay on *common sense* logic, an automatic and systematic approach suggests the uses of *formal* logic. As a consequence, one should be easily convinced that testing reasoning capabilities corresponds to testing the ability of a system to identify a valid assertion, which is, by definition, a valid formula. The nature of LLMs to seemingly comprehend natural language should therefore not be seen as a limit, i.e., by focusing on testing common sense, natural language reasoning instances, but as an opportunity to explore their ability of following instructions, such as, given a sound and complete explanation of a chosen formal logic system, identify whether a certain reasoning is valid in it, that is, identify whether a given formula is valid. Moreover, the practice of testing and using LLMs to deal with code, such as LaTex code, programming code, Markdown, and tasking models with writing, correcting, completing, and modifying it, is now folklore. In the same spirit, the idea of testing LLMs with formal logic should be considered natural, and it should not be criticized as *unnatural*. The question we pose is: can formal reasoning tasks be carried out with distributional semantics?

Automatic theorem generation is a simply defined problem: given a set of theorems, produce a new theorem. However, it is also an ill-defined one, as it is unclear what constitutes an *interesting* theorem, especially from the point of view of its proof. While there exist attempts at solving this problem in propositional logic [18], automatic theorem generation is at its initial research stage (unlike, for example, automatic theorem proving) and, as it seems, there are no available systems for the case of modal, and in particular temporal case.

The starting point for theorem generation is existing theorems or axioms. In classic axiomatic theory, new theorems are generated by applying *sound deduction rules* to existing ones; classical deduction rules include *modus ponens*, *universal generalization*, and *uniform*

substitution. For most modal, temporal, and spatial logics that do have a sound and complete finite axiomatization, the latter is based on the above rules only; in some cases, such as that of HS, other, non-standard, rules must be added. It is well-known that Hilbert-style deduction system does not excel in producing very intuitive proofs, unlike other systems such as natural deduction (however, while Hilbert-style axiomatic systems have been studied for several logics, there exist essentially no natural deduction systems other than for propositional and first-order logics, plus some few minor exceptions). The purpose of an axiomatic system is to be able to produce a proof of a valid formula, and, as a consequence, to prove that in fact a given formula is valid, while the purpose of an automatic theorem generator is that of producing new valid formulas from existing ones, and an immediate algorithm reduces the latter to the former. In the case of HS, known validities in the language of HS come from three sources, that is, the original axiomatic system for HS [32], the axiomatic system for the fragment A$\overline{\text{A}}$ [12], and the collection of inter-definability of operators presented in [1] (examples of axioms can be seen in Tab. 2), and the process can be described as follows. Let $\mathcal{L}$ be a collection of valid HS-formulas, and $\mathcal{S}$ a collection of random well-formed HS-formulas, and apply one of the following rules:

- **i)** *uniform substitution*: choose a random validity $\varphi \in \mathcal{L}$, a random formula $\psi \in \mathcal{S}$, and a propositional letter p that occurs in φ, and produce the formula $\varphi[p/\psi]$;
- **ii)** *universal generalization*: choose a random validity $\varphi \in \mathcal{L}$ and a universal modality $[X]$, and produce the formula $[X]\varphi$;
- **iii)** *modus ponens*: choose two random validities $\varphi, \varphi \to \psi \in \mathcal{L}$, and produce the formula ψ.

▶ **Proposition 1.** *Given a set of valid HS-formulas $\mathcal{L} = \{\varphi_1, \ldots, \varphi_n\}$ and a set of well-formed HS-formulas $\mathcal{S}$, one application of the above algorithm produces a valid formula φ_{n+1}.*

The above algorithm produces valid formulas of the type $\varphi \to \psi$, with arbitrary syntactical complexity. Well-knowingly, (modal, temporal) logical formulas with Boolean semantics can be valid, if they are satisfied in every model (and world), contradictory, if they are never satisfied, or contingencies, if they are not valid nor contradictory. In this work, we focused on the ability of a LLM to distinguish between valid and contradictory formulas. In order to generate a random contradictory formula, it suffices to negate a valid one generated by the above algorithm; however, this creates a clear syntactic difference between the two classes, which may create bias towards one of the two classes. To circumvent this problem, we applied the following strategy:

- **i)** we produced a set S of valid formulas of the type $\varphi \to \psi$, randomly partitioned into two sets S_v and S_c;
- **ii)** we replaced every formula $\varphi \to \psi$ in S_v by its equivalent one $\neg\neg(\varphi \to \psi)$;
- **iii)** we replaced every formula $\varphi \to \psi$ in S_c by its opposite one $\neg(\varphi \to \psi)$;
- **iv)** finally, for every resulting formula in both S_v and S_c, we applied standard transformation rules to progressively push the negation symbols within the formula, up to a randomly chosen level.

As a result, formulas in both S_v and S_c have a non-predefined syntactical aspect, eliminating the risk of syntactic bias.

5 Results and Discussion

We approached this problem using the standard prompting techniques *context* (*ctx*), *few shots* (*fs*) [7], and *chain of thought* (*cot*) [33], combining them in a systematic way. As a form of baseline, we also prompted each model with no instructions, except the question itself; we

Table 2 Examples of axioms used to generate HS-theorems.

Axiom	Comment
all propositional validities	
$\langle A\rangle\langle A\rangle p \rightarrow \langle L\rangle p$	*definition of later*
$\langle B\rangle\langle E\rangle p \leftrightarrow \langle D\rangle p$	*definition of during*
$\langle \overline{B}\rangle\langle \overline{E}\rangle p \leftrightarrow \langle \overline{D}\rangle p$	*definition of during*
$\langle B\rangle\langle B\rangle p \leftrightarrow \langle B\rangle p$	*transitivity of starts*
$\langle A\rangle\langle A\rangle\langle A\rangle p \leftrightarrow \langle A\rangle\langle A\rangle p$	*pseudo-transitivity of meets*
$\langle B\rangle\langle E\rangle p \leftrightarrow \langle E\rangle\langle B\rangle p$	*commutativity of starts/finishes*

refer to this technique as *barebone*; on the other hand, *chain of thought*, *few shots*, and *context* are combined in the 8 possible ways, whereas the minimal configuration corresponding to a context without instructions is referred to as *base*, obtaining, in the end, 9 different prompts per single problem.

Taken individually, the prompts we used are as follows. The *barebone* baseline:

Given an interval temporal logic formula in the language of Halpern and Shoham's Modal Logic of Allen's Relations, reply with uppercase "[VALID]" if the formula is valid or uppercase "[INVALID]" if it is not.

Then, we designed the following *context*, structured, in turn, into the sections *purpose, syntax, semantics, task,* and *objective*:

*## **Purpose**
HS is a formal system for reasoning about interval-based events on a linear model based on the natural numbers. This context will define HS's syntax and semantics. The ultimate goal is to check if a HS formula is logically valid.
Syntax of HS
Propositional Letters
Let AP be a countable set of atomic propositions (p, q, r, ...), representing basic facts.
Well-Formed Formulas (wffs)
HS formulas are built inductively:*

*- **Base case**: Every p in AP is a wff.
- **Inductive cases**: If φ and ψ are wffs, then so are:*
...
*## **Semantics over Infinite Traces** Formulas of HS are interpreted over interval models based on the natural numbers N. Define I(N) as the set of all intervals [x,y] where x and y are natural numbers and x<y, and V as a function that assigns to each interval [x,y], the subset of AP of all and only propositional letters that are true on [x,y]. A model M is a pair (I(N),V). The satisfaction relation ** M,[x,y]|= φ ** for a model M and an interval [x,y] is defined by induction on the formula:*

*- **Atomic Propositions:**
- M,[x,y] |= p if and only if p belongs to V([x,y]), for all atomic propositions p in AP.
- **Boolean Operators:***
...
*## **Task: Evaluate HS Formula Validity***

Table 3 Overall accuracy in positive (TP) and negative (TN) cases, and overall average accuracy (AC) per model and prompt configuration.

	Gemma 3 27b It			Llama 4 Maverick			DeepSeek Chat V3			Qwen 3 32b			Qwen 3 235b		
	TP	TN	AC	TP	TN	AC	TP	TN	AC	TP	TN	AC	TP	TN	AC
barebone	0.17	0.92	0.55	0.69	0.72	0.71	0.39	0.92	0.65	0.41	0.93	0.67	0.50	0.72	0.61
base	0.07	0.97	0.52	0.35	0.91	0.62	0.04	1.00	0.52	0.12	0.96	0.54	0.51	0.73	0.62
ctx	0.09	0.96	0.52	0.47	0.78	0.62	0.05	1.00	0.53	0.09	0.98	0.54	0.34	0.88	0.61
cot	0.20	0.97	0.58	0.47	0.86	0.67	0.55	0.91	0.73	0.41	0.96	0.69	0.45	0.94	0.69
fs	0.48	0.80	0.64	0.83	0.73	0.77	0.53	0.86	0.69	0.71	0.55	0.63	0.80	0.60	0.70
ctx+cot	0.19	0.95	0.57	0.54	0.83	0.69	0.54	0.92	0.73	0.43	0.97	0.70	0.48	0.94	0.71
ctx+fs	0.48	0.68	0.58	0.50	0.79	0.65	0.58	0.86	0.72	0.45	0.75	0.60	0.75	0.66	0.70
cot+fs	0.26	0.94	0.60	0.87	0.78	0.82	0.58	0.84	0.71	0.48	0.93	0.71	0.49	0.88	0.68
ctx+cot+fs	0.30	0.92	0.61	0.83	0.78	0.81	0.64	0.85	0.75	0.47	0.93	0.71	0.61	0.84	0.72
average	0.25	0.90	0.57	0.62	0.80	0.71	0.43	0.91	0.67	0.40	0.88	0.64	0.55	0.80	0.67

*### **Objective**
Determine whether the formula is valid using HS semantics and reasoning. The formula can be written using symbols for atomic propositions (e.g., p, q, r, ...), negation operator (i.e., !), conjunction operator (i.e., &), ...*

The *objective* section when we prompted the models without chain of thought has the following structure:

*### **Instructions**
Reply **only** with uppercase "[VALID]" if the formula is valid or uppercase "[INVALID]" if it is not. **Do not explain your reasoning**.*

When using *chain of thought* the latter becomes:

*### **Instructions**
Follow these steps rigorously:
1. **Parse the Formulas**: Identify operators and subformulas.
2. **Apply Semantics**: Check if the formula necessarily holds in all infinite traces.
3. **Construct Proof/Counterexample**:
- If valid: Provide a **step-by-step proof** showing an argument for validity.
- If invalid: Build a **concrete model** M and identify an interval on it where the formula does not hold.
4. **Conclude**: Answer with uppercase "[VALID]" if the formula is valid or uppercase "[INVALID]" if it is not. No other responses are allowed.*

When *few shots* are used, three positive examples and three negative examples are extracted from a pool of pre-determined positive and negative examples containing 600 formulas, 300 of which are valid while the remaining ones are not, and randomly rotated for each individual problem.

We generated 1000 valid instances and 1000 non-valid ones, with length up to 139 symbols and modal depths up to 11, and submitted them in each of the 9 prompt configurations to each of the models. We used the following providers: DeepInfra, for Gemma 3 27b It, Qwen 3 32b, and Qwen 3 235b, NovitaAI for Gemma 3 27b It, and CentML for Llama 4.

The overview of the overall accuracies per model and per prompt configuration is reported in Tab. 3. The performances of each model and prompt configuration across progressively longer and progressively modally more complex is shown in Fig. 2. The first important

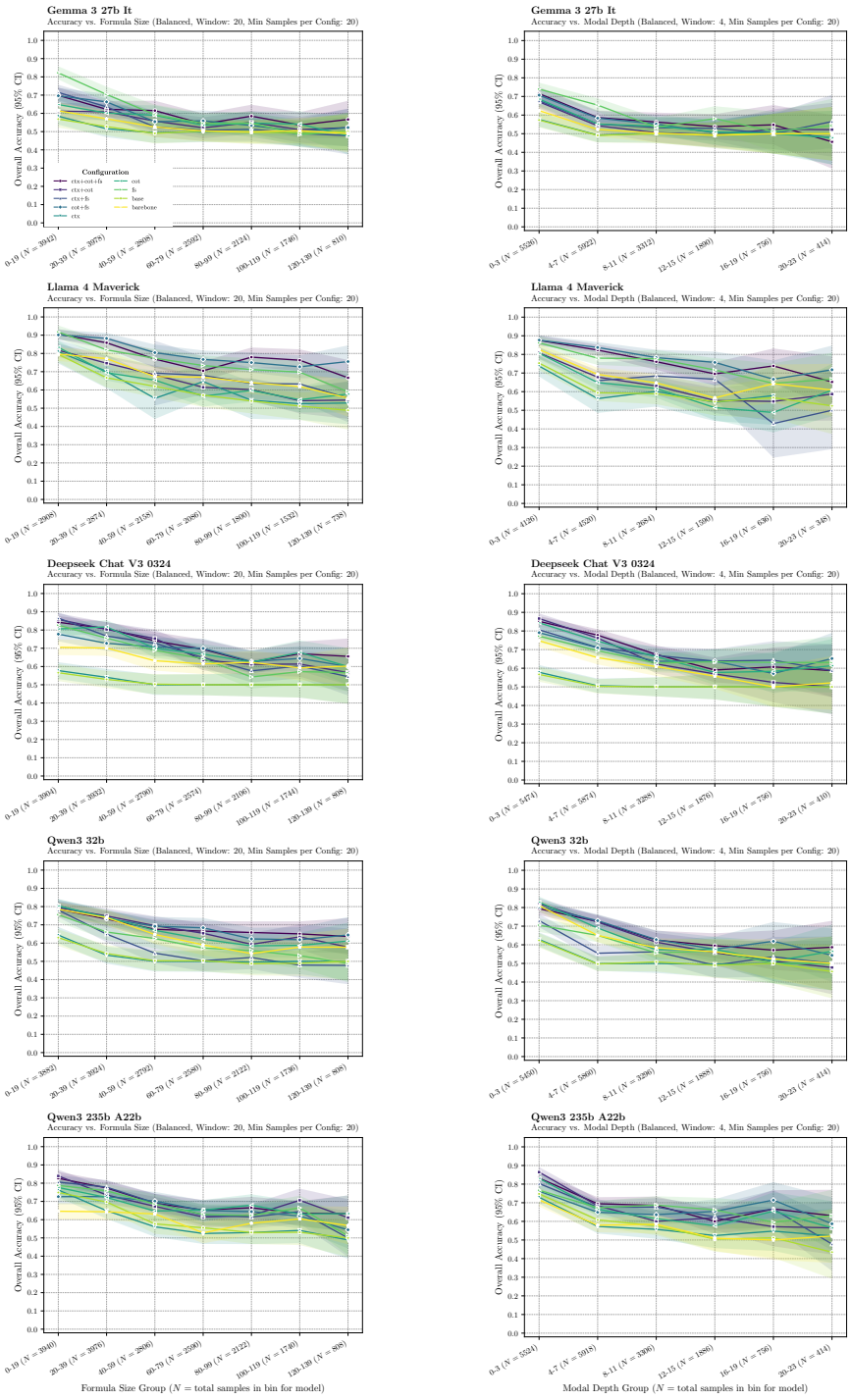


Figure 2 Accuracy per model, prompt configuration, and difficulty level, in terms of formula length (left hand side) and modal depth (right hand side).

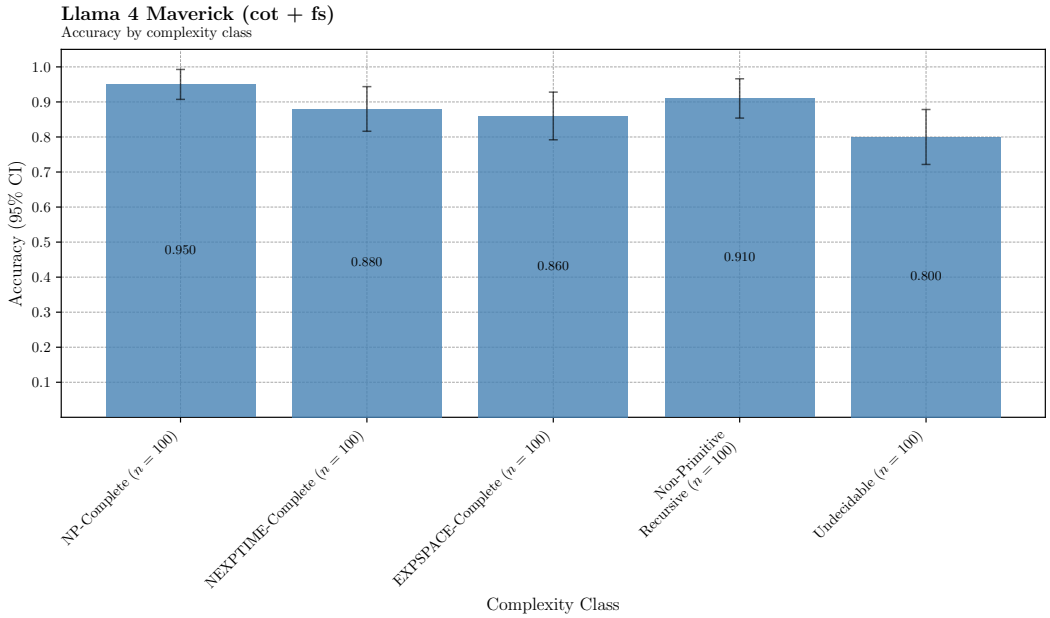


Figure 3 Accuracy of Llama 4 in the *ctx+fs* configuration for different complexity classes.

observation that can be drawn is that no model and no configuration reached more than 0.82 in overall accuracy; such value was achieved by Llama 4 in the *cot+fs* configuration. The overall accuracy in other models and configuration varies in a quite wide range, with a lower end of 0.52, which is essentially equivalent to the random answer. The models have behaved in very different ways to the different configurations. Those that have a lower overall accuracy across configurations, such as Gemma 3 27b It, seem to react positively to the progressively more detailed and precise information that is prompted from the *base* configuration up to the *ctx+cot+fs* one, although the improvement does not seem to be always linear. On the other hand, the ones with higher overall accuracy across configurations, such as Llama 4, seem not be too influenced by the type of prompt; Qwen 3 32b, in particular, presents an accuracy between 0.54 and 0.71 in all configurations, including *barebone*, indicating a close-to-null response from the instructions. All models have a predetermined strong bias towards answering that a formula is not valid (which in absolute terms is the most probable status of a random formula); DeepSeek Chat V3 showed the strongest bias: in two configurations, *ctx* and *base*, returned a true negative rate of 1.00, balanced by a true positive rate of 0.04 and 0.05, respectively. Adding *few shots* to the prompt, in general, slightly improves the results in almost every model.

Let us now analyse of the performances from the point of view of the intrinsic difficulty of the problem. The most evident phenomenon is the variability of the performances compared to the increasing hardness of the problem. Models, in general, exhibit the expected decrease of accuracy proportional to the length of the problem or its modal depth, but such decrease is not always clear. Thus, in some cases the worst performances do not correspond to the most difficult problems, such as in the case of Llama 4 Maverick and Qwen 3 235b, for several configurations.

Finally, in Fig. 3 we can see the result of a further experiment to assess the relationship between the ability of LLMs for interval temporal logic reasoning and the hardness of the problem in terms of computational complexity of the fragment that contains a formula. We

considered the top performing model(Llama 4) in the top performing configuration (*ctx+cot*), and devised a small dataset of 500 (small) formulas, 100 for each different computational class. As it can be seen, essentially no difference arises, despite the fact that the computational problem underlying such questions varies very much. The generally high performance is most probably due to formulas being short and with a low modal depth.

6 Conclusions

In this paper we considered the problem of benchmarking Large Language Models on their ability for formal logical reasoning, specifically interval temporal logical reasoning. Our results seem to indicate, quite reasonably, that statistical tools may not be the right solution for logical tasks; the fact that such tools are sometimes presented as representative of general intelligence, as well as the resonance that they have received in the recent past contributes to this confusion.

The high variability, the generally low accuracy, but most importantly the lack of consistency of the results is a clear indication of the unreliability of LLMs to perform logical reasoning on unseen problems. It is however of notice that some of models tested on our benchmark were capable, at least in some configurations, to correctly identify several valid and invalid formulas despite their high syntactical complexity, even if the tokenizer often produces syntactic mistakes such as merging double symbols (e.g., negation), useful for natural language but detrimental in this scenario.

References

1 L. Aceto, D. Della Monica, A. Ingólfsdóttir, A. Montanari, and G. Sciavicco. On the expressiveness of the interval logic of allen's relations over finite and discrete linear orders. In *Proc. of the 14th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 8761 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2014. doi:10.1007/978-3-319-11558-0_19.

2 AI Insiders. Simple bench. https://simple-bench.com, 2024.

3 J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.

4 F. Baader, D. Calvanese, D.L. McGuinness, and *others*, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

5 D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theor. Comput. Sci.*, 560:269–291, 2014. doi:10.1016/J.TCS.2014.03.033.

6 D. Bresolin, D. Della Monica, A. Montanari, and G. Sciavicco. A tableau system for right propositional neighborhood logic over finite linear orders: An implementation. In *Proc. of the 22th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 8123 of *LNCS*, pages 74–80. Springer, 2013. doi:10.1007/978-3-642-40537-2_8.

7 T.B. Brown, B. Mann, N. Ryder, and *others*. Language models are few-shot learners. In *Proc. of the 33rd Annual Confernce on Advances in Neural Information Processing Systems*, pages 1–25, 2020.

8 F. Chollet. On the measure of intelligence. *CoRR*, abs/1911.01547, 2019. arXiv:1911.01547.

9 P. Clark, O. Tafjord, and K. Richardson. Transformers as soft reasoners over language. In *Proc. of the 29th International Joint Conference on Artificial Intelligence*, pages 3882–3890, 2020.

10 K. Cobbe, V. Kosaraju, M. Bavarian, and *others*. Training verifiers to solve math word problems, 2021. arXiv:2110.14168.

11 V. Goranko, A. Montanari, P. Sala, and G. Sciavicco. A general tableau method for propositional interval temporal logics: Theory and implementation. *Journal of Applied Logics*, 4(3):305–330, 2006. doi:10.1016/J.JAL.2005.06.012.

12 V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood temporal logics. *Journal of Universal Computer Science*, 9(9):1137–1167, 2003. doi:10.3217/JUCS-009-09-1137.

13 V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal logics and duration calculi. *Journal of Applied Non-Classical Logics*, 14(1–2):9–54, 2004. doi:10.3166/JANCL.14.9-54.

14 Joseph Y. Halperns and Yoav Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991. doi:10.1145/115234.115351.

15 S. Han, H. Schoelkopf, Y. Zhao, and *others*. FOLIO: natural language reasoning with first-order logic. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031, 2024.

16 D. Hendrycks, C. Burns, S. Kadavath, and *others*. Measuring mathematical problem solving with the MATH dataset. In *Proc. of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks*, 2021.

17 T. Kojima, S. Shane Gu, M. Reid, and *others*. Large language models are zero-shot reasoners. In *Proc. of the 35th Annual Conference on Advances in Neural Information Processing Systems*, pages 1–15, 2022.

18 X. Lin, Q. Cao, Y. Huang, and *others*. ATG: benchmarking automated theorem generation for generative language models. In *Findings of the Association for Computational Linguistics*, pages 4465–4480. Association for Computational Linguistics, 2024. doi:10.18653/V1/2024.FINDINGS-NAACL.279.

19 E. Lucena-Sánchez, G. Sciavicco, and I.E Stan. Feature and language selection in temporal symbolic regression for interpretable air quality modelling. *Algorithms*, 14(3):76, 2021. doi:10.3390/A14030076.

20 F. Manzella, G. Pagliarini, G. Sciavicco, and I.E. Stan. The voice of COVID-19: breath and cough recording classification with temporal decision trees and random forests. *Artificial Intelligence in Medicine*, 137:102486, 2023. doi:10.1016/J.ARTMED.2022.102486.

21 T. Morishita, G. Morio, A. Yamaguchi, and *others*. Learning deductive reasoning from synthetic corpus based on formal logic. In *Proc. of the International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 25254–25274, 2023. URL: https://proceedings.mlr.press/v202/morishita23a.html.

22 E. Muñoz-Velasco, M. Pelegrín-Garcí, P. Sala, G. Sciavicco, and I. E. Stan. On coarser interval temporal logics. *Artificial Intelligence*, 266:1–26, 2019. doi:10.1016/J.ARTINT.2018.09.001.

23 T. Olausson, A. Gu, B. Lipkin, and *others*. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, pages 5153–5176. Association for Computational Linguistics, 2023. doi:10.18653/V1/2023.EMNLP-MAIN.313.

24 L. Pan, A. Albalak, X. Wang, and *others*. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics*, pages 3806–3824. Association for Computational Linguistics, 2023. doi:10.18653/V1/2023.FINDINGS-EMNLP.248.

25 M. Parmar, N. Patel, N. Varshney, and *others*. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 13679–13707, 2024.

26 D.A. Randell, Z. Cui, and A.G. Cohn. A spatial logic based on regions and connection. In *Proc. of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufmann, 1992.

27 A. Saparov and H: He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *Proc. of the 11th International Conference on Learning Representations*, 2023.

28 A. Srivastava, A. Rastogi, A. Rao, and *others*. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

29 O. Tafjord, B. Dalvi, and P. Clark. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics*, pages 3621–3634, 2021.

30 W. Tang and V. Belle. LTLBench: Towards benchmarks for evaluating temporal logic reasoning in large language models. *CoRR*, abs/2407.05434, 2024. doi:10.48550/arXiv.2407.05434.

31 J Tian, Y Li, W. Chen, and *others*. Diagnosing the first-order logical reasoning ability through logicNLI. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, pages 3738–3747, 2021.

32 Y. Venema. Expressiveness and completeness of an interval tense logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990. doi:10.1305/NDJFL/1093635589.

33 J. Wei, X. Wang, D. Schuurmans, and *others*. Chain-of-thought prompting elicits reasoning in large language models. In *Proc. of the 35th Annual Conference on Advances in Neural Information Processing Systems*, pages 1–14, 2022.

34 W. Zhong, R. Cui, Y. Guo, and *others*. AGIEval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics*, pages 2299–2314. Association for Computational Linguistics, 2024. doi:10.18653/V1/2024.FINDINGS-NAACL.149.

35 W. Zhong, S. Wang, D. Tang, and *others*. Analytical reasoning of text. In *Findings of the Association for Computational Linguistics*, pages 2306–2319, 2022.

Higher-Order Timed Automata and Tail Recursion

Florian Bruse
TUM School of Computation, Information and Technology,
Technical University of Munich, Munich, Germany

Abstract

Timed Automata (TA) are a popular formalism to model systems in dense linear time. However, due to their finite state-space, they can only model systems with a finitary logical behavior. There are extensions to e.g., timed pushdown systems and timed recursive state machines. Higher-Order Recursion Schemes (HORS) are another popular model for infinite-state, non-regular systems, naturally stratified by their type-theoretic order. We recently introduced Real-Time Recursion schemes as an approximation of HORS to real-time systems.

This paper updates Real-Time Recursion Schemes into Higher-Order Timed Automata, a formalism that defines a tree-shaped timed automaton, which is more suitable to model actual systems. We show that the model-checking problem against the timed version of the modal mu-calculus exhibits the expected complexity bounds, i.e., an increase by one exponential towards the untimed version. We also show that, in the presence of tail recursion, half an exponential can be recovered, mirroring similar gains in the untimed setting. We also give a matching lower bound for the special case of order-1 HORTA. We conjecture that this can be generalized for all orders.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Tree languages; Theory of computation → Automata over infinite objects

Keywords and phrases Timed Automata, Higher-Order Recursion Schemes, Tree Automata, Tail Recursion

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.5

1 Introduction

Verification of real-time systems plays an ever important role, as more and more aspects of life become computerized. Often, systems are required to not only respond to certain input eventually, but within specific time bounds. For example, electronic stability control or anti-lock braking systems need to respond within fractions of a second.

Timed Automata (TA) [5] are a popular tool to model real-time systems [1, 16]. TA model time not in a discrete fashion, as used in e.g., CTL or the modal μ-calculus, but rather in a dense linear fashion. The well-known region abstraction allows us to recover finiteness by re-discretization of said dense linear time, which yields decidability (and PSPACE-completeness) of e.g., the model-checking problem for the temporal version of CTL, TCTL [5]. A less well-known real-time logic used in conjunction with timed automata is the timed μ-calculus (T_μ) [17], for which model-checking is EXPTIME-complete.

There are several attempts to generalize the notion of timed automata beyond a finite state space, for example timed recursive state machines [7], recursive timed automata [22], or timed pushdown automata [8, 13]. Recently, we proposed *Real-Time Recursion Schemes* [3], an extension of the notion of timed automata to Higher-Order Recursion Schemes (HORS). HORS are a well-studied framework in the context of infinite-state verification [14, 18]. A HORS is a higher-order grammar that generates a tree, e.g., the syntax tree of a functional program. The question of HORS model-checking is to decide whether a given alternating parity tree-automaton (APT) accepts the tree generated by the HORS. This is known to be decidable for order-k HORS in k-EXPTIME [20, 19]. We showed in [3] that mixing HORS and real-time systems had the typical effect of increasing the complexity of the model-checking problem by one exponential as compared to the untimed base problem.

This paper extends the idea of Real-Time Recursion Schemes into a proper recursive grammar for timed automata called Higher-Order Timed Automata (HORTA), which makes the problem more natural and rather intuitive to handle. We show that the model-checking problem remains in $(k+1)$-EXPTIME for the updated problem, which asks whether the timed transition system defined by a HORTA satisfies a given formula of the timed μ-calculus.

Moreover, we connect the new formalism to existing research in the area of *tail recursion*. Higher-order model-checking problems like the HORS model-checking problem often have high theoretical complexities, and passing to tail-recursive fragments can shave off half an exponential, not only for recursion schemes [9], but also for model-checking problems where the logic is higher-order [12, 11]. We confirm that these findings also hold for HORTA and the timed μ-calculus, i.e., that the model-checking problem becomes easier by half an exponential in the tail-recursive setting. We establish a matching lower bound which, for space considerations, is given for the order-1-case only.

The paper is structured as follows: in Sect. 2, we introduce APT, TA, the timed μ-calculus, HORS, and existing results regarding tail recursion. In Sect. 3 we define HORTA, in Sect. 4 we define the tail-recursive fragment of the HORTA model-checking problem. In Sect. 5, we establish upper bounds for the model-checking problem, and give a matching lower bound for the order-1 case in Sect. 6. We conclude in Sect. 7. Most proofs have been omitted due to space considerations.

2 Preliminaries

2.1 Trees and Automata

A tree is a finite, left-closed set $T \subseteq \mathbb{N}^*$, i.e., for all $vi \in T$, we have $v \in T$ and, moreover, $vi-1 \in T$ if $i > 0$. A *tree alphabet* is a finite, nonempty set Σ and a function $ar\colon \Sigma \to \mathbb{N}$ indicating the arity of each symbol. We write Σ_i for the set of symbols of arity i in Σ. A Σ-tree is a pair (T, l) of a tree T and a labeling function $l\colon T \mapsto \Sigma$, such that each $v \in T$ has exactly $ar(l(v))$ successors. We often identify a tree with its labeling function.

Let S be a set. $\mathcal{B}^+(S)$ is the set of *positive Boolean expressions* over S, derived from the grammar $\varphi \coloneqq s \mid \bot \mid \top \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$ with $s \in S$.

An *alternating parity tree-automaton* (APT) is a $\mathcal{P} = (Q, \Sigma, \delta, q_I, \Lambda)$ where Q is a finite, nonempty set of *states*, Σ is a tree alphabet containing a special nullary symbol ω, $\delta = \bigcup_{i \leq n} \delta^i$ is the *transition function* with $\delta^i\colon Q \times \Sigma_i \to \mathcal{B}^+(Q^i)$ and n being the maximal arity that occurs in Σ. We write $\delta(q, l(v))$ for $\delta^i(q, l(v))$ if $ar(l(v)) = i$. Finally, $q_I \in Q$ is the *starting state* and $\Lambda\colon Q \to \mathbb{N}$ is the *priority function*. The *size* of an APT is the number of its states.

A *run* of an APT $\mathcal{P}$ on some Σ-tree T (for matching Σ) is a parity game $G(\mathcal{P}, T)$ between $\exists$ and $\forall$, called the *acceptance game*. It has positions from $T \times (Q \cup \bigcup_{i \leq n} \mathcal{B}^+(Q^i))$ where n is the maximal arity in Σ. Its starting position is (ϵ, q_I). In a position of the form (v, q), the unique successor is $(v, \delta(q, l(v)))$. Positions of the forms $(v, \varphi_1 \vee \varphi_2)$ and $(v, \varphi_1 \wedge \varphi_2)$ have two successors, namely (v, φ_1) and (v, φ_2). A position of the form $(v, (q_0, \ldots, q_{i-1}))$ has i successors $(v0, q_0), \ldots, (vi-1, q_{i-1})$, a position of the form $(v, \top)$ or $(v, \bot)$ has no successors. Positions of the forms $(v, \top)$ and $(v, \varphi_1 \wedge \varphi_2)$ and $(v, (q_0, \ldots, q_{i-1}))$ belong to $\forall$, all other positions belong to $\exists$. Finally, the priorities of the game are $\Omega(v, q) = \Lambda(q)$; for all other positions we have $\Omega(v, \varphi) = 1$. $\mathcal{P}$ *accepts* a tree T if $\exists$ wins $G(\mathcal{P}, T)$. In the following, we assume APT to have designated states $q_\top, q_\bot$ from which all, resp. no trees are accepted.

2.2 Higher-Order Recursion Schemes

Instead of the classical definition of Higher-Order Recursion Schemes (HORS, cf. e.g., [18]), we use a version going back to Damm [14] and re-introduced by Walukiewicz and Salvati [21], using the λY-calculus as syntax.

Types and Terms. The set of *types* is defined inductively via $\tau ::= \bullet \mid \tau \to \tau$ where $\bullet$ is the type of trees, $\bullet \to \bullet$ is the type of functions that consume a tree and yield a tree, etc. We write $\tau^i \to \bullet$ for the type $\tau \to \cdots \tau \to \bullet$ with i many copies of τ. The *order* of a type is defined via $ord(\bullet) = 0$ and $ord(\tau_1 \to \tau_2) = \max\{1 + ord(\tau_1), ord(\tau_2)\}$.

Let Σ be a tree alphabet. Each $a \in \Sigma$ of arity i is a *tree constructor* of type $\bullet^i \to \bullet$. Let $\mathcal{L}$ be a set of typed λ *variables*, $\mathcal{F}$ be a set of typed Y *variables*. Lower case letters $a, b, \dots$ denote tree constructors, lower case letters $x, y, \dots$ denote λ variables, upper case letters $F, G, \dots$ denote Y variables. If necessary, $(x\colon \tau)$, $(F\colon \tau)$ indicates the type of a variable.

Given $\Sigma, \mathcal{L}, \mathcal{F}$, the set of λY terms is defined inductively: a tree constructor of type $\bullet^i \to \bullet$ is a term of type $\bullet^i \to \bullet$, a lambda variable $x\colon \tau$ and a Y variable $F\colon \tau$ are terms of type τ. Given $x\colon \tau_1$ and a term t of type τ_2, $\lambda(x\colon \tau).\ t$ is a term of type $\tau_1 \to \tau_2$. Given terms t_1, t_2 of type $\tau_2 \to \tau_1$ and τ_2, resp., $(t_1\ t_2)$ is a term of type τ_1. If $F\colon \tau$ is a Y variable and t is a term of type τ, then $Y(F\colon \tau).\ t$ is a term of type τ.

$t[t'/x]$ denotes capture-avoiding substitution of t' for all free occurrences of x in t, and similarly for $t[t'/F]$, assuming that the types match. Each variable may also be bound at most once in a term. Hence, for closed terms there is a function *term* that maps each Y variable F to $term(F)$, defined as t if the unique binding of F is $YF.\ t$. The *order* of a term is that of the highest order of any of its subterms, its size the number of its distinct subterms.

Semantics. Besides α-conversion, i.e., variable renaming, the λY calculus has β-reduction and δ-reduction. β-reduction $\to_\beta$ reduces $((\lambda x.\ t)\ t')$, where x and t' have the same type, to $t[t'/x]$. δ-reduction $\to_\delta$ reduces $YF.\ t$ to t and F to $term(F)$. The reflexive transitive closure of $\to_\beta \cup \to_\delta$ is $\to^*_{\beta\delta}$. Since both β-reduction and δ-reduction maintain the type of a term, so does $\to^*_{\beta\delta}$, and this relation is confluent. A closed term of type $\bullet$ is in *weak head normal form* if it is of the form $a\ t_0 \cdots t_j$. Not every λY term reduces to a head normal form.

Let Σ be a tree signature and let ω be a new nullary symbol, i.e, of type $\bullet$. We define the *Böhm tree* $BT(t)$ of a closed term of type $\bullet$ as follows: If t does not reduce via $\to^*_{\beta\delta}$ to a weak head normal form, then $BT(t) = \omega$, i.e., the tree has just one node. Otherwise, let $a\ t_0 \cdots t_j$ be a weak head normal form of t. The root of $BT(t)$ is labeled by a, and its successors are, in order, the roots of $BT(t_0), \dots, BT(t_j)$. Due to confluence, this definition is sound.

The HORS model-checking problem (defined via λY terms) is the following: Given a closed term t of type $\bullet$ over Σ and an APT $\mathcal{A}$ with alphabet $\Sigma \cup \{\omega\colon \bullet\}$, does $\mathcal{A}$ accept $BT(t)$? For terms of order $k \geq 0$, this is known to be a k-EXPTIME complete problem [20].

2.3 Timed Automata

Let $\mathcal{X} = \{\mathtt{x}, \mathtt{y}, \dots\}$ be a set of $\mathbb{R}^{\geq 0}$-valued variables, called *clocks*. $CC(\mathcal{X})$ is the set of *clock constraints* over $\mathcal{X}$, defined as conjunctive formulas over $\top$ and $\mathtt{x} \oplus c$ for $\mathtt{x} \in \mathcal{X}$ and $c \in \mathbb{N}$, where $\oplus \in \{\leq, <, >, \geq\}$. Clock constraints are denoted by χ, χ' etc.

A clock evaluation is a mapping $\eta\colon \mathcal{X} \to \mathbb{R}^{\geq 0}$; it satisfies a clock constraint as follows: (I) $\eta \models \top$ always, (II) $\eta \models \mathtt{x} \oplus c$ iff $\eta(\mathtt{x}) \oplus c$ and (III) $\eta \models \varphi_1 \wedge \varphi_2$ iff $\eta \models \varphi_1$ and $\eta \models \varphi_2$. For a clock evaluation η and $d \in \mathbb{R}^{\geq 0}$, we write $\eta{+}d$ for the clock evaluation defined via $(\eta{+}d)(\mathtt{x}) = \eta(\mathtt{x}) + d$ for all $\mathtt{x} \in \mathcal{X}$. For $R \subseteq \mathcal{X}$, $\eta|_R$ is the clock evaluation defined via $\eta|_R(\mathtt{x}) = \eta(\mathtt{x})$ if $x \notin R$ and $\eta|_R(\mathtt{x}) = 0$ if $\mathtt{x} \in R$.

Let *Prop* be a finite set of *propositions*. A *timed automaton* over clocks in $\mathcal{X}$ and with propositions in *Prop* is an $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \Delta, \lambda)$ where (I) L is the set of *locations* of the timed automaton, including the *initial location* ℓ_0, (II) $\mathcal{X}$ is a finite set of clocks, (III) $\iota\colon L \to CC(\mathcal{X})$ assigns a clock constraint called an *invariant* to each location, (IV) $\Delta \subseteq L \times CC(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is a finite set of transitions; we write $\ell \xrightarrow{g,R} \ell'$ for $(\ell, g, R, \ell') \in \delta$. In such a transition, g is the *guard* and R are the *resets* of the transition. Finally, (V) $\lambda\colon L \to 2^{Prop}$ labels each location with the propositions valid there. The *index* $m(\mathcal{A})$ of a timed automaton $\mathcal{A}$ is the largest constant that occurs in its guards or invariants. Its size is defined as

$$|\mathcal{A}| = |\Delta| \cdot (2 \cdot \log(|L|) + |\mathcal{X}|^2 \cdot \log m(\mathcal{A})) + |L| \cdot (\log |\mathcal{X}|^2 \cdot \log m(\mathcal{A})) + |L| \cdot |Prop|.$$

due to the coding of the constants in clock constraints in binary. A TA defines a timed Transition System (tTS), to be defined in more general fashion in Sect. 3.

2.4 The Timed μ-Calculus

We define the *timed modal μ-calculus* (T_μ) following [17]. Let $\mathcal{X}$ be a set of clocks and let $\mathcal{Y}$ disjoint from $\mathcal{X}$ be a set of *specification clocks*. Let *Prop* be a set of propositions and let $\mathcal{V}$ be a set of *fixpoint variables*. A formula of T_μ is one derived from the following grammar:

$$\varphi ::= p \mid X \mid \chi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \rhd \varphi \mid \varphi \boxminus \varphi \mid \mathtt{z}.\ \varphi \mid \mu X.\ \varphi \mid \nu X.\ \varphi$$

where $p \in Prop, X \in \mathcal{V}, \chi \in CC(\mathcal{X} \cup \mathcal{Y})$ and $\mathtt{z} \in \mathcal{Y}$. Moreover, every variable X must be bound exactly once and occur under an even number of negations in $\mu X.\ \varphi$ or $\nu X.\ \varphi$, a standard convention for the modal μ-calculus. This induces, for each T_μ formula φ, a function *fp* where $fp(X)$ is the unique ψ with $\mu X.\ \psi$ or $\nu X.\ \psi$ a subformula of φ. Moreover, we assume w.l.o.g. that every T_μ formula is *guarded* in the sense that, on the path from $\mu X.\ \psi$ or $\nu X.\ \psi$ to X in the syntax tree of the formula, there is an instance of $\rhd$ or $\boxminus$. Analogously to the ordinary μ-calculus [10], every T_μ formula can be converted into an equivalent guarded one, potentially at the cost of exponential blowup[1].

$\mathtt{z}.\ \varphi$ resets the specification clock $\mathtt{z}$ to 0. Hence, patterns like $\mathtt{z}.\ \ldots \mathtt{z} = 4 \ldots$ serve to test for elapsed time. The intuition for $\varphi_1 \rhd \varphi_2$ is that of a *temporal next* operator: it is satisfied if φ_2 is true after a sequence of a delay, a discrete transition, and another delay, and φ_1 is true all the way before. The operator $\boxminus$ is the dual of it, i.e., a temporal next with universal quantification over delays and transitions.

Let $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$ be a tTS, let $\alpha\colon \mathcal{V} \to 2^{\mathcal{S}}$ be a *variable assignment*. The semantics $[\![\varphi]\!]^\alpha_{\mathcal{T}}$ of a T_μ formula φ is defined inductively via

$$[\![p]\!]^\alpha_{\mathcal{T}} = \{s \in \mathcal{S} \mid p \in \lambda(s)\} \qquad [\![X]\!]^\alpha_{\mathcal{T}} = \alpha(X)$$
$$[\![\chi]\!]^\alpha_{\mathcal{T}} = \{s \in \mathcal{S} \mid s \models \chi\} \qquad [\![\mathtt{z}.\ \psi]\!]^\alpha_{\mathcal{T}} = \{s \in \mathcal{S} \mid s|_{\{\mathtt{z}\}} \in [\![\psi]\!]^\alpha_{\mathcal{T}}\}$$
$$[\![\psi_1 \vee \psi_2]\!]^\alpha_{\mathcal{T}} = [\![\psi_1]\!]^\alpha_{\mathcal{T}} \cup [\![\psi_2]\!]^\alpha_{\mathcal{T}} \qquad [\![\psi_1 \wedge \psi_2]\!]^\alpha_{\mathcal{T}} = [\![\psi_1]\!]^\alpha_{\mathcal{T}} \cap [\![\psi_2]\!]^\alpha_{\mathcal{T}}$$
$$[\![\neg\psi]\!]^\alpha_{\mathcal{T}} = \mathcal{S} \setminus [\![\psi]\!]^\alpha_{\mathcal{T}} \qquad [\![\psi_1 \boxminus \psi_2]\!]^\alpha_{\mathcal{T}} = [\![\neg(\neg\psi_1 \boxminus \neg\psi_2)]\!]^\alpha_{\mathcal{T}}$$

with

$$[\![\mu X.\ \psi]\!]^\alpha_{\mathcal{T}} = \bigcap \{\mathcal{S}' \subseteq \mathcal{S} \mid [\![\psi]\!]^{\alpha[X \mapsto \mathcal{S}']}_{\mathcal{T}} \subseteq \mathcal{S}'\}$$
$$[\![\nu X.\ \psi]\!]^\alpha_{\mathcal{T}} = \bigcup \{\mathcal{S}' \subseteq \mathcal{S} \mid [\![\psi]\!]^{\alpha[X \mapsto \mathcal{S}']}_{\mathcal{T}} \supseteq \mathcal{S}'\}$$

[1] This blowup can be avoided at the cost of extra complexity, so we decide to simply stipulate guardedness for the sake of simplicity.

and where $[\![\psi_1 \rhd \psi_2]\!]^\alpha_{\mathcal{T}}$ is defined as

$$\{s \mid \text{ ex. } s' \in \mathcal{S}, d, d' \in \mathbb{R}^{\geq 0} \text{ s.t. } s + d \to s' \text{ and } s' + d' \in [\![\psi_2]\!]^\alpha_{\mathcal{T}}, \\ \text{and } s + d'' \in [\![\psi_1]\!]^\alpha_{\mathcal{T}} \text{ f.a. } 0 \leq d'' \leq d \text{ and } s' + d''' \in [\![\psi_1 \vee \psi_2]\!]^\alpha_{\mathcal{T}} \text{ f.a. } 0 \leq d''' \leq d'\}.$$

Here, the *temporal next relation* $\rhd$ is defined as the composition of a delay, a discrete transition, and another delay. The clause that all states after the discrete transition need to satisfy $\psi_1 \vee \psi_2$ is standard to cope with the fact that there might not be a smallest delay that makes ψ_2 true. Standard patterns in temporal logic, e.g., $\mathtt{E}(p\, \mathtt{U}_{[3,4]}\, q)$, which says that there is a path on which p holds until q holds, and that q holds after time elapsed between 3 and 4 time units, can be expressed as e.g., $\mathtt{z}.\ \mu X.\ (q \wedge 3 \leq \mathtt{z} \wedge \mathtt{z} \leq 4) \vee p \rhd X$.

The model-checking problem for T_μ is then to decide, given a timed automaton and a T_μ formula φ, whether $\mathcal{T} \models \varphi$ where $\mathcal{T}$ is the tTS defined by the timed automaton.

▶ **Proposition 1** ([2]). *The model-checking problem for T_μ is EXPTIME-complete.*

2.5 Tail Recursion

Bounded-Alternation Parity Automata. Let Σ be partitioned into $\Sigma_{\mathsf{re}} \cup \Sigma_{\mathsf{ur}}$. Let $\mathcal{P} = (Q, \Sigma, \delta, q_I, \Lambda)$ be an APT such that Q is partitioned into sets $\{q_\top, q_\bot\} \cup Q_{\mathsf{ur}}, Q_1, \ldots, Q_m$ for some m. Let $q \in Q_j$ for some $1 \leq j \leq m$ and let $a \in \Sigma^2$. We say that $\mathcal{A}$ is *branching* at q and a if $\delta(q, a) = (q_1, q_\top) \vee (q_\top, q_2)$ with $q_1, q_2 \in Q_j \cup \{q_\top, q_\bot\}$ or $\delta(q, a) = (q_1, q_2)$ with $q_2 \in Q_j \cup \{q_\top, q_\bot\}$ and $q_1 \in Q_{j-1} \cup \cdots \cup Q_1 \cup \{q_\top, q_\bot\}$. It is *universal* at q and a if $\delta(q, a) = (q_1, q_2)$ with both $q_1, q_2 \in Q_j \cup \{q_\top, q_\bot\}$, or $\delta(q, a) = (q_1, q_\top) \vee (q_\top, q_2)$ with $q_1 \in Q_{j-1} \cup \cdots \cup Q_1 \cup \{q_\top, q_\bot\}$ and $q_2 \in Q_j \cup \{q_\top, q_\bot\}$.

$\mathcal{P}$ is called a *bounded-alternation APT* (baAPT) if it satisfies the following conditions:

1. Q is partitioned into sets $\{q_\top, q_\bot\}, Q_{\mathsf{ur}}, Q_1, \ldots, Q_m$ as per above.
2. For each $1 \leq i \leq m$, each Q_i is labeled as either branching or universal. Q_{ur} is not labeled.
3. If Q_i is branching, then for each $q \in Q_i$ and for each $a \in \Sigma_{\mathsf{re}}$, $\mathcal{A}$ is branching at q and a. If Q_i is universal, then for each $q \in Q_i$ and for each $a \in \Sigma_{\mathsf{re}}$, $\mathcal{A}$ is universal at q and a.
4. For each $q \in Q$ and $a \in \Sigma^i_{\mathsf{ur}}$, we have that $\delta(q, a) \in \mathcal{B}^+(Q^i_{\mathsf{ur}})$.
5. For each $q \in Q_{\mathsf{ur}}$ and $a \in \Sigma^i_{\mathsf{re}}$, we have that $\delta(q, a) \in \mathcal{B}^+((Q \setminus Q_{\mathsf{ur}})^i)$.

The intuition here is as follows: The acceptance problem of a baAPT in which each state is branching or in which each state is universal can be reduced to a one-player game. The acceptance problem for such a baAPT with empty set Q_{ur} is a bounded-alternation game. For general baAPT, the reduction to a bounded-alternation game is still possible if offending states are in a set of lower index and there is an a priori bound on the number of times a play passes through states in Q_{ur}. The definition of tail-recursive HORS below yields this bound.

Tail-Recursion for λY Terms. Let t be a term of the λY calculus over $\Sigma_{\mathsf{re}} \cup \Sigma_{\mathsf{ur}}$, $\mathcal{L}$ and $\mathcal{F}$. It is called *tail-recursive* if it satisfies the following conditions:

1. For all subterms of t of the form $t_1\ t_2$, the operand-side subterm t_2 has no free $\mathcal{F}$ variables.
2. For all subterms of the form $a\ t_0 \cdots t_j$ with $a \in \Sigma_{\mathsf{ur}}$, none of the t_i has free $\mathcal{F}$ variables.

Note that there are no restrictions w.r.t. variables in $\mathcal{L}$.

▶ **Definition 2.** *Let Σ be partitioned into $\Sigma_{\mathsf{re}} \cup \Sigma_{\mathsf{ur}}$. Let t be a closed tail-recursive term of type $\bullet$ in the λY calculus over Σ, and let $\mathcal{P}$ be a baAPT. The problem of tail-recursive HORS model checking is to decide whether $\mathcal{P}$ accepts $BT(t)$.*

▶ **Proposition 3** ([9]). *The problem of tail-recursive HORS model checking (i.e., against a baAPT) for terms of order $k > 0$ is complete for $(k-1)$-EXPSPACE.*

3 Higher-Order Recursive Timed Automata

Timed automata define tTS, where each state is a pair of a location and a clock evaluation, the former which defines the propositonal labeling of the state. Transitions in tTS are either delay transitions that let time flow but keep the location component, or discrete transitions that keep time fixed, change the location, and reset clocks as indicated by the transition. The first kind of transition respects location invariants, while the second kind respects guards.

The idea for recursively defined tTS is that locations are not a fixed, finite set, but are the nodes of a tree generated by a λY term. The propositional labeling is derived from the labeling of the respective node, i.e., the tree constructor in question. Moreover, we define HORS for tTS in such a way that a tree constructor also carries information on the location invariant associated to it, and the guards and resets of its successors.

▶ **Definition 4.** *Let $\mathcal{X}$ be a finite set of clocks and let Prop be a finite set of propositions. A* timed tree alphabet *(over $\mathcal{X}$ and Prop) is a finite, nonempty set Σ together with functions $ar, \lambda, \iota, trns$ of domain Σ where, for each $a \in \Sigma$,*

1. *$ar(a) \in \mathbb{N}$ indicates the* arity *of the symbol (as for standard tree alphabets),*
2. *$\lambda(a) \subseteq Prop$ indicates the* propositional labeling *of a node labeled by the symbol,*
3. *$\iota(a) \in CC(\mathcal{X})$ indicates the* invariant *of a node labeled by the symbol, and*
4. *$trns(a) \in (CC(\mathcal{X}) \times 2^{\mathcal{X}})^i$ where $i = ar(a)$ indicates the guards and resets on the edges to the i many successors of a node labeled by the symbol.*

We write $trns_i(a)$ to denote the ith element of the tuple $trns(a)$.

Of course, a timed tree alphabet is a tree alphabet by ignoring the extra functions. A tree labeled by such a timed alphabet gives rise to a tTS as follows.

▶ **Definition 5.** *Let Σ be a timed tree alphabet over $\mathcal{X}$ and Prop and let (T, l) be a Σ-tree. The tTS defined by (T, l) is $\mathcal{T}_T = (\mathcal{S}, \to, s_0, \lambda)$ where*

- *$\mathcal{S} = \{(v, \eta) \mid \eta \models \iota(l(v))\}$, where $v \in T$ and $\eta \colon \mathcal{X} \to \mathbb{R}^{\geq 0}$,*
- *the initial state is (ϵ, η_0) with $\eta_0(\boldsymbol{x}) = 0$ f.a. $\boldsymbol{x} \in \mathcal{X}$,*
- *delay transitions keep the tree node but let time flow: for any $(v, \eta) \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$ we have a transition $(v, \eta) \xrightarrow{d} (v, \eta + d)$ if $\eta + d' \models \iota(l(v))$ for all $0 \leq d' \leq d$.*
- *discrete transitions are derived from $trns(l(v))$: for all $(v, \eta) \in \mathcal{S}, i < ar(lv)$, we have $(v, \eta) \to (vi, \eta|_R)$ if $trns_i(l(v)) = (\chi, R)$ and $\eta \models \chi$ and $\eta|_R \models \iota(l(vi))$,*
- *the propositional labeling λ feeds through the labeling of a tree node, i.e., $\lambda(v, \eta) = \lambda(l(v))$,*
- *all states are labeled by the clock constraints that hold there, i.e., $(v, \eta) \models \chi$ iff $\eta \models \chi$.*

Note that $\mathcal{T}_T$ is generally not a tree due to additivity of time: there are uncountably many paths from (v, η) to $(v, \eta + d)$ for any $d > 0$ s.t. the location invariant is satisfied. Other than this, the transition system is a tree though.

A λY term t over a timed alphabet is called a *Higher-Order Recursive Timed Automaton* (HORTA). It generates a tTS $\mathcal{T}_t$ via the tree generated from the term. Its order is that of the order of the underlying λY term. The *index* $m(t)$ of a HORTA is the largest constant that occurs in its guards or invariants, its *arity* $ar(t)$ is the maximal arity of a symbol in its tre alphabet. The *size* of a HORTA is $\|t\| = |t| \cdot |\Sigma| \cdot (|Prop| + |\mathcal{X}|^2 \cdot m(\mathcal{A}) \cdot ar(t))$ where $|t|$ defines the size of the underlying λY term.

The HORTA model-checking problem is then the following:

given:	a HORTA t and a formula φ of the timed μ-calculus
decide:	does $\mathcal{T}_t \models \varphi$ hold?

An example can be found in the appendix.

4 Tail Recursion

Let $\mathcal{X}$ be a set of clocks, $\mathcal{Y}$ be a set of specification clocks and *Prop* be a set of atomic propositions with $p_{\mathsf{sf}} \in \mathit{Prop}$ a designated proposition. This proposition takes the role of a tree constructor from Σ_{ur} for tail-recursive terms, i.e., it signals tree nodes s.t. its subtrees are defined using terms without free Y variables. Hence, given a HORTA, this designated proposition p_{sf} partitions its tree alphabet Σ into $\Sigma_{\mathsf{ur}} = \{a \mid p_{\mathsf{sf}} \in \lambda(a)\}$ and $\Sigma_{\mathsf{re}} = \{a \mid p_{\mathsf{sf}} \notin \lambda(a)\}$. A HORTA is *tail recursive* if it is tail recursive as a HORS using this partition.

Now let $\mathcal{V} = \mathcal{V}_1^{\mathsf{br}} \cup \mathcal{V}_1^{\mathsf{un}} \cup \cdots \cup \mathcal{V}_m^{\mathsf{br}} \cup \mathcal{V}_m^{\mathsf{un}} \cup \mathcal{V}_{\mathsf{ur}}$ be the set of fixpoint variables, partitioned into $2m+1$ different sets. Each such variable plays the same role as a state in a baAPT, including being either branching, universal, or unrestricted, an being partitioned into different sets. Hence, the the annotations mark the variables as branching, universal, or unrestricted (superscript), and fix the respective set of the partition (subscript).

A T_μ formula has *bounded-alternation* if it can be derived from ψ_m^{br} or ψ_m^{un} in the following grammar:

$$
\begin{aligned}
\psi^{\mathsf{sf}} &= \mathtt{tt} \mid \mathtt{ff} \mid p \mid \neg p \mid \chi \mid \neg\chi \mid \psi^{\mathsf{sf}} \vee \psi^{\mathsf{sf}} \mid \psi^{\mathsf{sf}} \wedge \psi^{\mathsf{sf}} \mid p_{\mathrm{sf}} \wedge \psi^{\mathsf{ur}} \\
\psi_m^{\mathsf{br}} &= \psi^{\mathsf{sf}} \mid X_m^{\mathsf{br}} \mid \neg\psi_{m-1} \mid \psi_m^{\mathsf{br}} \vee \psi_m^{\mathsf{br}} \mid \psi_{m-1} \wedge \psi_m^{\mathsf{br}} \mid \psi_{m-1} \rhd \psi_m^{\mathsf{br}} \mid \mu X_m^{\mathsf{br}}.\ \psi_m^{\mathsf{br}} \mid \nu X_m^{\mathsf{br}}.\ \psi_m^{\mathsf{br}} \\
\psi_m^{\mathsf{un}} &= \psi^{\mathsf{sf}} \mid X_m^{\mathsf{un}} \mid \neg\psi_{m-1} \mid \psi_{m-1} \vee \psi_m^{\mathsf{un}} \mid \psi_m^{\mathsf{un}} \wedge \psi_m^{\mathsf{un}} \mid \psi_{m-1} \boxminus \psi_m^{\mathsf{un}} \mid \mu X_m^{\mathsf{un}}.\ \psi_m^{\mathsf{un}} \mid \nu X_m^{\mathsf{un}}.\ \psi_m^{\mathsf{un}} \\
&\ \vdots \qquad\qquad \vdots \\
\psi_1^{\mathsf{br}} &= \psi^{\mathsf{sf}} \mid X_1^{\mathsf{br}} \mid \psi_1^{\mathsf{br}} \vee \psi_1^{\mathsf{br}} \mid \psi^{\mathsf{sf}} \wedge \psi_1^{\mathsf{br}} \mid \psi^{\mathsf{sf}} \rhd \psi_{1m}^{\mathsf{br}} \mid \mu X_m^{\mathsf{br}}.\ \psi_m^{\mathsf{br}} \mid \nu X_m^{\mathsf{br}}.\ \psi_m^{\mathsf{br}} x \\
\psi_1^{\mathsf{un}} &= \psi^{\mathsf{sf}} \mid X_1^{\mathsf{un}} \mid \psi^{\mathsf{sf}} \vee \psi_1^{\mathsf{un}} \mid \psi_1^{\mathsf{un}} \wedge \psi_1^{\mathsf{un}} \mid \psi^{\mathsf{sf}} \boxminus \psi_1^{\mathsf{un}} \mid \mu X_1^{\mathsf{un}}.\ \psi_1^{\mathsf{br}} \mid \nu X_1^{\mathsf{un}}.\ \psi_1^{\mathsf{br}} \\
\psi^{\mathsf{ur}} &= \psi^{\mathsf{sf}} \mid \psi^{\mathsf{ur}} \vee \psi^{\mathsf{ur}} \mid \psi^{\mathsf{ur}} \wedge \psi^{\mathsf{ur}} \mid \psi^{\mathsf{sf}} \rhd \psi^{\mathsf{sf}} \mid \psi^{\mathsf{sf}} \boxminus \psi^{\mathsf{sf}} \\
&\quad \mid \mu X^{\mathsf{ur}}.\ \psi^{\mathsf{ur}} \mid \nu X^{\mathsf{ur}}.\ \psi^{\mathsf{ur}} \mid (\neg p_{\mathrm{sf}} \wedge \psi_m)
\end{aligned}
$$

where $X_i^{\mathsf{br}}, X_i^{\mathsf{un}}$ are in the respective sets of variables for all i, and so is X^{ur}. A term ψ_i can denote both ψ_i^{un} and ψ_i^{br}.

We give some intuition for this definition of bounded-alternation T_μ-formulas. A key point for tail-recursive behavior is limited boolean alternation, i.e. between $\vee$ and $\wedge$. The reason for that is that space-bounded algorithms need bounded boolean alternation, for (fully) alternating space is exponential time. baAPT achieve this by (partially) limiting the acceptance game to a game that is played by one player only, either $\exists$ or $\forall$, with the other player only making token moves without associated freedom of decision. This is then captured in the intuition for bounded-alternation T_μ formulas in a syntactic way.

However, a better intuition is that a formula is alternation-free if it either does not contain $\wedge$ and $\boxminus$, or if it does not contain $\vee$ and $\rhd$, and no negations. An example of this is $\mu X.\ p \vee (\mathtt{tt} \rhd X)$, or $\mu X.\ p \wedge (\mathtt{tt} \boxminus X)$. Note that the polarity of the fixpoint does not matter here.

Limited alternation is then allowed by stratifying these formulas (via the indices in e.g., $\psi_m^{\mathsf{br}}, \ldots, \psi_1^{\mathsf{br}}$) and allowing one-time use of forbidden operators at the price of dropping to a lower level in the hierarchy. Of course, formulas that contain no recursive definitions at all, e.g., those in ψ^{sf} are also safe. Hence, we obtain e.g.,

$$\mu X.(\mathtt{tt} \rhd x) \vee \big(\neg p \wedge (q \vee \nu Z.\ p \wedge (\mathtt{tt} \boxminus Z))\big).$$

Since bounded-alternation formulas are to be evaluated over tTS defined by tail-recursive λY terms, there is an additional ingredient: In tail-recursive λY terms, subterms in operand position cannot contain free Y variables, whence the tree generated by such a subterm is

completely independent of the rest of the term (excluding arguments of its own, of course). Moreover, we use special alphabet symbols (those in Σ^{ur}) to signal that a term appears e.g., in operand position, or that it contains no free Y variables. Since the tree defined by such a term is independent of the rest of the tree, and since this can only repeat a polynomial amount of times (due to passing to a strict subterm), a model-checking procedure can solve the problem for the tree generated by the subterm first, and then return to the model-checking problem for the overall tree. Since this is such an independent subproblem, it is not harmful to "reset" the amount of boolean alternation possible in a T_μ formula/a baAPT. This is what happens in formulas of the form ψ^{ur}, which have nontrivial semantics only in states where p_{sf} is true (and, hence a tree constructor from Σ^{ur} was read, whence we are in such an independent subproblem). Note that it is not important that this generates a polynomial bound on the number of "alternation resets" in a run of a baAPT/the evaluation of a bounded-alternation T_μ formula. In general, there is no such bound. However, the model-checking problem will always only generate a polynomial stack of open subproblems due to alternation resets, and that suffices to yield the complexity bound.

The example in the appendix contains a simple example of a pair of a tail-recursive HORTA and a bounded-alternation T_μ formula.

5 Upper Bounds for Model-Checking

The goal of this section is to establish upper bounds for model checking of the trees generated by HORTA (tail recursive or not) against T_μ formulas. For the rest of the section, fix a tree alphabet Σ, a λY term t and a T_μ formula φ. Assume w.l.o.g. that φ is in *negation normal form*, i.e., that negations appear only in front of propositions or clock constraints.

5.1 The Region Abstraction

The region abstraction is a classical result (see e.g., [4] or [6], Def. 9.42), that maps the infinite transition system defined by a TA onto a finite transition system. While we work with tTS defined by HORTA, the key point is the same: the region abstraction uses an equivalence relation $\simeq_m$, for $m \in \mathbb{N}$, on clock evaluations. Let $\mathcal{X}$ be a set of clocks, specification or otherwise. Then $\simeq_m$ is defined as follows: $\eta \simeq_m \eta'$ iff

$$\begin{aligned}
&\text{for all } \mathtt{x} \in \mathcal{X} : \eta(\mathtt{x}) > m \text{ and } \eta'(\mathtt{x}) > m \\
&\qquad \text{or } \lfloor \eta(\mathtt{x}) \rfloor = \lfloor \eta'(\mathtt{x}) \rfloor \text{ and } \mathit{frac}(\eta(\mathtt{x})) = 0 \Leftrightarrow \mathit{frac}(\eta'(\mathtt{x})) = 0 \\
&\qquad\qquad \text{and for all } \mathtt{y} \in \mathcal{X} \text{ with } \eta(\mathtt{y}) \leq m \text{ and } \eta'(\mathtt{y}) \leq m : \\
&\qquad\qquad\qquad \mathit{frac}(\eta(\mathtt{x})) \leq \mathit{frac}(\eta(\mathtt{y})) \Leftrightarrow \mathit{frac}(\eta'(\mathtt{x})) \leq \mathit{frac}(\eta'(\mathtt{y})).
\end{aligned}$$

$\mathit{frac}(r)$ is the fractional part of a real number. Clock evaluations are equivalent if, for each clock, (I) either both clocks have a value greater than m, or (II) they compare in the same way with respect to all integers $\leq m$. Moreover, the passage of time makes equivalent evaluations reach the next integral value first for the same clock. $\simeq_m$ is an equivalence relation for any m; an equivalence class in $\simeq_m$ is called a *region*. The equivalence class of η under $\simeq_m$ is $[\eta]_m$ (or just $[\eta]$ when m is clear). We define the notion of a *successor region*:

▶ **Definition 6.** *Let $\simeq_m$ denote the region equivalence w.r.t. m. For each region $[\eta]_m$, the unique* successor region *is*

- *$\mathit{suc}([\eta]_m) = [\eta]_m$ if $\eta(\mathtt{x}) > m$ for all $\mathtt{x} \in \mathcal{X}$,*
- *$\mathit{suc}([\eta]_m) = [\eta']_m$ iff there is $d \in \mathbb{R}^{\geq 0}$ such that $\eta{+}d = \eta'$, and $\eta{+}d' \in [\eta]_m \cup [\eta']_m$ for all $0 < d' < d$, and $[\eta]_m \neq [\eta']_m$.*

The second term defines the successor region of $[\eta]$ to be the first region that is entered if time passes from any $\eta'' \in [\eta]$, and the first term makes the successor region well-defined in regions where all clocks have values greater than m. We call this region the *maximal region*.

5.2 The Untiming Construction

Let $\mathcal{X}$ be a set of clocks, let $\mathcal{Y}$ be a set of specification clocks. Let $\mathcal{T}_t = (\mathcal{S}, \to, s_0, \lambda)$ be the tTS generated by the HORTA t. Note that $\mathcal{S} \subseteq \{(v, \eta) \mid v \in T, \eta \models \iota(lv)\}$.

Let Prop, Ξ be the propositions, resp. clock constraints mentioned by φ, and let $\mathcal{Y}$ be the specification clocks that φ mentions. Define an (untimed) μ-calculus formula $\hat{\varphi}$ via

$$\hat{p} = p \qquad \hat{X} = X$$
$$\hat{\chi} = \chi \qquad \widehat{\neg\varphi'} = \neg\hat{\varphi'}$$
$$\widehat{\varphi_1 \vee \varphi_2} = \hat{\varphi}_1 \vee \hat{\varphi}_2 \qquad \widehat{\varphi_1 \wedge \varphi_2} = \hat{\varphi}_1 \wedge \hat{\varphi}_2$$
$$\widehat{\varphi_1 \rhd \varphi_2} = \mu Z.\ (\hat{\varphi}_1 \wedge \Diamond_D Z) \vee (\Diamond_T(\mu Z'.\ (\hat{\varphi}_1 \wedge \Diamond_D Z') \vee (\hat{\varphi}_1 \vee \hat{\varphi}_2))) \qquad \widehat{\mathbf{z}.\ \varphi'} = \Diamond_z \hat{\varphi'}$$
$$\widehat{\varphi_1 \boxminus \varphi 2} = \nu Z.\ (\hat{\varphi}_1 \wedge \Box_D Z) \wedge (\Box_T(\nu Z'.\ (\hat{\varphi}_1 \wedge \Box_D Z') \wedge (\hat{\varphi}_1 \vee \hat{\varphi}_2)))$$
$$\widehat{\mu X.\ \varphi'} = \mu X.\ \hat{\varphi'} \qquad \widehat{\nu X.\ \varphi'} = \nu X.\ \hat{\varphi'}$$

where the Z, Z' for the $\rhd$ and $\boxminus$ are fresh for each subformula. The semantics of the untimed modal operators are standard: $[\![\Diamond_D \psi]\!]^\alpha_{\mathcal{T}} = \{s \mid \text{ ex. } s' \in [\![\psi]\!]^\alpha_{\mathcal{T}} \text{ s.t. } s \xrightarrow{D} s'\}$ and similarly for $\Box_D$ and the other modalities. Note that this definition is for untimed transition systems.

Let m be that largest integer in any location invariant or guard in t or a clock constraint in φ. Let $[\eta]$ denote $[\eta]_m$ from now on. Define a new (untimed) transition system with transitions $\{D, T\} \cup \{z \mid z \in \mathcal{Y}\}$ and propositions in $\mathit{Prop} \cup \Xi$ via $\mathcal{T}^u_T = (\mathcal{S}_u, \to_u, s'_0, \lambda')$ via

- $\mathcal{S}_u = \{(v, [\eta]) \mid (v, \eta) \in \mathcal{S}\}$,
- $\to_u = \{(v, [\eta]) \xrightarrow{D} (v, \mathit{suc}(\eta)) \mid \mathit{suc}(\eta) \models \iota(l(v))\} \cup \{(v, [\eta]) \xrightarrow{T} (vi, [\eta']) \mid (v, \eta) \to (vi, \eta')\} \cup \{(v, [\eta]) \xrightarrow{\mathbf{z}} (v, [\eta]|_{\mathbf{z}}) \mid \eta|_{\mathbf{z}} \models \iota(l(v))\}$,
- $s'_0 = (\epsilon, [\eta_0])$,
- $\lambda'((v, [\eta])) = \lambda(v, \eta) = \lambda(v) \cup \{\chi \in \Xi \mid \eta \models \chi\}$.

Finally, given a variable assignment $\alpha \colon \mathcal{V} \to 2^{\mathcal{S}}$, define $\hat{\alpha} \colon \mathcal{V} \to 2^{\mathcal{S}'}$ via $\hat{\alpha}(X) = \{(v, [\eta]) \mid (v, \eta) \in \alpha(X)\}$.

▶ **Proposition 7** ([2]). *For all subformulas ψ of φ, for all $(v, \eta) \in \mathcal{S}$, and for all variable assignments α we have $(v, \eta) \in [\![\psi]\!]^\alpha_{\mathcal{T}}$ iff $(v, [\eta]) \in [\![\hat{\psi}]\!]^{\hat{\alpha}}_{\mathcal{T}^u_T}$.*

Technically the proof in [2] is only for tTS defined via TA, not via HORTA. However, finiteness is not a crucial ingredient of the proof, and the region graph construction is very robust.

5.3 The Reduction

We have seen before that, on the region graph, the formula $\hat{\varphi}$ captures the semantics of φ. We define an APT $\mathcal{P}_\varphi$ alongside the logic of $\hat{\varphi}$, but based on the syntax of φ itself. The reason for that is in the details of the translation of $\rhd$ and $\boxminus$, where we have to make sure that the regions are visited one by one, but must also avoid excessive pointless delays. The states of $\mathcal{P}_\varphi$ will be the product of the set of subformulas of φ and the set of regions in the region graph, plus some extra copies in some places. Transitions will follow the logic of subformulas in the first component, and pass through the regions as need be at formulas with real-time semantics, e.g., $\rhd, \boxminus$.

As a preparation, let Σ be the tree alphabet used in t. Define $\Sigma' = \Sigma \cup \{a_{\text{aux}} \mid a \in \Sigma\}$ where all the new tree constructors are unary. Let m be the integer used for the region graph. The longest sequence of delay transitions that can be taken before reaching the maximal region is less than $k' = |\mathcal{X} \cup \mathcal{Y}| \cdot 3 \cdot m$. Let k be the length of the longest path in the syntax tree of φ that contains no formulas of the form $\psi_1 \rhd \psi_2$ or $\psi_1 \boxminus \psi_2$, including passing through fixpoint variables. This number exists due to guardedness. Write $\mathbf{k}$ for $2 \cdot k' + k$.

Obtain a λY term $\hat{t}$ from t by replacing each occurrence of a tree constructor a by $a^{\mathbf{k}}_{\text{aux}}\ a$. i.e., by prepending it by a sequence of $\mathbf{k}$ many copies of a_{aux}. These tree constructors give the yet-to-be-defined automaton enough space to (I) explicitly simulate delay transitions in the region graph by transitions reading a_{aux} in the tree, and (II) allow easy handling of the logic of subformulas by transitions without advancing in the tree proper.

Let $\mathcal{P}_\varphi = (Q, \Sigma', \delta, q_I, \Lambda)$ with the individual components defined as follows:

- $Q = sub(\varphi) \times \mathbb{R}^{\geq 0}/_{\simeq_m} \times \{0,1,2\}$ and $q_I = (\varphi, [\eta_0], 0)$
- Λ is defined inductively via the *alternation index* of a subformula of φ. Let $ai(\psi)$ be defined inductively for formulas of the form $\mu X.\ \psi'$ and $\nu x.\ \psi'$ as follows:
  - For formulas of the form $\psi = \mu x.\ \psi'$ set $ai(\psi)$ to the smallest even integer that is at least as big as $ai(\psi'')$ for all subformulas ψ'' of ψ, but at least 0.
  - For formulas of the form $\psi = \mu x.\ \psi'$ set $ai(\psi)$ to the smallest odd integer that is at least as big as $ai(\psi'')$ for all subformulas ψ'' of ψ, but at least 1.

  Fo other ψ', set $ai(\psi)$ to $ai(\psi')$ where ψ' is the first subformula of the form $\mu X.\ \psi'$ or $\nu X.\ \psi'$ on the path to the root in the syntax tree of φ. Then $\Lambda((\psi, [\eta], i)) = ai(\psi)$.

Finally, δ is defined (for the non-modalities) as follows:

$$
\begin{aligned}
\delta((p, [\eta], 0), a_{\text{aux}}) &= \top \text{ if } p \in \lambda(a); \bot \text{ ow.} \\
\delta((\neg p, [\eta], 0), a_{\text{aux}}) &= \bot \text{ if } p \in \lambda(a); \top \text{ ow.} \\
\delta((\chi, [\eta], 0), a_{\text{aux}}) &= \top \text{ if } \eta \models \chi; \bot \text{ ow.} \\
\delta((\neg\chi, [\eta], 0), a_{\text{aux}}) &= \bot \text{ if } \eta \not\models \chi; \top \text{ ow.} \\
\delta((\psi_1 \vee \psi_2, [\eta], 0), a_{\text{aux}}) &= (\psi_1, [\eta], 0) \vee (\psi_2, [\eta], 0) \\
\delta((\psi_1 \wedge \psi_2, [\eta], 0), a_{\text{aux}}) &= (\psi_1, [\eta], 0) \wedge (\psi_2, [\eta], 0) \\
\delta((\mu X.\ \psi, [\eta], 0), a_{\text{aux}}) &= (\psi, [\eta], 0) \\
\delta((\nu X.\ \psi, [\eta], 0), a_{\text{aux}}) &= (\psi[\eta], 0) \\
\delta((X, [\eta], 0), a_{\text{aux}}) &= (fp(X), [\eta], 0) \\
\delta((\mathtt{z}.\ \psi, [\eta], 0), a_{\text{aux}}) &= (\psi, [\eta|_{\{z\}}], 0) \text{ if } \eta|_{\{\mathtt{z}\}} \models \iota(a)
\end{aligned}
$$

The intuition here is that the acceptance game simply follows the logic of the subformula in question. The transitions for $\rhd$ are

$$
\begin{aligned}
\delta((\psi_1 \rhd \psi_2, [\eta], 0), a_{\text{aux}}) &= (\psi_1, [\eta], 0) \wedge (\psi_1 \rhd \psi_2, [\eta], 1) \text{ if } suc([\eta]_m) \not\models \iota(a) \text{ or } [\eta] \text{ is max.} \\
\delta((\psi_1 \rhd \psi_2, [\eta], 0), a_{\text{aux}}) &= (\psi_1, [\eta], 0) \wedge ((\psi_1 \rhd \psi_2, [\eta], 1) \vee (\psi_1 \rhd \psi_2, suc([\eta]), 0)) \\
&\qquad \text{if } suc([\eta]_m) \models \iota(a) \text{ and } [\eta] \text{ is not max.} \\
\delta((\psi_1 \rhd \psi_2, [\eta], 1), a_{\text{aux}}) &= (\psi_1 \rhd \psi_2, [\eta], 1) \\
\delta((\psi_1 \rhd \psi_2, [\eta], 1), a) &= \bigvee_{j \in \{0,\ldots,ar(a), trns_j(a) = (\chi, R), \eta \models \chi} (q^i_\top, (\psi_1 \rhd \psi_2, [\eta|_R], 2), q_\top^{ar(a)-i-1}) \\
\delta((\psi_1 \rhd \psi_2, [\eta], 2), a_{\text{aux}} &= \bot \text{ if } \eta \not\models \iota(a) \\
\delta((\psi_1 \rhd \psi_2, [\eta], 2), a_{\text{aux}}) &= (\psi_2, [\eta], 0) \text{ if } \eta \models \iota(a) \text{ and } suc([\eta]_m) \not\models \iota(a) \text{ or } [\eta] \text{ is max.} \\
\delta((\psi_1 \rhd \psi_2, [\eta], 2), a_{\text{aux}}) &= (\psi_1, [\eta], 2) \wedge ((\psi_2, [\eta], 0) \vee (\psi_1 \rhd \psi_2, suc([\eta]), 2)) \\
&\qquad \text{if } \eta \models \iota(a) \text{ and } suc([\eta]_m) \models \iota(a) \text{ and } [\eta] \text{ is not max.}
\end{aligned}
$$

We omit the clauses for $\psi_1 \boxminus \psi_2$ as they are dual to those for $\psi_1 \rhd \psi_2$. The logic of a temporal next follows three phases: a delay, a discrete transition, and then another delay. First, if the last component of the state is 0, $\exists$ advances to successor regions of she so desires, and if this is possible. Then, if the last component is 1, the sequence of a_{aux} is consumed and an actual transition happens at a, with an immediate check for satisfaction of the location invariant. Then, $\exists$ can delay further if she so desires. After she is done with this, the APT tracks ψ_2.

▶ **Lemma 8.** *Let $\mathcal{T}_{\hat{t}}$ be the tree generated by $\hat{t}$. Then $\mathcal{P}_\varphi$ accepts $\mathcal{T}_{\hat{t}}$ iff $(\epsilon, \eta_0) \in [\![\varphi]\!]_{\mathcal{T}_t}$.*

It is immediate that $\hat{t}$ is tail-recursive if t is. Moreover, one can show that $\mathcal{P}_\varphi$ is a baAPT if φ has bounded alternation.

▶ **Theorem 9.** *The model-checking problem for order-k HORTA against T_μ formulas is in $(k+1)$-EXPTIME. For tail-recursive order-k-HORTA and bounded-alternation formulas, the problem is in k-EXPSPACE.*

Proof. Note that $\mathcal{P}_\varphi$ is exponential in the input, since it contains the region graph in its state space. Since the model-checking problem for order-k HORS is k-EXPTIME-complete [20], we obtain the result. For the tail-recursive case, we obtain the claim via Prop. 3. ◀

6 Matching Lower Bounds

We now show that the model-checking problem for bounded-alternation T_μ formulas against trees generated by order-1 HORTA is EXPSPACE-hard, via a reduction from the corridor tiling problem (see below). The general case of k-EXPSPACE-hardness for model-checking order-k HORTA is left for a future publication. The proof uses an idea from [12, 11].

6.1 Tiling Problems

A *tiling system* is a $\mathcal{K} = (K, H, V, k_I, t_\square, t_F)$ where K is a finite set of *tiles*, $H, V \subseteq K \times K$ are the *horizontal* and *vertical matching relations*, and $k_I, k_\square, k_F$ are the *initial*, *blank* and *final tiles*. The corridor tiling problem consumes as input a tiling system $\mathcal{K}$ and some $n \in \mathbb{N}$ encoded unarily. It asks whether there is a sequence $\rho_0, \dots, \rho_{m-1}$, where each of the ρ_i is a K-word of length 2^n that satisfies the following:

- $\rho_0 = k_I k_\square \cdots k_\square$,
- for each $0 \leq i \leq m-1$, and $0 \leq j < 2^n - 1$, we have $(\rho_i(j), \rho_i(j+1)) \in H$,
- for each $0 \leq i \leq m-2$ and $0 \leq j \leq 2^n - 1$, we have $(\rho_i(j), \rho_{i+1}(j)) \in V$, and
- $\rho_{m-1}(0) = k_F$,

where $\rho_i(j)$ denotes the jth tile in ρ_i. We call the ith word in this solution the ith *row*.

▶ **Proposition 10** ([23, 15])**.** *The corridor tiling problem is EXPSPACE-hard.*

6.2 Encoding Rows

Let $\mathcal{K} = (K, H, V, k_I, k_\square, k_F)$ be a tiling system, and assume that K is ordered, e.g., via $K = \{k_0, \dots, k_{m-1}\}$ for $m = |K|$. W.l.o.g. let $k_I = k_0, k_\square = k_1$. A *row candidate* is a K-word of length 2^n. Such a row candidate is a row if it satisfies the horizontal matching. Row candidates are also lexicographically ordered. Given a row candidate ρ, we obtain the lexicographically next one ρ' by setting $\rho'(j) = \rho(j)$ if there is $j < j' < 2^n$ s.t. $\rho(j) \neq k_{m-1}$, and if $\rho(j) = k_i$ and $\rho(j') = k_{m-1}$ for all $j < j' < 2^n$, then $\rho'(j) = k_{i+1 \pmod m}$.

We now encode rows and row candidates into trees generated by λY terms over a timed tree alphabet (i.e., HORTA), and, for each $k \in K$, define a T_μ formula that checks whether a given tile in a the row encoded by such a tree is k. These formulas are mutually recursive, but the recursion is guarded by the designated proposition p_{sf} (cf. Sec. 4). The HORTA has one clock $\mathtt{x}$. A node v in the underlying tree encodes a row ρ if, for each $0 \leq i < 2^n$, $(v, \mathtt{x} \mapsto i) \in [\![\psi_k]\!]_{\mathcal{T}_T}$ iff $k = \rho(i)$. Here, $\mathcal{T}_T$ is the tTS defined by the HORTA.

Let $\mathit{Prop} = \{p_I, p_\square, p_{\mathrm{sf}}\}$ and let $\Sigma = \{a^1_{nxt}, a^2_{rw}, b^1_I, b^1_\square\}$ with arities indicated by superscripts. Let $\mathit{trns}_0(b_I) = \mathit{trns}_0(b_\square) = (\mathtt{z} = 1, \{\mathtt{z}\})$. Set $\lambda^{-1}(p_I) = \{b_I\}$ and $\lambda^{-1}(p_\square) = \{b_\square\}$ and $\lambda^{-1}(p_{\mathrm{sf}}) = \{a_{nxt}\}$, where p_{sf} is the designated proposition from the definition of tail recursion/bounded alternation. We write $\rhd_i \psi$ for $\mathtt{z}.\ \mathtt{tt} \rhd (\mathtt{z} = i \wedge \psi)$, and $\boxminus_i \psi$ for $\mathtt{z}.\ \mathtt{tt} \boxminus (\mathtt{z} = i \wedge \psi)$. Consider the λY terms

$$\mathtt{init} = b_I\ (YF.\ b_\square\ F) \qquad\qquad \mathtt{next}\ t = a_{nxt}(YF.\ a_{rw}\ F\ t).$$

The first one defines a tree with b_I as the root and then an infinite sequence of $b_\square$ after that. The other consumes a tree and returns a tree that contains a_{nxt} as the root, and then below it an infinite leftward sequence of a_{rw} that each contain t as their right son. All rows and row candidates we work with are obtained from $\mathtt{init}$ via repeated application of $\mathtt{next}$.

Now consider the mutually recursive T_μ formulas

$$\begin{aligned}
\psi_{k_0} &= p_I \vee \big(\rhd_0 (p_{\mathrm{sf}} \wedge \psi_{k_{m-1}} \wedge \rhd_0 \mu X.\ \neg p_{\mathrm{sf}} \wedge \psi_{k_{m-1}} \wedge (\mathtt{x} = 2^n - 1 \vee \rhd_1 X))\big) \\
&\quad \vee \big(\rhd_0 (p_{\mathrm{sf}} \wedge \psi_{k_0} \wedge \rhd_0 \mu X.\ \neg p_{\mathrm{sf}} \wedge (\mathtt{x} \leq 2^n - 1 \wedge \neg \psi_{k_{m-1}} \vee \rhd_1 X))\big) \\
\psi_{k_1} &= p_\square \vee \big(\rhd_0 (p_{\mathrm{sf}} \wedge \psi_{k_0} \wedge \rhd_0 \mu X.\ \neg p_{\mathrm{sf}} \wedge \psi_{k_{m-1}} \wedge (\mathtt{x} = 2^n - 1 \vee \rhd_1 X))\big) \\
&\quad \vee \big(\rhd_1 (p_{\mathrm{sf}} \wedge \psi_{k_0} \wedge \rhd_0 \mu X.\ \neg p_{\mathrm{sf}} \wedge (\mathtt{x} \leq 2^n - 1 \wedge \neg \psi_{k_{m-1}} \vee \rhd_1 X))\big) \\
\psi_{k_{i+2}} &= \quad \big(\rhd_0 (p_{\mathrm{sf}} \wedge \psi_{k_{i+1}} \wedge \rhd_0 \mu X.\ \neg p_{\mathrm{sf}} \wedge \psi_{k_{m-1}} \wedge (\mathtt{x} = 2^n - 1 \vee \rhd_1 X))\big) \\
&\quad \vee \big(\rhd_0 (p_{\mathrm{sf}} \wedge \psi_{k_{i+2}} \wedge \rhd_0 \mu X.\ \neg p_{\mathrm{sf}} \wedge (\mathtt{x} \leq 2^n - 1 \wedge \neg \psi_{k_{m-1}} \vee \rhd_1 X))\big).
\end{aligned}$$

The idea is that encoding a tile is signaled by a proposition (only for $k_I, k_\square$), or it holds due to the recursive characterization of the lexicographic successor, applied recursively until the propositional base case is reached. This is done by traversing along a_{rw} (used to let time elapse in increments of 1) or a_{nxt} (from $\mathtt{next}\ t$ to t). Given a row candidate ρ, let $\mathtt{next}^i(\rho)$ be the lexicographically ith successor of ρ modulo 2^n.

▶ **Lemma 11.** *The tree generated by* $\mathtt{init}$ *encodes the initial row. Given a tree that encodes a row candidate* ρ*, the tree generated by* $\mathtt{next}^i\ t$ *encodes* $\mathtt{next}^i(\rho)$*.*

6.3 The Reduction

Extend Σ by $\{a^1_v, a^1_h\}$ and Prop by $\{p_h, p_v\}$ with $\lambda^{-1}(p_v) = \{a_v\}, \lambda^{-1}(p_h) = \{a_h\}$. Now consider the λY terms

$$\begin{aligned}
&\mathtt{horiz}\ t = a_h(YF.\ a_{rw}\ F\ t) \\
&\mathtt{vert}\ t\ t' = (YF.\ a_v\ (a_v\ (F\ t\ t'))\ (a_{rw}\ t\ (a_{rw}(t'\ t'))))
\end{aligned}$$

and the T_μ formulas

$$\psi_{\mathtt{horiz}} = \rhd_0 \Big(p_h \wedge \big(\bigwedge_{(k,k') \notin H} \nu X.\ (\psi_k \to \neg \psi_{k'}) \wedge ((\mathtt{x} = 2^n - 1 \vee \rhd_1 (\neg p_{\mathrm{sf}} \wedge X)))\big)\Big)$$

$$\psi_{\mathtt{vert}} = \bigwedge_{(k,k') \notin V} \nu X.\ (\rhd_0 \psi_k \to \neg\, \rhd_0 (\neg p_{\mathrm{sf}} \wedge \rhd_0 \neg p_{\mathrm{sf}} \wedge \psi_{k'}) \wedge ((\mathtt{x} = 2^n - 1 \vee \rhd_0 (p_v \wedge \rhd_1 \neg p_{\mathrm{sf}} \wedge X))))$$

▶ **Lemma 12.** *ψ_{horiz} holds on the tree generated by* `horiz` t *iff t encodes a row, i.e., if the row candidate it encodes satisfies horizontal matching. ψ_{vert} holds on the tree generated by* `vert` $t\ t'$ *iff the rows encoded by t and t' match vertically.*

Finally, extend *Prop* by $\{p_\vee, p_\wedge, p_f\}$ and Σ by $\{a^2_\vee, a^2_\wedge, a^1_f\}$ with $\lambda^{-1}(p_\vee) = \{a_\vee\}$ and $\lambda^{-1}(p_\wedge) = \{a_\wedge\}$ and $\lambda^{-1}(p_f) = \{a_f\}$. Consider the λY terms

$$\begin{aligned}
\texttt{check}\ t\ t' &= a_\wedge\ (\texttt{horiz}\ t')\ (\texttt{vert}\ t\ t') \\
\texttt{exists}\ t\ t' &= Y F.\ a_\vee\ (a_\wedge(\texttt{check}\ t\ t')\ (G\ t'))\ (F\ t\ (\texttt{next}\ t')) \\
\texttt{tiling} &= \big(Y G.\ \lambda t.\ a_\vee\ (a_f\ t)\ (\texttt{exists}\ t\ \texttt{init})\big)\ \texttt{init}
\end{aligned}$$

and the T_μ formulas

$$\begin{aligned}
\psi_{\texttt{final}} &= \psi_{k_1} \\
\psi_{\texttt{chk}} &= \boxminus_0((p_h \wedge \psi_{\texttt{horiz}}) \vee (p_v \wedge \psi_{\texttt{vert}})) \\
\psi_{\texttt{srch}} &= \mu X_{\texttt{srch}}.\ \rhd_0 \big(p_f \wedge \psi_{\texttt{final}}\big) \\
&\qquad \big(p_\vee \wedge \mu Y.\ \rhd_0 (p_\wedge \wedge (\boxminus_0(p_\wedge \wedge \psi_{\texttt{check}} \vee (p_\vee \wedge X))) \vee (p_\vee \wedge Y))\big)
\end{aligned}$$

▶ **Lemma 13.** *ψ_{final} holds on the tree generated by* `final` t *iff t encodes a final row. ψ_{check} holds on the tree generated by* `check` $t\ t'$ *if (I) t encodes a row and (II) t and t' match vertically.*

Finally, ψ_{srch} holds on the tree generated by `tiling` *iff $\mathcal{K}$ has a solution.*

It is not hard to show that `tiling` is tail recursive, but not obvious that $\psi_{\texttt{srch}}$ has bounded alternation.

▶ **Lemma 14.** *The λY term* `tiling` *is tail recursive. The T_μ fomula ψ_{srch} is bounded-alternation and of polynomial size in $\mathcal{K}$.*

This yields the following theorem.

▶ **Theorem 15.** *The model-checking problem of tail-recursive HORTA against bounded-alternation T_μ formulas is EXPSPACE-hard.*

Proof. By Prop. 10, the corridor tiling problem is EXPSPACE-hard. By Lem. 13, an instance of the corridor tiling problem has a solution iff $\psi_{\texttt{srch}}$ holds on the tree generated by `tiling`. Since by Lem. 14, the former is bounded-alternation and the latter is tail-recursive, and both are of polynomial size in $\mathcal{K}$, this constitutes a polynomial-time reduction. ◀

7 Conclusion

We have refined the notion of Real-Time Recursion Schemes [3] into that of HORTA, and established matching upper bounds for the model-checking problem. We have also shown that the model-checking problem for HORTA becomes easier by half an exponential in the tail-recursive setting. The lower bound has been established for the order-1 case, but we conjecture that it can be extended to all orders. The corridor tiling problem needs to be replaced by an exponential version [15], and the encoding of a row needs to be lifted from a tree to a function that consumes a tree, and returns a tree etc., see e.g., [3, 9].

References

1 Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theor. Comput. Sci.*, 354(2):272–300, 2006. doi:10.1016/J.TCS.2005.11.018.

2 Luca Aceto and François Laroussinie. Is your model checker on time? on the complexity of model checking for timed modal logics. *J. Log. Algebraic Methods Program.*, 52-53:7–51, 2002. doi:10.1016/S1567-8326(02)00022-X.

3 Eric Alsmann and Florian Bruse. Real-time higher-order recursion schemes. In Pietro Sala, Michael Sioutis, and Fusheng Wang, editors, *31st International Symposium on Temporal Representation and Reasoning, TIME 2024, October 28-30, 2024, Montpellier, France*, volume 318 of *LIPIcs*, pages 16:1–16:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.TIME.2024.16.

4 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993. doi:10.1006/inco.1993.1024.

5 Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990. doi:10.1007/BFB0032042.

6 C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

7 Massimo Benerecetti, Stefano Minopoli, and Adriano Peron. Analysis of timed recursive state machines. In *2010 17th International Symposium on Temporal Representation and Reasoning*, pages 61–68, 2010. doi:10.1109/TIME.2010.10.

8 Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems II*, pages 64–85, Berlin, Heidelberg, 1995. Springer-Verlag.

9 Florian Bruse. Space-efficient model-checking of higher-order recursion schemes. In Shankaranarayanan Krishna, Sriram Sankaranarayanan, and Ashutosh Trivedi, editors, *Verification, Model Checking, and Abstract Interpretation - 26th International Conference, VMCAI 2025, Denver, CO, USA, January 20-21, 2025, Proceedings, Part I*, volume 15529 of *Lecture Notes in Computer Science*, pages 29–51. Springer, 2025. doi:10.1007/978-3-031-82700-6_2.

10 Florian Bruse, Oliver Friedmann, and Martin Lange. On guarded transformation in the modal μ-calculus. *Log. J. IGPL*, 23(2):194–216, 2015. doi:10.1093/JIGPAL/JZU030.

11 Florian Bruse and Martin Lange. The tail-recursive fragment of timed recursive CTL. *Inf. Comput.*, 294:105084, 2023. doi:10.1016/J.IC.2023.105084.

12 Florian Bruse, Martin Lange, and Étienne Lozes. Space-efficient fragments of higher-order fixpoint logic. In Matthew Hague and Igor Potapov, editors, *Reachability Problems - 11th International Workshop, RP 2017, London, UK, September 7-9, 2017, Proceedings*, volume 10506 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2017. doi:10.1007/978-3-319-67089-8_3.

13 Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 738–749. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.73.

14 Werner Damm. The IO- and oi-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

15 Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CBO9781139236119.

16 Martijn Hendriks and Marcel Verhoef. Timed automata based analysis of embedded system architectures. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 8–pp. IEEE, 2006.

17 Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994. doi:10.1006/INCO.1994.1045.

18 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

19 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

20 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

21 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Information and Compututation*, 239:340–355, 2014. doi:10.1016/J.IC.2014.07.012.

22 Ashutosh Trivedi and Dominik Wojtczak. Recursive timed automata. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings*, volume 6252 of *Lecture Notes in Computer Science*, pages 306–324. Springer, 2010. doi:10.1007/978-3-642-15643-4_23.

23 P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker, Inc., 1997.

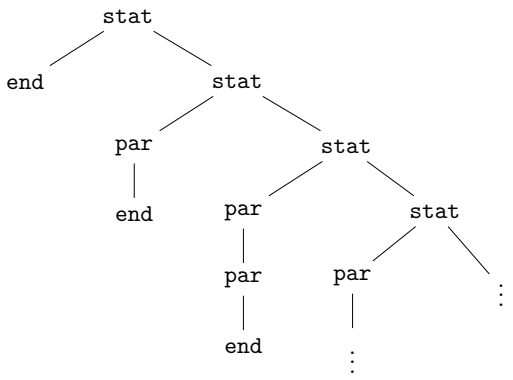
A Additional Material

A.1 An example of a HORTA

Consider the example of a delivery truck that waits at a station to leave for its deliveries. It can either leave, or stay at the station, for example because the driver's mandatory break is not over. Every time the option is presented and the truck stays, however, it is loaded with an additional parcel that will need to be delivered during the next tour of the truck. Waiting at the station takes between 1 and 2 time units, and the driver has to wait for at least 5 minutes. Delivering each parcel will take between 2 or 3 minutes. We ask: is it possible to complete the mandatory waiting time of 5 minutes, but end up with a tour that can be completed in 10 minutes or less?

We model this as follows: Let $\{\texttt{stat}^2, \texttt{par}^1, \texttt{end}^0\}$ be a tree alphabet with arities as indicated. Add a unique proposition p_{end} with $\lambda(p_{\text{end}}) = \{\texttt{end}\}$.

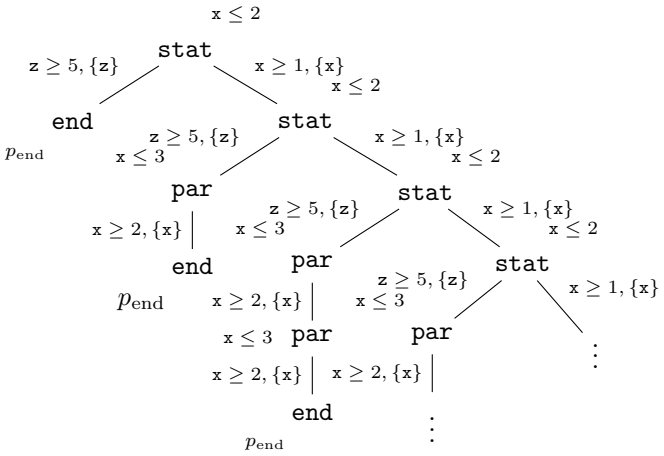
Consider the λY term $t = \big(Y F.\ \lambda x.\ \texttt{stat}\ x\ (F\ (\texttt{par}\ x))\big)\ \texttt{end}$. Note that this term is tail recursive. It defines an infinite tree of the following form, which encodes the different ways waiting and delivering can play out:



We now convert the λY term t into a HORTA by setting $\iota(\texttt{stat}) = \texttt{x} \leq 2$, $\iota(\texttt{par}) = \texttt{x} \leq 3$ and $trns_0(\texttt{stat}) = (\texttt{z} \geq 5, \{\texttt{z}\})$, $trns_1(\texttt{stat}) = (\texttt{x} \geq 1, \{\texttt{x}\})$, $trns_0(\texttt{par}) = (\texttt{x} \geq 2, \{\texttt{x}\})$.

The intuition here is that clock x is used to measure waiting times at the station and delivery times of the individual parcels, while clock z is used to measure the break, and the overall delivery time.

We can visualize the tree like this:



Here, location invariants are drawn next to the nodes, and pairs of transition guards and resets drawn next to the edges. The proposition p_{end} appears next to the final nodes.

A T_μ formula that verifies the property above is $\mu X.\ (p \wedge \texttt{z} \leq 10) \vee (\texttt{tt} \rhd X)$. Note that the dependence between the waiting times and the overall delivery time cannot be incorporated into a finite-state TA due to the strictly nonregular behavior of loading additional parcels. Also note that this formula has bounded boolean alternation. Since the term that gives riseto the HORTA above is also tail recursive, a problem like this can be verified more efficiently, since the

In a real-time recursion scheme (cf. [3]), such a situation could also be modeled, but less naturally. Such a real-time recursion scheme generated trees where the tree constructors are simply annotated by plain intervals, which roughly signals how much time it takes to process such a constructor. No connection to individual clocks is made by the tree. Instead, the tree generated by such a real-time scheme is to be verified against a pair of a TA and an APT, with some synchronization between the two. This means that a significant part of the behavior of the system that is being modeled must be added to this automaton, and great care must be taken to synchronize the APT and he TA. Conversely, in a HORTA, the system being modeled manifests itself almost exclusively in the (recursive) TA, as can be seen above. The formulas that specify the desired property can then be comparatively simple. In the case above, a simple reachability property suffices.

GradSTL: Comprehensive Signal Temporal Logic for Neurosymbolic Reasoning and Learning

Mark Chevallier
School of Engineering, University of Edinburgh, UK

Filip Smola
School of Informatics, University of Edinburgh, UK

Richard Schmoetten
School of Informatics, University of Edinburgh, UK

Jacques D. Fleuriot
School of Informatics, University of Edinburgh, UK

Abstract

We present GradSTL, the first fully comprehensive implementation of signal temporal logic (STL) suitable for integration with neurosymbolic learning. In particular, GradSTL can successfully evaluate any STL constraint over any signal, regardless of how it is sampled. Our formally verified approach specifies smooth STL semantics over tensors, with formal proofs of soundness and of correctness of its derivative function. Our implementation is generated automatically from this formalisation, without manual coding, guaranteeing correctness by construction. We show via a case study that using our implementation, a neurosymbolic process learns to satisfy a pre-specified STL constraint. Our approach offers a highly rigorous foundation for integrating signal temporal logic and learning by gradient descent.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Computing methodologies → Neural networks

Keywords and phrases Signal temporal logic, spatio-temporal reasoning, neurosymbolic learning

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.6

Acknowledgements We thank the referees for their feedback. This research was funded by the Edinburgh Laboratory for Integrated Artificial Intelligence (ELIAI) EPSRC (EP/W002876/1), by the Fonds National de la Recherche, Luxembourg (AFR 15671644), and by the Legal & General Group (research grant to establish the independent Advanced Care Research Centre at University of Edinburgh). Legal & General had no role in conducting the study, interpretation or the decision to submit for publication. The views expressed are those of the authors and not necessarily those of Legal & General. The Artificial Intelligence and its Applications Institute (AIAI) provided support for attending TIME 2025.

1 Introduction

Signal temporal logic (STL) [18] is a formal language suitable for expressing a wide variety of temporal constraints with relevance to control methods [21], robotics [1], and more recently, machine learning [11, 14]. These constraints can be used to describe safety conditions, for example in autonomous vehicles [3]. In neurosymbolic processes, STL can be used to define interpretable objectives or constraints over learning. However, previous implementations of differentiable STL support either only a part of the language or rely on assumptions about the signal it is applied to (for example, assuming uniform time sampling). This fails to achieve the full potential of STL, limiting its applicability.

We introduce GradSTL, the first comprehensive and formally verified implementation of a differentiable semantics of STL expressed over tensors, applicable to any finite signal and suitable for immediate use in neurosymbolic learning. By "comprehensive", we mean a full

implementation of all aspects of the STL language, syntax and full semantics, usable with any signal. Using the Isabelle theorem prover [19], we formalise the STL language and its standard semantics, its robustness and the derivative of its robustness, all in an algorithmic form which recurses over arbitrary signals.

We formally prove both the soundness of the robustness function with respect to STL's standard semantics, and the correctness of the derivative function. GradSTL's specification is comprehensive, allowing for nested temporal constraints, the Until constraint and making no additional assumptions about the signals it is defined over. We have found no previous implementation of STL for learning via gradient descent that has achieved all of these properties (see Section 2.2). To do this, we use a novel *adaptive temporal window* technique to change temporal constraints as they recurse down a signal temporally (see Section 3.1).

From this specification, we automatically generate executable OCaml code, avoiding any unverified manual implementation. This integrates directly with PyTorch-based learning processes, allowing the STL robustness value to be used as a loss function in (for example) a neural network. We demonstrate experimentally that we can then learn behaviours satisfying arbitrary STL constraints.

Our contribution is both theoretical and practical: a sound and comprehensive STL semantics for recursing over irregularly sampled signals; a formally verified, differentiable robustness function; a verified-to-executable implementation pipeline; and empirical validation in a modern machine learning framework. By eliminating ad hoc and unverifiable logic handling, our work offers a rigorous and trustworthy foundation for constrained neural learning and formal specification-guided AI.

This work is an extension of previous work integrating formalised linear temporal logic over finite traces (LTL_f) into neurosymbolic learning [5]. It expands on this previous work substantially, with a novel approach to working with temporal constraints in STL, as well as expanding the applicability of STL to learning as discussed above. The most important innovations over the previous work are a method of evaluating differentiable functions as atomic constraints, and a temporal recursion method required for STL to work well over irregularly sampled time periods (discussed further in Section 3.1). The current work also demonstrates how the same pipeline of formal verification methods and code generation can be successfully used with a substantially more complicated logic.

1.1 Organisation of the paper

In the next section, we briefly introduce STL and discuss the background to our work. Then, in Section 3, we present our formal specification of a recursive algorithm to evaluate STL and statements of its formally proven properties. We also detail how we were able to overcome previous limitations using the adaptive temporal window technique. We also briefly describe the automatic code generation for our algorithms and how to integrate them into a learning process. In Section 4, we consider a case study showing the comprehensiveness of our implementation, concentrating specifically on irregular sampling. Lastly, in Section 5, we discuss the significance of our work.

2 Background

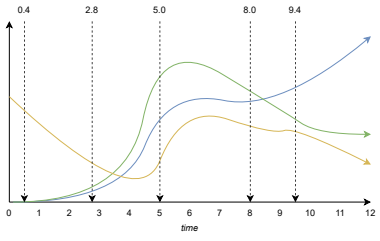
In this section, we introduce signal temporal logic before going on to examine previous work integrating it with neurosymbolic processes.

2.1 Signal temporal logic: a brief overview

STL is a formal language intended to make a statement ρ about some continuous function of time C [18]. This function C can model anything that changes over time: performance in an engine, the location of a vehicle, or health statistics of a patient, for example. The state at time t, $C(t)$, is typically a vector in $\mathbb{R}^m$ of state variables $(v_0, v_1, \ldots, v_{m-1})$ representing measures of interest – from our examples, perhaps these measures are the heat in the fuel chamber, the x-coordinate of a path, or the systolic blood pressure of a patient. This vector is called the *sample* at time t of C.

In this paper, we examine two ways of assigning meaning to an STL constraint: the *standard* boolean semantics and the *robustness* (see below). The standard semantics for STL evaluates a statement ρ as being either satisfied (*true*) or unsatisfied (*false*). Such STL statements are often called *constraints* as they describe conditions that might be breached or respected by C. This evaluation takes place in the context of a temporal *signal* which summarises C.

A signal $S_{C,T} = (C(t_0), C(t_1), \ldots, C(t_{n-1}))$ is a vector of samples of C where T is a vector of *time indices* in $\mathbb{R}^n$, $(t_0, t_1, \ldots, t_{n-1})$, ordered so that for all $i < j$ we have $t_i < t_j$. These time indices t_i hold the time when each sample was taken, and can be arbitrarily close together or far apart. We typically drop the subscripts C, T where there is no ambiguity. As S is a vector of vectors, we can represent it as a matrix (a two dimensional tensor). Figure 1 shows how a signal samples an underlying function of three state variables.



$$S = \begin{bmatrix} t & x & y & z \\ 0.0 & 0.0 & 0.0 & 25.0 \\ 0.4 & 0.1 & 0.1 & 20.6 \\ 2.8 & 2.0 & 2.4 & 8.1 \\ 5.0 & 18.4 & 28.6 & 8.2 \\ 8.0 & 24.7 & 26.1 & 17.9 \\ 9.4 & 26.9 & 18.2 & 17.0 \end{bmatrix} \text{Time} \downarrow$$

Figure 1 Example of a signal sampling three state variables x (in blue), y (in green), and z (in yellow), at times 0.0, 0.4, 2.8, 5.0, 8.0 and 9.4. An illustration of sampling is on the left, and the signal in matrix form is on the right.

The precise time indices used for the samples can make a particular constraint true or false, even over the same underlying function C. Thus, effective sampling is critical to ensure that the signal reflects C sufficiently well for the constraint to be useful.

In the remainder of this paper, we will use the notation $t \leftarrow S$ to express that t is a time at which signal S is sampled, in other words, that t is a component of the time vector T used by S. We define $|S|$ as the total number of samples in the signal S.

We define $S_n = C(t_n)$ if $t_n \leftarrow S$. We denote the standard semantic evaluation of a constraint ρ at time $t_n \leftarrow S$ over signal S by $E_S(\rho, n)$ and assume that $n < |S|$, otherwise E_S is undefined.

As well as the standard semantics, STL also can be evaluated via a *robustness* semantics, which is also considered over a signal. This returns a real value whose magnitude corresponds, roughly speaking, to how much the signal would need to be changed for the constraint's truth value to change from true to false (or vice versa). If robustness is negative for some constraint ρ, then ρ is false. If it is positive, then ρ is true. If robustness is exactly 0 for ρ, we do not know if it is true or false – it is on the cusp and might be either. We denote the robustness of a constraint ρ at time $t_n \leftarrow S$ by $R_S(\rho, n)$ and, as for E_S, we assume $n < |S|$.

Syntactically, STL constraints are represented by logical formulae. The smallest constraint possible is an atomic one which, when evaluated at sample n, compares $f(S_n)$ to some constant c, where f is some differentiable real-valued function. We denote these constraints by $\mu_{f,c}$ whose semantics depends on the function f and constant c. For example, if we are monitoring the location of a robot, and we can extract its x and y coordinates from S_n, we can calculate its distance from the origin using $f(x, y) = \sqrt{x^2 + y^2}$ to evaluate if it has travelled past some boundary distance. The semantics of an atomic constraint is then defined as follows:

$$E_S(\mu_{f,c}, n) ::= f(S_n) > c$$

Building on this, the STL formulae used to capture the more general constraints are:

$$\rho ::= \mu_{f,c} \mid \neg\rho \mid \rho_1 \wedge \rho_2 \mid \Box_{[x,y]}\rho \mid \Diamond_{[x,y]}\rho \mid \rho_1 \,\mathcal{U}_{[x,y]}\, \rho_2$$

The first three are understood as per the syntax and semantics of propositional logic, with $\mu_{f,c}$ being the atomic constraint (proposition), and $\neg\rho$ and $\rho_1 \wedge \rho_2$ representing negation and conjunction, respectively. Disjunction can be represented by $\neg(\neg\rho_1 \wedge \neg\rho_2)$.

The remaining formulae introduce various temporal constraints. Each of these has a subscript $[x, y]$, with $0 \leq x \leq y$, which shows that when the constraint is evaluated at a sample with time index t, it applies at times sampled between $t + x$ and $t + y$. These can be understood as follows:

Always

$$E_S(\Box_{[x,y]}\rho, n) ::= \forall t_i \leftarrow S.\ t_n + x \leq t_i \leq t_n + y \implies E_S(\rho, i)$$

In other words, when evaluated at a sampled time t_n, $\Box_{[x,y]}\rho$ is true if and only if ρ is true at all sampled times between and including $t_n + x$ and $t_n + y$.

Eventually

$$E_S(\Diamond_{[x,y]}\rho, n) ::= \exists t_i \leftarrow S.\ t_n + x \leq t_i \leq t_n + y \wedge E_S(\rho, i)$$

When evaluated at a sampled time t_n, $\Diamond_{[x,y]}\rho$ is true if and only if ρ is true at some sampled time between and including $t_n + x$ and $t_n + y$.

Until

$$E_S(\rho_1 \,\mathcal{U}_{[x,y]}\, \rho_2, n) ::= \exists t_i \leftarrow S.\ t_n + x \leq t_i \leq t_n + y \wedge E_S(\rho_2, i) \wedge \big(\forall t_j \leftarrow S.\ t_n + x \leq t_j \leq t_i \rightarrow E_S(\rho_1, j)\big)$$

When evaluated at a sampled time t_n, $\rho_1 \,\mathcal{U}_{[x,y]}\, \rho_2$ is true if and only if ρ_2 is true at some time between $t_n + x$ and $t_n + y$, and ρ_1 is true for all sampled times between and including $t_n + x$ and whenever ρ_2 is first true.

Our treatment of STL is not minimal – we could define the language without expressly including the $\Box_{[x,y]}\rho$ and $\Diamond_{[x,y]}\rho$ as primitive constraints, instead using $\neg(\top \,\mathcal{U}_{[x,y]}\, \neg\rho)$ and $\top \,\mathcal{U}_{[x,y]}\, \rho$, respectively (where $\top$ is True which can be defined to be any tautology). We provide these constraints and their semantics explicitly to allow more efficient code generation (see Section 3.3).

Robustness is calculated using the $R_S(\rho, n)$ function, which returns a real value that, as noted above, corresponds to how well the constraint is satisfied over the signal. Its definition over our STL syntax is as follows:

- $R_S(\mu_{f,c}, n) = f(S_n) - c$
- $R_S(\neg\rho, n) = -R_S(\rho, n)$
- $R_S(\rho_1 \wedge \rho_2, n) = \min\{R_S(\rho_1, n),\ R_S(\rho_2, n)\}$
- $R_S(\Box_{[x,y]}\rho, n) = \min\{R_S(\rho, i) : i < |S| \wedge t_n + x \leq t_i \leq t_n + y\}$
- $R_S(\Diamond_{[x,y]}\rho, n) = \max\{R_S(\rho, i) : i < |S| \wedge t_n + x \leq t_i \leq t_n + y\}$
- $R_S(\rho_1 \mathcal{U}_{[x,y]}\rho_2, n) =$

$$\max\Big\{ \min\big\{R_S(\rho_2, i), \min\{R_S(\rho_1, j) : j < |S| \wedge t_n + x \leq t_j \leq t_n + t_i\}\big\} : i < |S| \wedge t_n + x \leq t_i \leq t_n + y\Big\}$$

Examining these sub-definitions, one can prove by induction, through both the constraint structure and the temporal dimension, that whenever $R_S(\rho, n) > 0$, $E_S(\rho, n)$ is true, and that whenever $R_S(\rho, n) < 0$, $E_S(\rho, n)$ is false [18]. When $R_S(\rho, n) = 0$, it also holds that $R_S(\neg\rho, n) = 0$, so this tells us nothing about $E_S(\rho, n)$. We call this R_S *sound* with respect to E_S. Soundness is one of many useful (in this case, essential) properties a robustness can satisfy [8]. In Section 3.2, we give an algorithmic specification of a semantics similar to the above, but which is differentiable and recurses efficiently.

The definition of R_S can be derived from that of E_S following some simple principles. The R_S rule for the atomic constraint is fundamental and clearly follows from the equivalent rule for E_S (likewise for negation). For the others, we take minimums of sets of R_S values to represent the universal quantifier over (resp. conjunctions between) their corresponding E_S truth values, and maximums to represent the existential quantifier (resp. disjunction). If we conjoin two values, and one is false – implying that its R_S value is non-positive – the result will also be non-positive (similar reasoning for disjunctions).

It is important to note that these definitions given for E_S and R_S cannot easily be translated to an efficient, recursive algorithm that computes its value over a signal. GradSTL uses provably equivalent definitions of these functions that give us recursive algorithms that compute over arbitrary signals efficiently. We discuss the details in Sections 3.1 and 3.2.

2.2 Previous work on neurosymbolic integration of STL

There are several previous approaches to integrating STL with neurosymbolic learning. Our work is aimed at using an arbitrary STL constraint to learn signal outputs which satisfy that constraint. Not all neurosymbolic work using STL has the same goal, and, to our knowledge, no previous neurosymbolic approach provides a comprehensive implementation of STL against arbitrary signals. We next review previous work in this area, assessing each approach in terms of its goal and how comprehensively it implements STL.

Leung et al. worked on predicting a STL constraint (given a fixed structure) would best satisfy a given signal [13]. The goal of this work was to learn the numeric parameters for the fixed STL constraint using neural methods and a parametric STL approach [4]. Computation graphs implemented through the PyTorch library were fundamental to this work, which assumed that any signal was uniformly sampled.

The use of computation graphs is also a hallmark of later work by the same authors on STLCG [14], and work on its successor STLCG++ [11]. Of the work discussed in this section, these two methods are the closest to our own – aimed at providing a way to use satisfaction of an STL constraint in learning some temporal output. Backpropagation using these computation graphs is key to how they enable learning. However, STLCG and its successor both assume uniform time sampling across the signal, which means they can only be applied to a proper subset of signals. In contrast, as discussed in Section 3.1, our own work can be applied to signals with any sampling.

Yan et al. worked on STONE, a method aimed at learning weighted STL subconstraints that efficiently classify time series [23]. Rather than using an STL constraint to help learn a temporal output, this work uses a temporal input (a time series) to learn the weights for a STL constraint's subconstraints, to help interpret and classify subsequent time series. Each subconstraint is represented as an individual neuron within a network, integrating the constraint with the neural process. However, the STL implemented by STONE is only a fragment of STL – it omits the important Until constraint, and requires its initial structure to be configured ahead of time; only the weights are learned.

X. Li et al. have enhanced the work on STL constraint classification, but their approach retains the limitation of only considering a fragment of STL, ignoring the Until constraint [15]. This inability to account for Until is shared by the work on differentiable logic layers by D. Li et al. [16]. Again, aimed at learning an STL constraint using temporal inputs, this uses a layer of the neural network to represent the STL constraint itself, adjusting the layer's weights to represent how the constraint might function. As these methods use the structure of the neural network to learn the constraint, what they learn is limited by said structure but also the size of the neural network. They cannot learn general constraints.

Liu et al. use earlier work on BarrierNets [22] to guarantee that the output of a given process will satisfy an STL constraint [17]. These approaches have a similar goal to our own work, but the use of BarrierNets provides a guarantee that a given STL constraint will be satisfied by modifying the neural network's output. Again, this uses a fragment of STL ignoring the Until constraint, and also enforces restrictions on the domain it can work with: those with safety methods involving reaching or avoiding circular regions.

As can be seen, no previous work in this area has managed to deal with a fully comprehensive treatment of STL. Either the language is limited (excluding the Until constraint or lacking nested temporal constraints), or its applicability relies on significant assumptions (working only with uniformly sampled signals). It is precisely these limitations GradSTL overcomes.

Outside of work applicable to neurosymbolic learning, other STL monitors like Breach [7] and S-TaLiRo [2] can work with irregularly sampled signals, but unlike GradSTL, they find a non-differentiable robustness and are only used for falsification. They are also coded by hand, leading to the potential for error.

3 Formal Semantics for GradSTL

We next discuss the details of our algorithmic specification of STL and its properties in two sections: first we discuss our work specifying an algorithm for STL semantics in the Isabelle theorem prover, presenting a simple algorithm to compute its boolean semantics and showing how, from this, we can derive a function for its robustness that is differentiable. Secondly, we briefly review code generation and how our formalisation feeds into it.

3.1 Recursive algorithm for the standard semantics of STL

Notation

S_n	Sample n in signal S, expressed as a vector of state variables
$\|S\|$	The total number of temporal samples in S
$t \leftarrow S$	Time t, where S contains a sample at time t
Δt_n	$t_{n+1} - t_n$ (time gap between sample n and $n+1$)

We now examine how to algorithmically evaluate the semantics of an STL constraint over a signal. We specify the function $E^*_S(\rho, n)$ for this purpose, where S is a signal, ρ is the constraint being evaluated, and n a natural number indexing the signal's samples. This algorithm is structured such that $E^*_S(\rho, n) = E_S(\rho, n)$. The evaluation proceeds by recursion through the constraint ρ. The propositional component of the constraint proceeds exactly as in Section 2.1:

- $E^*_S(\mu_{f,c}, n) ::= f(S_n) > c$
- $E^*_S(\neg\rho, n) ::= \neg\big(E^*_S(\rho, n)\big)$
- $E^*_S(\rho_1 \wedge \rho_2, n) ::= E^*_S(\rho_1, n) \wedge E^*_S(\rho_2, n)$

The temporal constraints are more complicated. In addition to recursion through the constraint itself, E^*_S may recurse down the signal's temporal dimension to evaluate at a later time. The key to correctly evaluating temporal constraints is the *adaptive temporal window*, which changes as temporal recursion takes place. After each temporal position n, if $n < |S| - 1$, we subtract Δt_n from the temporal window, where $\Delta t_n = t_{n+1} - t_n$. This technique allows for nesting of temporal constraints to any depth, and the full expressiveness of STL over arbitrary signals. This overcomes the limitations of previous attempts that did not use this method at applying STL to neurosymbolic learning.

We illustrate this using $E^*_S(\Diamond_{[x,y]}\rho, n)$. Informally, this is true if and only if:

Case 1: $x \leq 0 \leq y \wedge E^*_S(\rho, n)$ is true. The first part of this case ($x \leq 0 \leq y$) checks if the temporal window contains 0. If so, it means that if the second condition ($E^*_S(\rho, n)$) is true, we can confirm that the Eventually constraint has been satisfied (as defined in Section 2.1: $\exists t_i \leftarrow S.\ t_n + x \leq t_i \leq t_n + y \wedge E_S(\rho, i)$). If and only if this case is not true, then we check Case 2.

Case 2: $E^*_S(\Diamond_{[x-\Delta t_n, y-\Delta t_n]}\rho, n+1)$ is true. In other words, we re-evaluate at the next temporal position. The adaptive temporal window for the constraint changes, subtracting how much time difference there is between positions n and $n+1$ (Δt_n). If this changes the temporal window so that it contains 0, we know that the time index for the current position is within the temporal window, so Case 1 above might become true.

We also perform certain boundary checks as we recurse, ensuring that $n < |S|$, i.e. that we do not pass the final temporal position in the signal, leaving the function undefined if we breach this boundary check. We also terminate temporal recursion if we would otherwise exit the temporal window (as once $y < 0$, we know that there are no more samples that could satisfy the constraint).

For example, consider an STL constraint of $\Diamond_{[5,10]}\mu_{v,20}$. This is true at time index t in signal S if and only if $\exists t' \leftarrow S.\ t + 5 \leq t' \leq t + 10$ where $v > 20$ is true. The adaptive temporal window begins as $[5, 10]$, but changes as it recurses to capture when it is in the correct position *relative to the point of its first evaluation*. This is illustrated in Figure 2.

The full definitions of E^* for the temporal constraints, including the boundary checks, are given below:

$$E^*_S(\Diamond_{[x,y]}\rho, n) ::= \begin{cases} x \leq 0 \wedge E^*_S(\rho, n) & \text{if } n = |S| - 1 \vee \\ & \quad y - \Delta t_n < 0 \\ \big(x \leq 0 \wedge E^*_S(\rho, n)\big) \vee & \text{if } n < |S| - 1 \\ \quad E^*_S(\Diamond_{[x-\Delta t_n, y-\Delta t_n]}\rho, n+1) & \end{cases}$$

$$
\begin{array}{r}
 \\
E^*_S(\Diamond_{[5,10]}\mu_{v,20}, 0) \longrightarrow \\
E^*_S(\Diamond_{[2.7,7.7]}\mu_{v,20}, 1) \longrightarrow \\
E^*_S(\Diamond_{[1.1,6.1]}\mu_{v,20}, 2) \longrightarrow \\
E^*_S(\Diamond_{[-2.7,2.3]}\mu_{v,20}, 3) \longrightarrow \\
E^*_S(\Diamond_{[-4.1,0.9]}\mu_{v,20}, 4) \longrightarrow \\
 \\
\end{array}
\begin{array}{c}
\begin{array}{cc} t & v \end{array} \\
\left[\begin{array}{cc}
0.0 & 1.6 \\
2.3 & 1.9 \\
3.9 & 12.0 \\
7.7 & 15.3 \\
9.1 & 14.2 \\
11.4 & 28.2
\end{array}\right]
\end{array}
\begin{array}{c}
\Delta t \\
\\
2.3 \\
1.6 \\
3.8 \\
1.4 \\
2.3 \\
\cdots
\end{array}
$$

Figure 2 How $E^*_S(\Diamond_{[5,10]}\mu_{v,20}, 0)$ changes as it temporally recurses over S. At each recursion, the value Δt is subtracted from the adaptive temporal window. When the temporal window contains 0, $\mu_{v,20}$ is evaluated. Temporal recursion terminates early at position 4, as the next recursion would breach a boundary condition. In this example, $v > 20$ only after the temporal recursion terminates, so $E^*_S(\Diamond_{[5,10]}\mu_{v,20}, 0)$ is false.

$$
E^*_S(\Box_{[x,y]}\rho, n) ::= \begin{cases} x \leq 0 \wedge E^*_S(\rho, n) & \text{if } n = |S| - 1 \vee \\ & \quad y - \Delta t_n < 0 \\ \big(x > 0 \vee E^*_S(\rho, n)\big) \wedge & \text{if } n < |S| - 1 \\ \quad E^*_S(\Box_{[x-\Delta t_n, y-\Delta t_n]}\rho, n+1) & \end{cases}
$$

$$
E^*_S(\rho_1 \mathcal{U}_{[x,y]} \rho_2, n) ::= \begin{cases} x \leq 0 \wedge E^*_S(\rho_1 \wedge \rho_2, n) & \text{if } n = |S| - 1 \vee \\ & \quad y - \Delta t_n < 0 \\ \Big(\big(x > 0 \vee E^*_S(\rho_1, n)\big) \wedge & \text{if } n < |S| - 1 \\ \quad E^*_S(\rho_1 \mathcal{U}_{[x-\Delta t_n, y-\Delta t_n]}\rho_2, n+1)\Big) \vee & \\ \quad \big(x \leq 0 \wedge E^*_S(\rho_1 \wedge \rho_2, n)\big) & \end{cases}
$$

Each time that $E^*(\rho, n)$ recurses, it either reduces the size of the STL constraint or approaches the signal termination by a single step. Isabelle proves that the lexicographic measure $\langle |S| - n, |\rho| \rangle$ strictly decreases with every recursion (where $|\rho|$ is the number of sub-formulae in ρ). Thus, E^* runs in $\mathcal{O}(|S||\rho|)$ time. This is also true of R^* and dR^*, described in Section 3.2.

The E^* function is formally specified in the Isabelle theorem prover. It is equivalent to the standard semantics for STL given in Section 2.1. The E^* function thus allows us to algorithmically evaluate constraints with STL's standard semantics over a signal by recursion.

3.2 Recursive algorithm for smooth robustness semantics of STL

In an analogous way to how we implement the evaluation of the standard semantics, we now address formalising a robustness function. Recall that this is a function describing to what degree a signal does or does not satisfy a constraint. This makes it useful as a loss function in learning to satisfy the constraint, as it can be treated as an error. In order to enable learning by gradient descent, this robustness function should also be differentiable.

Neither the minimum nor the maximum function used extensively in the usual robustness function (as described in Section 2.1) are differentiable everywhere. Because of this, we implement smooth, binary versions of these functions, $\max_\gamma$ and $\min_\gamma$, which use γ as a smoothing parameter [6]. Their definitions are as follows:

$$\max_\gamma(a,b) = \begin{cases} \max\{a,b\} & \text{if } \gamma \leq 0 \\ \gamma \ln(e^{a/\gamma} + e^{b/\gamma}) & \text{if } \gamma > 0 \end{cases}$$

$$\min_\gamma(a,b) = \begin{cases} \min\{a,b\} & \text{if } \gamma \leq 0 \\ -\max_\gamma(-a,-b) & \text{if } \gamma > 0 \end{cases}$$

We prove in Isabelle that $\lim_{\gamma \to 0}(\max_\gamma(a,b)) = \max\{a,b\}$, and likewise for $\min_\gamma$. Importantly, for $\gamma > 0$ both functions are differentiable everywhere with respect to either parameter a or b.

We now specify the differentiable function $R^*_{\gamma,S}(\rho,n)$, used to compute the robustness of ρ recursively over a signal S from temporal position n, with a smoothness parameter γ as used above. This function is equivalently referred to as a *smooth semantics* for STL.

We derive the R^* function from E^* following similar guidelines as those used in deriving the standard robustness function R from the standard semantics E (discussed in Section 2.1). We replace disjunctions in E^* with $\max_\gamma$ in R^*, and conjunctions with $\min_\gamma$. Where E^* checks if $x \leq 0$ we simply use $-x$ (and x if checking $x > 0$). We give an example of this translation below using the Always constraint:

$$R^*_{\gamma,S}(\square_{[x,y]}\rho, n) = \begin{cases} \min_\gamma\big(-x, R^*_{\gamma,S}(\rho,n)\big) & \text{if } n = |S|-1 \vee \\ & \quad y - \Delta t_n < 0 \\ \min_\gamma\Big(\max_\gamma\big(x, R^*_{\gamma,S}(\rho,n)\big), & \text{if } n < |S|-1 \\ \qquad R^*_{\gamma,S}(\square_{[x-\Delta t_n, y-\Delta t_n]}\rho, n+1)\Big) & \end{cases}$$

We now have a function that gives us an algorithm to compute robustness. As already mentioned, robustness can tell us how well a constraint is satisfied, or how badly it is unsatisfied. This second use can be adapted as a loss function. But we need to go further to meet the needs of learning that uses gradient descent, by having an algorithm to compute the derivative of robustness with respect to any state variable of the signal.

We proceed to build this derivative function $dR^*_{\gamma,S}(\rho,n,v_{i,k})$, which calculates the derivative with respect to $v_{i,k}$, the i^{th} state variable at temporal position k. We derive dR^* using the chain derivative rule and the derivatives for $\max_\gamma$, $\min_\gamma$ and the atomic constraint. The derivatives for $\max_\gamma$ (and $\min_\gamma$) take four parameters: the values being compared and their derivatives. For example, the function $\max_\gamma(a,b)$ has a derivative calculated using the function $\text{dmax}_\gamma(a,da,b,db,v_{i,k})$, where da and db are the derivatives (with respect to $v_{i,k}$) for the functions used to calculate a and b. We formally prove with Isabelle that this derivative is correct under the assumption that da and db are correct.

We present an illustration of how dR^* is defined using the Always constraint as an example (compare with the R^* definition above):

$$dR^*_{\gamma,S}(\square_{[x,y]}\rho,n,v_{i,k}) = \begin{cases} \text{dmin}_\gamma\big(-x, 0, & \text{if } n = |S|-1 \vee \\ \quad R^*(\rho,n), dR^*(\rho,n,v_{i,k})\big) & \quad y - \Delta t_n < 0 \\ \text{dmin}_\gamma\Big(\max_\gamma\big(x, R^*_{\gamma,S}(\rho,n)\big), & \text{if } n < |S|-1 \\ \quad \text{dmax}_\gamma\big(x, 0, R^*_{\gamma,S}(\rho,n), dR^*_{\gamma,S}(\rho,n,v_{i,k}), v_{i,k}\big), & \\ \quad R^*_{\gamma,S}(\square_{[x-\Delta t_n, y-\Delta t_n]}\rho, n+1), & \\ \quad dR^*_{\gamma,S}(\square_{[x-\Delta t_n, y-\Delta t_n]}\rho, n+1, v_{i,k}), v_{i,k}\Big) & \end{cases}$$

We can then use dR^* with any gradient descent method to learn how that variable can be changed to maximise robustness.

As with the E^* function, both the R^* function and its derivative dR^* are formally specified in the Isabelle theorem prover. We use Isabelle to formally prove two important theorems about these functions, namely the soundness and the correctness of the derivative:

▶ **Theorem 1** (Soundness).

$$\lim_{\gamma \to 0} R^*_{\gamma,S}(\rho, n) > 0 \implies E^*_S(\rho, n)$$

$$\lim_{\gamma \to 0} R^*_{\gamma,S}(\rho, n) < 0 \implies \neg E^*_S(\rho, n)$$

▶ **Theorem 2** (Derivative correctness). *Assuming* $\gamma > 0$,

$$\frac{d\big(R^*_{\gamma,S}(\rho, n)\big)}{dv_{i,n}} = dR^*_{\gamma,S}(\rho, n, v_{i,n})$$

The proofs of both these theorems proceed by induction on the size of the constraint and the size of the signal being examined. In the base case we establish the conclusion for the simplest constraint at a single time index. In the inductive step we extend the conclusion from smaller constraints and shorter segments of the signal to increasingly larger constraints and segments. Both of these directions of induction are vital to cover both the propositional and temporal aspects of the logic. Initially, we establish the soundness without smoothing (i.e. $\gamma = 0$) and expand this result to $\gamma > 0$ by using continuity of R^*.

The full formal proofs of these facts, which are part of an Isabelle formalisation that is too extensive to cover in this article, provide us with strong, *a priori* guarantees about our overall approach.

3.3 Code generation

We use the code generation capabilities of Isabelle to export our specifications as working OCaml code [9]. This means that the structure of our formal definitions is translated via a thin layer of trusted equivalences into OCaml code, without additional human intervention. This translation layer is small and comes with partial correctness proofs [10], and so we have strong confidence that the proofs we have established for our Isabelle specifications remain valid for the code that is generated.

As we discussed in Section 2.1, our definition of STL is not minimal. We include the Eventually and Always constraints as primitives instead of defining them via the Until constraint. This means our generated code is more efficient, as our approach prevents a complicated chain of constraints slowing down calculations during code execution.

The code is then called from an instance of the `torch.autograd.Function` PyTorch class, with R^* as its forward method, and dR^* as its backwards method [5, 20]. This can then be handled like any other function used by PyTorch for optimisation via gradient descent, and incorporated into neurosymbolic learning.

4 Case Study

We replicated many of the experiments from the STLCG paper [14], then extended these with one that uses an irregularly sampled signal (which is out of scope for STLCG). In the latter, described below, we demonstrate how an agent can learn to respect pre-specified (spatio-)temporal rules in a complicated and irregularly sampled environment.

Consider this setting: a medical robot, assisting a disabled patient in a single room, which must perform certain tasks in a 50 second time limit while also avoiding risks to the patient and to the structure of the room. All tasks begin with the robot in a docking position. The specific task we use in our demonstration requires that the robot go from the docking position to a medical cabinet, retrieve some medicine, and then administer it to the patient in bed, before returning to its dock. To complete the retrieval and administration tasks the robot must remain in the correct area for 5 seconds (we do not specify the exact requirements of these fine motor control tasks as we are only concerned with navigation in our problem). At all times, the robot must be careful to avoid furniture (the chair, a desk, and the bed), and not to exceed the speed limit v_m.

We represent the task by the following STL constraint, annotated with an informal interpretation for each of the clauses that need to be simultaneously satisfied:

$$
\begin{aligned}
&\Box_{[0.0,50.0]}\mu_{-v,-v_m} \wedge && \text{The whole time follow the speed limit, } v < v_m \text{ and}\\
&\Box_{[0.0,50.0]}\neg\textbf{Desk} \wedge && \text{avoid the desk area for the whole time period, and}\\
&\Box_{[0.0,50.0]}\neg\textbf{Chair} \wedge && \text{likewise avoid the chair area, and}\\
&\Box_{[0.0,50.0]}\neg\textbf{Bed} \wedge && \text{likewise avoid the bed area, and}\\
&\Diamond_{[0.0,50.0]}\Big(\Box_{[0.0,5.0]}\textbf{Access} \wedge && \text{access the cabinet for 5s to gather medicine, and}\\
&\Diamond_{[0.0,50.0]}\big(\Box_{[0.0,5.0]}\textbf{Bedside} \wedge && \textit{then} \text{ go to bedside for 5s to administer medicine, and}\\
&\Diamond_{[0.0,50.0]}(\Box_{[0.0,50.0]}\textbf{Dock})\big)\Big) && \textit{then} \text{ return to the dock for any remaining time.}
\end{aligned}
$$

For our speed limit, recall that at any sample n with velocity v_n, $E^*(\mu_{-v,-v_m}, n) \iff -v_n > -v_m \iff v_n < v_m$. Each constraint in boldface above is an abbreviation for a conjunction of atomic constraints on the x, y co-ordinates, which are satisfied if the robot is within a specific region (either rectangular or circular).

Planning the trajectory to finish each task involves plotting a path through the room. We can sample from this path to produce a signal, but this sampling need not be uniform. Physical constraints (disruption to transmission, for example) may introduce randomness to the sampling times. Additionally we may wish to sample more often when there is a higher risk of breaching a constraint, and less often when we are in safer areas.

The trajectory to learn is initialised as a set of straight lines going from each task location to the next, ignoring possible risks and time-requirements for tasks, and assuming uniform travel speed. The *initial signal* is found from this path by sampling fifty times, each time taking two state variables directly from the path, the (x, y) co-ordinates. A third state variable is computed from these and the time indices of the samples, representing the speed v. We introduce non-uniform sampling by sampling more frequently where there is higher risk of constraint breach – in particular in the second section of the path between the cabinet and the bed. We know the fixed room layout, so we can determine where this risk is highest by seeing where the initial path crosses through obstacles most frequently.

The STL constraint we are learning from is not simple: it involves extracting different kinds of data in a path, and nesting temporal constraints. In addition, the signal is sampled irregularly and more frequently where we suspect the path has higher risk, as discussed above. Nonetheless, as GradSTL is formally verified, we have confidence that the signal will be correctly evaluated by it, and this result is what we see in Figure 3.

GradSTL computes the robustness of the signal and its derivative for each state variable in the signal, using the STL constraint above. This is then optimised using the Adam optimiser to adjust the signal [12]. This process is repeated 500 times. The difference between the initial and the result signals, illustrated in Figure 3, clearly demonstrates that the optimisation process has learned to satisfy the constraint.

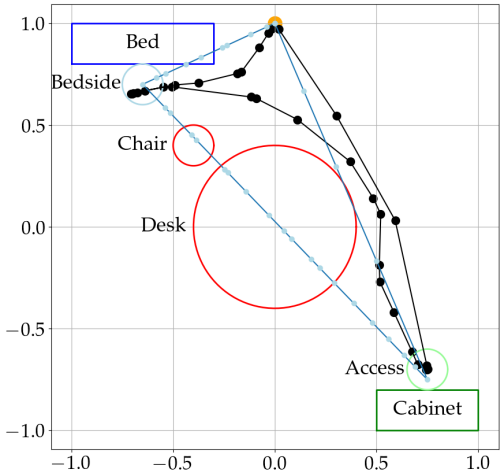


Figure 3 Result showing the agent accessing specific points and navigating the obstacles in the environment. The light blue line is the unaltered original path, with dots indicating when samples were taken for the initial signal. The black line and dots are the trained final path and signal.

5 Conclusion

To our knowledge, GradSTL is the first implementation of STL suitable for neurosymbolic learning over any signal. Our formally verified approach using theorem proving gives great confidence that it is both comprehensive and sound with respect to the standard semantics of STL. Experimental work in our case study demonstrates that it enables successful learning even with complicated constraints and irregular sampling of the signal.

GradSTL overcomes the limitations of previous work, namely implementations of STL which restrict the class of supported constraints or signals. This is possible because our adaptive temporal window technique adjusts STL constraints during temporal recursion. This allows us to identify the correct times that need to be evaluated by any temporal constraint, regardless of when it is first evaluated and whether it is nested.

GradSTL uses a formally specified, recursive algorithm for smooth STL semantics to automatically generate faithful code. Neurosymbolic AI is a growing research area, and finding ways to directly integrate logic and domain knowledge into learning will be of increasing importance. GradSTL is a step in that direction.

References

1 E Abele, F Haehn, M Pischan, and F Herr. Time optimal path planning for industrial robots using STL data files. *Procedia Cirp*, 55:6–11, 2016. doi:10.1016/j.procir.2016.08.038.

2 Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. [s-taliro: A tool for temporal logic falsification for hybrid systems]. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257. Springer, 2011. doi:10.1007/978-3-642-19835-9_21.

3 Nikos Arechiga. Specifying safety of autonomous vehicles in signal temporal logic. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 58–63. IEEE, 2019. doi:10.1109/IVS.2019.8813875.

4 Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers 2*, pages 147–160. Springer, 2012. doi:10.1007/978-3-642-29860-8_12.

5 Mark Chevallier, Filip Smola, Richard Schmoetten, and Jacques D. Fleuriot. Formally verified neurosymbolic trajectory learning via tensor-based linear temporal logic on finite traces, 2025. *arXiv preprint arXiv:2501.13712*. doi:10.48550/arXiv.2501.13712.

6 Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 894–903. PMLR, 2017. doi:10.48550/arXiv.1703.01541.

7 Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pages 167–170. Springer, 2010. doi:10.1007/978-3-642-14295-6_17.

8 Thomas Flinkow, Barak A. Pearlmutter, and Rosemary Monahan. Comparing differentiable logics for learning with logical constraints. *Sci. Comput. Program.*, 244:103280, 2025. doi:10.1016/j.scico.2025.103280.

9 Florian Haftmann and Lukas Bulwahn. Code generation from Isabelle/HOL theories, 2021. Found online at https://isabelle.in.tum.de/doc/codegen.pdf, last checked April 24 2025.

10 Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In Matthias Blume, Naoki Kobayashi, and Germán Vidal, editors, *Functional and Logic Programming*, pages 103–117, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-12251-4_9.

11 Parv Kapoor, Kazuki Mizuta, Eunsuk Kang, and Karen Leung. STLCG++: A Masking Approach for Differentiable Signal Temporal Logic Specification, 2025. doi:10.48550/arXiv.2501.04194.

12 Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. doi:10.48550/arXiv.1412.6980.

13 Karen Leung, Nikos Aréchiga, and Marco Pavone. Backpropagation for parametric STL. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 185–192. IEEE, 2019. doi:10.1109/IVS.2019.8814167.

14 Karen Leung, Nikos Aréchiga, and Marco Pavone. Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. *The International Journal of Robotics Research*, 42(6):356–370, 2023. doi:10.1177/02783649221082115.

15 Danyang Li, Mingyu Cai, Cristian-Ioan Vasile, and Roberto Tron. TLINet: Differentiable Neural Network Temporal Logic Inference, 2024. *arXiv preprint arXiv:2405.06670*. doi:10.48550/arXiv.2405.06670.

16 Xiao Li, Guy Rosman, Igor Gilitschenski, Jonathan A. DeCastro, Cristian Ioan Vasile, Sertac Karaman, and Daniela Rus. Differentiable logic layer for rule guided trajectory prediction. In Jens Kober, Fabio Ramos, and Claire J. Tomlin, editors, *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pages 2178–2194. PMLR, PMLR, 2020. URL: https://proceedings.mlr.press/v155/li21b.html.

17 Wenliang Liu, Wei Xiao, and Calin Belta. Learning robust and correct controllers from signal temporal logic specifications using BarrierNet. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 7049–7054. IEEE, 2023. doi:10.1109/CDC49753.2023.10383857.

18 Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 152–166. Springer, 2004. doi:10.1007/978-3-540-30206-3_12.

19 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002. doi:10.1007/3-540-45949-9.

20 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, volume 32, pages 8024–8035, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

21 Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pages 239–248, 2015. doi:10.1145/2728606.2728628.

22 Wei Xiao, Ramin Hasani, Xiao Li, and Daniela Rus. Barriernet: A safety-guaranteed layer for neural networks, 2021. *arXiv preprint arXiv:2111.11277*. doi:10.48550/arXiv.2111.11277.

23 Ruixuan Yan, Agung Julius, Maria Chang, Achille Fokoue, Tengfei Ma, and Rosario Uceda-Sosa. STONE: Signal temporal logic neural network for time series classification. In *2021 International Conference on Data Mining Workshops (ICDMW)*, pages 778–787. IEEE, 2021. doi:10.1109/ICDMW53433.2021.00101.

PDDL to DFA: A Symbolic Transformation for Effective Reasoning

Giuseppe De Giacomo
University of Oxford, UK

Antonio Di Stasio
City St George's, University of London, UK

Gianmarco Parretti
La Sapienza University of Rome, Italy

Abstract

LTL$_f$ reactive synthesis under environment specifications, which concerns the automated generation of strategies enforcing logical specifications, has emerged as a powerful technique for developing autonomous AI systems. It shares many similarities with Fully Observable Nondeterministic (FOND) planning. In particular, nondeterministic domains can be expressed as LTL$_f$ environment specifications. However, this is not needed since nondeterministic domains can be transformed into deterministic finite-state automata (DFA) to be used directly in the synthesis process. In this paper, we present a practical symbolic technique for translating domains expressed in Planning Domain Definition Language (PDDL) into DFAs. The technique allows for the integration of the planning domain, reduced to DFA in a symbolic form, into current symbolic LTL$_f$ synthesis tools. We implemented our technique in a new tool, PDDL2DFA, and applied it to solve FOND planning by using state-of-the-art reactive synthesis techniques in a tool called SYFT4FOND. Our empirical results confirm the effectiveness of our approach.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Theory of computation → Logic and verification; Computing methodologies → Symbolic and algebraic manipulation

Keywords and phrases Fully Observable Nondeterministic Planning, Linear Temporal Logics on finite traces, Reactive Synthesis, DFA

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.7

Supplementary Material *Software (Source Code)*: `https://github.com/GianmarcoDIAG/syft4fond` [33]

1 Introduction

In recent years there has been a growing interest in applying Formal Methods techniques to Artificial Intelligence in order to develop autonomous AI systems that can operate effectively in dynamic and complex environments.

These techniques include reactive synthesis, which concerns the automated generation of *winning strategies* that enforce requirements given in the form of logical specifications [16, 34]. Specifically, we consider reactive synthesis for specifications in Linear Temporal Logic on Finite Traces (LTL$_f$) [26, 27], which maintains the syntax of LTL [35], the formalism typically used to express complex dynamic properties in Formal Methods [8], but it is interpreted on finite traces.

A key aspect shared by all synthesis work in AI is the need for a model of the environment in which the agent acts. In fully observable nondeterministic (FOND) planning [18, 17], such a model is given as a state-based domain that specifies, in each state, how the environment enacts the (possibly nondeterministic) effects of agent actions. In its most common form, FOND planning involves computing a *strong plan* that guarantees reaching one of the goal states independently of the nondeterminism in the domain, thus sharing many similarities

with LTL$_f$ reactive synthesis [27, 15]. Specifically, a FOND domain represented with functions and sets can be expressed as an LTL$_f$ environment specification and transformed into a deterministic finite-state automaton (DFA) that accepts the traces consistent with the domain specification [25, 1, 6, 23]. It follows that FOND planning can be reduced to synthesizing a winning strategy over a DFA game [25, 23], as LTL$_f$ reactive synthesis [27].

In practice, FOND domains are often specified in a compact language like the Planning Domain Definition Language (PDDL) [31], which has been extensively used in planning competitions[1]. However, while how to transform a FOND domain into DFA is well-known in theory, how to effectively transform PDDL into DFA in practice is still open to further investigation.

In this paper, we present an effective technique for transforming PDDL into a symbolic DFA with transitions and final states represented as Boolean functions encoded by using Binary Decision Diagrams (BDDs) [13]. This technique allows for the integration of FOND domains into symbolic LTL$_f$ synthesis tools, which are known for their scalability and efficiency [39, 10, 9, 37]. The construction process involves representing the nondeterminism in the domain through suitable *agent actions* and *environment reactions*. Once this representation is established, the symbolic DFA of the domain can be efficiently constructed by manipulating BDDs. Our technique has the notable property that, while worst-case exponential in the size of the input domain, it is often polynomial due to its concise representation of the nondeterminism in the domain through a compact set of environment reactions.

We implemented our method in a new tool, PDDL2DFA, and applied it to devise a reduction of FOND planning into reactive synthesis (optimal wrt complexity of FOND planning, i.e., EXPTIME-complete [17]) in a tool called SYFT4FOND. We applied this construction to various case studies, including the classic blocks world, blocks world extended, an elevator system, and two navigation environments. Our empirical results show the performance of our approach in these different cases. Specifically, our technique successfully constructs the DFA for a considerable number of instances and solves the synthesis problem for a reasonable number of them, showing the practical feasibility of reducing planning to synthesis.

Our approach takes a step towards integrating planning and synthesis more closely and serves as a promising starting point for future research.

2 Preliminaries

Notations. A *trace* over an alphabet of symbols Σ is a finite or infinite sequence of elements from Σ. The empty trace is denoted λ. Traces are indexed starting at zero, and we write $\pi = \pi_0\pi_1 \cdots$. For a finite trace π, let $\mathsf{lst}(\pi)$ denote the index of the last element of π, i.e., $\mathsf{lst}(\pi) = |\pi| - 1$.

2.1 Linear Temporal Logic on finite traces (LTL$_f$)

LTL$_f$ is a variant of LTL interpreted over *finite traces* [26] instead of *infinite traces* [35]. LTL$_f$ has the same syntax as LTL. Given a set AP of atomic propositions (aka atoms), the LTL$_f$ formulas over AP are generated by the following grammar:

$$\varphi ::= a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

Where $a \in AP$. Here $\bigcirc$ (*Next*) and $\mathcal{U}$ (*Until*) are temporal operators. We use standard Boolean abbreviations such as $\vee$ (or), $\supset$ (implies), *true* and *false*. Moreover, we define the following abbreviations: $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$ (*Weak Next*), $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$ (*Eventually*), and

[1] See `https://www.icaps-conference.org/competitions/`.

$\Box\varphi \equiv \neg\Diamond\neg\varphi$ (*Always*). The size of φ, written $|\varphi|$, is the number of its subformulas. Formulas are interpreted over finite traces π over the alphabet $\Sigma = 2^{AP}$, i.e., the alphabet consisting of the propositional interpretations of the atoms. Thus, for $0 \leq i \leq \mathsf{lst}(\pi)$, $\pi_i \in 2^{AP}$ is the i-th interpretation of π. That an LTL$_f$ formula φ *holds* at instant $i \leq \mathsf{lst}(\pi)$, written $\pi, i \models \varphi$, is defined inductively:

- $\pi, i \models a$ iff $a \in \pi_i$ (for $a \in AP$);
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \mathsf{O}\varphi$ iff $i < \mathsf{lst}(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff $\exists j$ such that $i \leq j \leq \mathsf{lst}(\pi)$ and $\pi, j \models \varphi_2$, and $\forall k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

We say that π *satisfies* φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

2.2 Deterministic Finite Automata

A deterministic finite automaton (DFA) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where: Σ is a finite input alphabet; Q is a finite set of states; $q_0 \in Q$ is the initial state; $\delta : Q \times \Sigma \to Q$ is the transition function; and $F \subseteq Q$ is the set of final states. The size of $\mathcal{A}$ is $|Q|$. Given a finite trace $\alpha = \alpha_0\alpha_1 \ldots \alpha_n$ over Σ, we extend δ to be a function $\delta : Q \times \Sigma^* \to Q$ as follows: $\delta(q, \lambda) = q$, and, if $q_n = \delta(q, \alpha_0 \ldots \alpha_{n-1})$, then $\delta(q, \alpha_0 \ldots \alpha_n) = \delta(q_n, \alpha_n)$. A trace α is *accepted* by $\mathcal{A}$ if $\delta(q_0, \alpha) \in F$. The *language* of $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set of traces that the automaton accepts.

▶ **Theorem 1** ([26]). *Given an* LTL$_f$ *formula* φ*, we can build a* DFA *of at most doubly-exponential size in* $|\varphi|$ *whose language is the set of traces satisfying* φ*.*

2.3 LTL$_f$ Reactive Synthesis

LTL$_f$ *reactive synthesis* [27] concerns finding a strategy to satisfy an LTL$_f$ *goal* specification. Goals are expressed as LTL$_f$ formulas over $AP = \mathcal{X} \cup \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets of variables. Intuitively, $\mathcal{X}$ (resp. $\mathcal{Y}$) is under the environment's (resp. agent's) control. Traces over $\Sigma = 2^{\mathcal{X}\cup\mathcal{Y}}$ will be denoted $\pi = (X_0 \cup Y_0)(X_1 \cup Y_1)\ldots$ where $X_i \subseteq \mathcal{X}$ and $Y_i \subseteq \mathcal{Y}$ for every i. Infinite traces of this form are also called *plays*.

An *agent strategy* is a function $\sigma : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ mapping sequences of environment moves to an agent move. The domain of σ includes the empty sequence λ as we assumed that the agent moves first. A trace π is σ-consistent if $Y_0 = \sigma(\lambda)$ and $Y_{j+1} = \sigma(X_0 \cdots X_j)$ for every $j \geq 0$. Let φ be an LTL$_f$ formula over $\mathcal{X} \cup \mathcal{Y}$. An agent strategy σ is *winning* for (aka *enforces*) φ if, for every play π that is σ-consistent, some finite prefix of π satisfies φ. LTL$_f$ reactive synthesis is the problem of finding an agent strategy σ that enforces φ, if one exists, and is 2EXPTIME-complete in the size of φ [27].

In many AI applications the agent has some knowledge about how the environment works, which it can exploit to enforce the goal [2]. This knowledge can be expressed by an LTL$_f$ formula $\mathcal{E}$ over $\mathcal{X} \cup \mathcal{Y}$, which we call *environment specification*. In the synthesis of winning strategies, we can intuitively see $\mathcal{E}$ as restricting traces of interest to those satisfying $\mathcal{E}$. In this setting, synthesis amounts to computing a strategy that enforces the implication $\mathcal{E} \supset \varphi$, if one exists.

2.4 DFA Games

A DFA game is a DFA $\mathcal{G} = (\Sigma, Q, q_0, \delta, F)$ with input alphabet $\Sigma = 2^{\mathcal{X} \cup \mathcal{Y}}$. The notions of agent strategy and play defined above also apply to DFA games. A play is *winning* if it contains a finite prefix that is accepted by the DFA. An agent strategy is *winning* if, for every play π that is σ-consistent, π is winning. That is, the agent wins the game if it can force the play to visit the set of final states at least once. The *winning region* is the set of states $q \in Q$ for which the agent has a winning strategy in the game $\mathcal{G}'$, where $\mathcal{G}' = (2^{\mathcal{X} \cup \mathcal{Y}}, Q, q, \delta, F)$, i.e., the same game as $\mathcal{G}$, but with initial state q. *Solving* a DFA game is the problem of computing the agent winning region and a winning strategy, if one exists. DFA games can be solved in polynomial time by a backward-induction algorithm that performs a fixpoint computation over the state space of the game [7]. Synthesis of an LTL_f formula φ can be reduced in doubly-exponential time to solve the DFA game $\mathcal{G}_\varphi$ corresponding to φ [27].

Solving LTL_f synthesis reduces to solving a reachability game over the DFA corresponding to the LTL_f specification. The procedure for solving LTL_f synthesis is detailed in Algorithm 1.

Algorithm 1 LTL_f Synthesis.

Input: LTL_f formula φ
Output: strategy σ_{ag} realizing φ;

1: Compute the corresponding NFA $\mathcal{A}_\varphi$;
2: Determinize $\mathcal{A}_\varphi$ into a DFA $\mathcal{B}_\varphi$;
3: Solve the reachability game over $\mathcal{B}_\varphi$.

2.5 Symbolic Synthesis

We consider the DFA representation described above as an explicit-state representation. Instead, we are able to represent a DFA more compactly in a symbolic way by using a logarithmic number of propositions to encode the state space [40]. Formally, the *symbolic representation* of a DFA $\mathcal{A} = (2^{\mathcal{X} \cup \mathcal{Y}}, Q, q_0, \delta, F)$ is a tuple $\mathcal{A}^s = (\mathcal{X}, \mathcal{Y}, \mathcal{Z}, Z_0, \eta, f)$, where: $\mathcal{Z}$ is a set of *state variables* such that $|\mathcal{Z}| = \lceil \log |Q| \rceil$, and every state $q \in Q$ corresponds to an interpretation $Z \in 2^{\mathcal{Z}}$; $Z_0 \in 2^{\mathcal{Z}}$ is the interpretation corresponding to the initial state q_0; $\eta \colon 2^{\mathcal{Z}} \times 2^{\mathcal{X}} \times 2^{\mathcal{Y}} \to 2^{\mathcal{Z}}$ is a Boolean function such that $\eta(Z, X, Y) = Z'$ if and only if Z is the interpretation of a state q and Z' is the interpretation of the state $\delta(q, X \cup Y)$; and f is a Boolean function over $\mathcal{Z}$ such that $f(Z) = 1$ if and only if Z is the interpretation corresponding to a state $q \in F$. Note that the transition function η can be represented by an indexed family consisting of a Boolean formula η_z for each state variable $z \in \mathcal{Z}$, which when evaluated over an assignment to $\mathcal{Z} \cup \mathcal{X} \cup \mathcal{Y}$ returns the next assignment to z.

A symbolic DFA game can be solved by performing a least fixpoint computation over two Boolean formulas w over $\mathcal{Z}$ and t over $\mathcal{Z} \cup \mathcal{Y}$ which represent the winning region and winning states with agent moves such that, regardless of how the environment behaves, the agent reaches the final states, respectively [40]. Specifically, w and t are initialized as $w_0(\mathcal{Z}) = f(\mathcal{Z})$ and $t_0(\mathcal{Z}, \mathcal{Y}) = f(\mathcal{Z})$, since every final state is a winning state. Note that t_0 is independent of the propositions from $\mathcal{Y}$, since once the play reaches the final states, the agent can do whatever it wants. Then, t_{i+1} and w_{i+1} are constructed as follows:

$$t_{i+1}(Z, Y) = t_i(Z, Y) \vee (\neg w_i(Z) \wedge \forall X. w_i(\eta(X, Y, Z)))$$
$$w_{i+1}(Z) = \exists Y. t_{i+1}(Z, Y)$$

The computation reaches a fixpoint when $w_{i+1} \equiv w_i$. When the fixpoint is reached, no more states will be added, and all winning states have been collected. By evaluating Z_0 on w_{i+1} we can determine if there exists a winning strategy. If that is the case, t_{i+1} can be used to compute a uniform positional winning strategy through the mechanism of Boolean synthesis [28].

2.6 Fully Observable Non-Deterministic (FOND) Planning

Following [23], we define a FOND domain as a tuple $\mathcal{D} = (2^{\mathcal{F}}, s_0, Act, React, \alpha, \beta, \delta)$, where: $\mathcal{F}$ is a finite set of fluents, $|\mathcal{F}|$ is the size of $\mathcal{D}$, and $2^{\mathcal{F}}$ is the state space; Act and $React$ are finite sets of agent actions and environment reactions, respectively; $\alpha : 2^{\mathcal{F}} \to 2^{Act}$ is a function denoting agent action preconditions; $\beta : 2^{\mathcal{F}} \times Act \to 2^{React}$ is a function denoting environment reaction preconditions; and $\delta : 2^{\mathcal{F}} \times Act \times React \to 2^{\mathcal{F}}$ is the transition function such that $\delta(s, a, r)$ is defined if and only if $a \in \alpha(s)$ and $r \in \beta(s, a)$. We assume that planning domains satisfy the properties of:

- *Existence of agent action*: $\forall s \in 2^{\mathcal{F}}.\exists a \in \alpha(s)$;
- *Existence of environment reaction*: $\forall s \in 2^{\mathcal{F}}.\forall a \in \alpha(s).\exists r \in \beta(s, a)$;
- *Uniqueness of environment reaction*:
  $\forall s \in 2^{\mathcal{F}}.\forall a \in \alpha(s).\forall r_1, r_2 \in \beta(s, a).\delta(s, a, r_1) = \delta(s, a, r_2) \supset r_1 = r_2$.

With these properties, inspired by [22], we can capture classical FOND domains [18, 29] by explicitly introducing environment reactions corresponding to nondeterministic effects of agent actions.

A *state trace* of $\mathcal{D}$ is a finite sequence $\tau = s_0 \cdots s_n$ of states such that s_0 is the initial state of $\mathcal{D}$ and, for every $i < n$, there exists an agent action $a_i \in \alpha(s_i)$ and an environment reaction $r_i \in \beta(s_i, a_i)$ such that $s_{i+1} = \delta(s_i, a_i, r_i)$. A *plan* is a partial function $\kappa : 2^{\mathcal{F}} \to Act$ such that, if $\kappa(s)$ is defined, then $\kappa(s) \in \alpha(s)$. A plan terminates its execution in states where, being a partial function, it specifies no action. A state trace $\tau = s_0 \cdots s_n$ is κ-consistent if: (i) s_0 is the initial state of $\mathcal{D}$; (ii) for every $i < n$, $s_{i+1} = \delta(s_i, a_i, r_i)$ for $a_i = \kappa(s_i)$ and some $r_i \in \beta(s_i, a_i)$; and (iii) $\kappa(s_n)$ is undefined.

FOND (strong) planning concerns finding a plan to satisfy a goal regardless of the nondeterminism in the domain, called *strong plan*. A goal G is a conjunction of fluents and negations of fluents. Given a goal G and a domain $\mathcal{D}$, a plan κ is strong for G in $\mathcal{D}$ if, for every state trace $\tau = s_0, \cdots, s_n$ that is κ-consistent, $s_n \models G$. Formally, FOND planning is the problem of finding a strong plan for G in $\mathcal{D}$, if one exists. FOND planning is EXPTIME-complete in the size of $\mathcal{D}$ [17].

In this paper, we always assume that we have FOND domains expressed in PDDL [31], i.e., the fluents defining the states of the domain are predicates over objects. We write `predicate/k` to specify that k objects participate in the `predicate` relation. Predicates, actions, and action preconditions are specified in first-order syntax in a `domain.pddl` file, whereas the initial state, goal, and objects, are usually specified in a separate `problem.pddl` file.

3 PDDL to Symbolic DFA

In this paper, we present a technique for transforming PDDL into symbolic DFA. A naive approach is to translate PDDL into LTL_f [2] and build its corresponding symbolic DFA. However, this approach is limited by the doubly-exponential blow-up resulting from transforming LTL_f formulas in DFAs [26]. Instead, our technique ensures only a single-exponential blowup in the number of fluents.

Our transformation is based on representing the nondeterminism in the domain with agent actions and environment reactions, as described in Section 2. Once such a representation is established, the symbolic DFA of the domain can be easily built by constructing suitable Boolean formulas.

The core idea to represent the nondeterminism in the domain with agent actions and environment reactions is as follows. For every state of the domain $s \in 2^{\mathcal{F}}$ and possible nondeterministic effect $s_1, \cdots, s_n$ of an agent action $a \in \alpha(s)$ (specified in its `oneof` clause), we introduce an environment reaction $r_1, \cdots, r_n$ such that, for every $i < n$, we have $\delta(s, a, r_i) = s_i$ and $r_i \in \beta(s, a)$. That is, r_i represents the i-th nondeterministic effect of applying action a in s. Applying this construction to every state $s \in 2^{\mathcal{F}}$ and agent action $a \in Act$ generates a planning domain $\mathcal{D} = (2^{\mathcal{F}}, s_0, Act, React, \alpha, \beta, \delta)$ as detailed in Algorithm 2

Algorithm 2 PDDL2ACTSANDREACTS(`domain.pddl`, `problem.pddl`).

Require: PDDL description of planning domain as `domain.pddl` and `problem.pddl`
Ensure: A nondeterministic planning domain $\mathcal{D} = (2^{\mathcal{F}}, s_0, Act, React, \alpha, \beta, \delta)$
1: Let $\mathcal{F}$ be the set of fluents in `domain.pddl` and `problem.pddl`
2: Let s_0 be the initial state from `problem.pddl`
3: Let Act and α be actions and preconditions from `domain.pddl` and `problem.pddl`
4: **for** $s \in 2^{\mathcal{F}}$:
5: **for** $a \in Act$:
6: **if** $a \in \alpha(s)$:
7: Let $\{s_1, \cdots, s_n\}$ be the successor states of s specified in a's `oneof` clause
8: UPDATE: $React = React \cup \{r_1, \cdots, r_n\}$
9: DEFINE: $\delta(s, a, r_i) = s_i$ and $r_i \in \beta(s, a)$ for every i
10: **Return** $\mathcal{D} = (2^{\mathcal{F}}, s_0, Act, React, \alpha, \beta, \delta)$

▶ **Example 2.** Consider a robotic agent that operates in a blocksworld environment where it can pick up/drop blocks from/in top of other blocks as well as pick up/drop blocks from/on the table. The domain defines the predicates `emptyhand/0`, `holding/1`, `on-table/1`, `on/2`, `clear/1` to specify that the agent holds no block, the agent holds a block, a block is on the table, a block is on top of another block, and a block can be picked, respectively.

Assume that there exist two blocks, `yellow` and `green`, and that in the initial state `green` is on the table, `yellow` is on top of `green`, and the agent holds no block. Consider the agent actions `pick-up` and `put-on-block` defined as follows.

```
(:action pick-up
    :parameters    (?b1 ?b2 - block)
    :precondition  (and
                       (not (= ?b1 ?b2))
                       (emptyhand)
                       (clear ?b1)
                       (on ?b1 ?b2))
    :effect        (oneof
                       (and
                           (holding ?b1)
                           (clear ?b2)
                           (not (emptyhand))
                           (not (clear ?b1))
                           (not (on ?b1 ?b2)))
```

```
                        (and
                            (clear ?b2)
                            (on-table ?b1)
                            (not (on ?b1 ?b2)))))

(:action put-on-block
    :parameters     (?b1 ?b2 - block)
    :precondition   (and
                        (holding ?b1)
                        (clear ?b2))
    :effect         (oneof
                        (and
                            (on ?b1 ?b2)
                            (emptyhand)
                            (clear ?b1)
                            (not (holding ?b1))
                            (not (clear ?b2)))
                        (and
                            (on-table ?b1)
                            (emptyhand)
                            (clear ?b1)
                            (not (holding ?b1)))))
```

Intuitively, action `pick-up` specifies that whenever the agent tries to pick up a block, either it succeeds, or it does not and the block falls on the table. Similarly for action `put-on-block`. We can capture the nondeterministic effects of actions `pick-up` and `put-on-block` with two reactions `succeed` and `fail`. A fragment of the planning domain resulting from applying Algorithm 2 to the PDDL description above is shown in Figure 1.

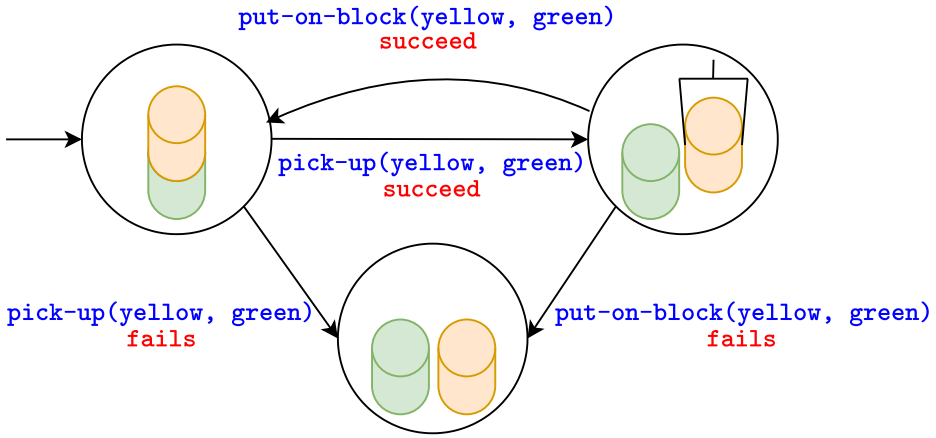


Figure 1 Fragment of the domain resulting from applying Algorithm 2 to the PDDL description in Example 2.

▶ Remark 3. Algorithm 2 returns a domain $\mathcal{D} = (2^{\mathcal{F}}, s_0, Act, React, \alpha, \beta, \delta)$ with fluents, initial state, agent actions, and action preconditions as in its PDDL description. The number of environment reactions is at most single-exponential in the number of fluents $|\mathcal{F}|$. To see this, observe that there may exist an action $a \in Act$ that, from some state $s \in 2^{\mathcal{F}}$ of

the domain, nondeterministically leads to every other state $s' \in 2^{\mathcal{F}}$, thus generating an environment reaction for each such successor state. However, in practice the number of reactions is often several orders of magnitude less than $2^{\mathcal{F}}$, since the possible nondeterministic effects in `oneof` clauses of agent actions are usually very few.

Once the nondeterminism in the domain is represented through deterministic action-reaction pairs (a, r), we can construct the symbolic DFA of the domain. To do this, we characterize each pair (a, r) in terms of its add-list $\texttt{add}(a, r)$ and delete-list $\texttt{del}(a, r)$, which are the sets of fluents added and deleted by (a, r), respectively. Formally, a fluent f is in $\texttt{add}(a, r)$ (resp. $\texttt{del}(a, r)$) iff for every $s \in 2^{\mathcal{F}}$, if $a \in \alpha(s)$ and $f \notin s$ (resp. $f \in s$), then $f \in \delta(s, a, r)$ (resp. $f \notin \delta(s, a, r)$). The add- and delete-lists of action-reaction pairs can be extracted immediately from the PDDL description of the domain. Specifically, fluents appearing without (resp. with) `not` in the `oneof` clause of an action a are in the add-list (resp. delete-list) of the corresponding action-reaction pair (a, r).

We give in Algorithm 3 a technique to transform a FOND domain with a goal into a symbolic DFA. Algorithm 3 constructs a symbolic DFA with a state variable for each fluent and two error state variables, `AgErr` and `EnvErr`, denoting that the agent and the environment violated the domain specification, respectively, and whose initial state is that of the domain (Line 1). The alphabet of the symbolic DFA is partitioned into actions and reactions, which are under the control of the agent and the environment, respectively (Line 2). For every state variable corresponding to a fluent, Algorithm 3 constructs its transition function as specified in Line 3. Intuitively, the transition function η_f of fluent f specifies that in the next time step f holds if and only if either:

1. f was true in the previous time step and was not deleted by an action-reaction pair (a, r) such that $f \in \texttt{del}(a, r)$; *or*
2. f was added by some action reaction pair (a, r) such that $f \in \texttt{add}(a, r)$.

Algorithm 3 constructs the transition functions of `AgErr` and `EnvErr` in Lines 4 and 5. Intuitively, the transition function $\eta_{\texttt{AgErr}}$ of `AgErr` specifies that in the next time step the agent reaches the error state if and only if either:

1. The agent was in its error state in the previous time step, written `AgErr`; *or*
2. The agent violated the mutual exclusion axiom for its actions, which states that, at each time step, the agent must execute one and only one action, written $\neg\texttt{AgMutex}(\mathcal{Y})$;
3. The agent violated an action precondition, written $\neg\texttt{AgPre}(\mathcal{Z}, \mathcal{Y})$.

The transition function of `EnvErr` is constructed similarly.

Taking the agent's point of view, Algorithm 3 constructs the final states of the DFA so that a trace is accepted if the agent does not reach its error state and either the environment reaches its error state or the goal is reached (Line 6).

The size of the symbolic DFA constructed by Algorithm 3, i.e., the number of its state variables, is polynomial in the size of the domain. Each line can be executed in polynomial time in the size of the domain. As a result, Algorithm 3 runs in polynomial time in the size of the input domain.

Together, Algorithms 2 and 3 form our technique for transforming PDDL into symbolic DFA. While worst-case exponential in the size of the input domain, our transformation is often polynomial due to its compact representation of nondeterministic effects of agent actions with suitable environment reactions.

4 Reduction of FOND Planning to Synthesis

As an application to demonstrate the effectiveness of our PDDL to DFA transformation technique, we show how to use it to solve FOND planning problems.

Algorithm 3 DomainToDfa($\mathcal{D}, G$).

Require: A FOND domain $\mathcal{D} = (2^{\mathcal{F}}, s_0, Act, React, \alpha, \beta, \delta)$ with goal G
Ensure: A symbolic DFA $\mathcal{A}^s = (\mathcal{Z}, \mathcal{X}, \mathcal{Y}, Z_0, \eta, f)$

1: Define $\mathcal{Z} = \mathcal{F} \cup \{\texttt{AgErr}, \texttt{EnvErr}\}$
2: Define initial state $Z_0 = s_0$
3: Define $\mathcal{Y} = Act$
4: Define $\mathcal{X} = React$
5: **for each** $f \in \mathcal{F}$:

$$\eta_f(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) = \left(f \wedge \neg \bigvee_{(a,r) | f \in \texttt{del}(a,r)} (a \wedge r) \right) \vee \bigvee_{(a,r) | f \in \texttt{add}(a,r)} (a \wedge r)$$

6: Define agent error transition:

$$\eta_{\texttt{AgErr}}(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) = \texttt{AgErr} \vee \neg \texttt{AgMutex}(\mathcal{Y}) \vee \neg \texttt{AgPre}(\mathcal{Z}, \mathcal{Y})$$

where:

$$\texttt{AgMutex}(\mathcal{Y}) = \left(\bigvee_{a \in Act} a \right) \wedge \left(\bigwedge_{a, a' \in Act, a \neq a'} a \supset \neg a' \right)$$

$$\texttt{AgPre}(\mathcal{Z}, \mathcal{Y}) = \bigwedge_{a \in Act} \left(a \supset \bigvee_{s \in 2^{\mathcal{F}}, a \in \alpha(s)} s \right)$$

7: Define environment error transition:

$$\eta_{\texttt{EnvErr}}(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) = \texttt{EnvErr} \vee \neg \texttt{EnvMutex}(\mathcal{X}) \vee \neg \texttt{EnvPre}(\mathcal{Z}, \mathcal{Y}, \mathcal{X})$$

where:

$$\texttt{EnvMutex}(\mathcal{X}) = \left(\bigvee_{r \in React} r \right) \wedge \left(\bigwedge_{r, r' \in React, r \neq r'} r \supset \neg r' \right)$$

$$\texttt{EnvPre}(\mathcal{Z}, \mathcal{Y}, \mathcal{X}) = \bigwedge_{r \in React} \left(r \supset \bigvee_{(s,a) \in (2^{\mathcal{F}} \times Act), r \in \beta(s,a)} (s \wedge a) \right)$$

8: Define accepting condition:

$$f(\mathcal{Z}) = \neg \texttt{AgErr} \wedge (\texttt{EnvErr} \vee G)$$

9: **Return** $\mathcal{A}^s = (\mathcal{X}, \mathcal{Y}, \mathcal{Z}, Z_0, \eta, f)$

Let $\mathcal{D}$ and G be a FOND domain and goal in PDDL. Construct its symbolic DFA $\mathcal{A}^s$ by using Algorithms 2 and 3. Solve the symbolic DFA game over $\mathcal{A}^s$, assigning actions and reactions to agent and environment, respectively.

▶ **Theorem 4.** *Let $\mathcal{D}$ and G be a* FOND *domain and goal in* PDDL *and $\mathcal{A}^s$ their* DFA. *There is a strong plan for G in $\mathcal{D}$ iff there is a winning strategy in $\mathcal{A}^s$.*

This synthesis technique is exponential in the size of the PDDL domain and therefore optimal wrt the computational complexity of FOND planning. However, we also observe that synthesis on DFA games is based on basic backward search algorithms. While we do not introduce sophisticated optimization techniques, we show that we can include them in our synthesis algorithm.

Specifically, we show how to include a simple form of *invariants*, i.e., properties of states of the domain that must remain unchanged during the execution of every sequence of actions [12]. Invariants can be used to prune the search space by eliminating actions or states that violate these unchanging properties.

We consider *mutual exclusion* invariants specifying that, at every state, no more than one fluent or negation of fluent l in a set I can be true. Such invariants can be captured as a Boolean formula $i(\mathcal{Z})$ over the state variables of the symbolic DFA constructed in Algorithm 3 as:

$$i(\mathcal{Z}) = \bigwedge_{l \in I} (l \supset \bigwedge_{l' \in I. l \neq l'} \neg l')$$

To include invariants in backward search, we rewrite the fixpoint computation in Section 2. Specifically, w and t are now initialized as $w_0(\mathcal{Z}) = f(\mathcal{Z}) \wedge i(\mathcal{Z})$ and $t_0(\mathcal{Z}, \mathcal{Y}) = f(\mathcal{Z}) \wedge i(\mathcal{Z})$, since every goal state must satisfy the invariant. Then, t_{i+1} and w_{i+1} are constructed as follows:

$$t_{i+1}(\mathcal{Z}, \mathcal{Y}) = t_i(\mathcal{Z}, \mathcal{Y}) \vee (\neg w_i(\mathcal{Z}) \wedge \forall \mathcal{X}. w_i(\eta(\mathcal{X}, \mathcal{Y}, \mathcal{Z})))$$

$$w_{i+1}(\mathcal{Z}) = (\exists \mathcal{Y}. t_{i+1}(\mathcal{Z}, \mathcal{Y})) \wedge i(\mathcal{Z})$$

That is, only states satisfying the invariant are added to the winning region.

▶ Remark 5. An approach alternative to ours is to reduce FOND planning to LTL_f synthesis of the formula $\mathcal{E} \supset \varphi$, where $\mathcal{E}$ and φ describe the domain and the agent goal, respectively [2]. The LTL_f formula is transformed in DFA in doubly-exponential time (Theorem 1) and a strong plan is obtained by solving the corresponding DFA game. However, this approach is limited by the doubly-exponential blowup resulting from transforming the LTL_f formula in DFA.

5 Evaluation

We implemented Algorithms 2 and 3 in a tool called PDDL2DFA. For parsing, grounding, and computing invariants for the input PDDL domain, we based on the tool PRP [32]. We code Boolean functions representing transitions and final states of symbolic DFAs by Binary Decision Diagrams (BDDs) [13] with the BDD library CUDD 3.0.0 [36]. The size of a BDD is the number of its nodes. PDDL2DFA also implements the transformation from PDDL to LTL_f (see Remark 5), in which case the DFAs of LTL_f formulas are constructed with LYDIA [21], which is among the best performing tools publicly available for LTL_f-to-DFA conversion.

We applied the transformation in PDDL2DFA to implement the reduction of FOND planning to synthesis (ref. Section 4) in a tool called SYFT4FOND[2]. For solving symbolic DFA games, we use the symbolic synthesis framework in [40] at the base of state-of-the-art LTL_f synthesis tools [11, 37]. We compute winning strategies for DFA games through Boolean synthesis [28].

[2] PDDL2DFA and SYFT4FOND at `https://github.com/GianmarcoDIAG/syft4fond`

Setup. Experiments were run on a laptop running 64-bit Ubuntu 22.04, 3.6 GHz CPU, and 12 GB of memory. Timeout was 1000 secs.

5.1 Benchmark

We performed experiments on a suite of 170 classical FOND planning benchmarks divided in five classes: blocks world (50 instance), extended blocks world (50 instances), triangle-tire world (40 instances), rectangle-tire world (15 instances), and elevators (15 instances). In blocks world instances, the agent manipulates blocks with actions that can nondeterministically succeed or fail. In extended blocks world instances, the agent can also move towers of two blocks, with similar nondeterministic success or failure in actions. In triangle-tire and rectangle-tire instances, the agent navigates a grid environment, dealing with nondeterministic success or failure when moving. Elevator instances involve managing elevators to collect coins across multiple floors. For each class, the instances grow by increasing the problem size, which depends on their parameters.

5.2 Empirical Results

We performed experiments to evaluate the efficiency of our technique for translating PDDL into symbolic DFA and the practical feasibility of reducing FOND planning to synthesis.

We evaluated the performance of PDDL2DFA in transforming PDDL to LTL_f and LTL_f to symbolic DFA. When employing this transformation, PDDL2DFA was unable to construct the symbolic DFA of the PDDL domain in the considered benchmarks, except in very few cases. The bottleneck was transforming LTL_f into DFA, which requires doubly-exponential time in the size of the LTL_f formula [26].

Table 1 Coverage achieved by PDDL2DFA (constructed DFAs/instances in benchmark), and domain size ($|\mathcal{F}|$), number of actions ($|Act|$), size of largest fluent BDD ($\text{Max}_{\{\mathcal{F}\}}$ BDD), size of smallest fluent BDD ($\text{Min}_{\{\mathcal{F}\}}$ BDD), size of agent error BDD (`AgErr` BDD), and size of environment error BDD (`EnvErr` BDD) in the largest solved instance.

Bench.	Coverage	$\lvert\mathcal{F}\rvert$	$\lvert Act\rvert$	$\text{Max}_{\{\mathcal{F}\}}$ BDD	$\text{Min}_{\{\mathcal{F}\}}$ BDD	`AgErr` BDD	`EnvErr` BDD
Blocks	31/50	1055	1953	514	14	6549	109
BlocksExt	19/50	379	12654	5085	277	156509	562
Triangle	30/40	2881	4709	1305	17	40247	740
Rect.	10/15	53	52249	5432	2077	66261	1783
Elev.	15/15	66	105	42	10	511	41
Total	**105**/170	–	–	–	–	–	–

We evaluated the performance of PDDL2DFA in transforming PDDL to symbolic DFA with Algorithms 2 and 3. Table 1 shows that PDDL2DFA constructed the DFA for a considerable number of benchmarks. Notably, PDDL2DFA was able to construct the DFA of triangle-tire domains with side 61. The bottleneck of the DFA construction was computing the agent error BDD. Indeed, Table 1 shows that the size of the agent error BDD in the hardest solved instances is orders of magnitude larger than that of the other BDDs, including that of the largest fluent BDD. The reason is that constructing the BDD of the agent error requires iterating over all actions of the domain. Instead, constructing the BDDs of the fluents requires only considering actions containing that fluent in their add- and delete-lists. Constructing the environment error BDD requires iterating over all environment reactions, but these are often several orders of magnitude less than fluents and agent actions (see Remark 3). As

a result, the number of actions is the parameter that affects most the performance of the DFA construction. The size of the PDDL domain, i.e., its fluents, has less impact than the number of agent actions on the DFA construction performance. Overall, this is a good result and shows the effectiveness of our construction.

Table 2 Coverage achieved by SYFT4FOND (solved instances/instances in benchmark), and domain size ($|\mathcal{F}|$), number of actions ($|Act|$), size of largest fluent BDD ($\text{Max}_{\{\mathcal{F}\}}$ BDD), size of smallest fluent BDD ($\text{Min}_{\{\mathcal{F}\}}$ BDD), size of agent error BDD (`AgErr` BDD), and size of environment error BDD (`EnvErr` BDD) in the largest solved instance.

Bench.	Coverage	$\lvert\mathcal{F}\rvert$	$\lvert Act\rvert$	$\text{Max}_{\{\mathcal{F}\}}$ BDD	$\text{Min}_{\{\mathcal{F}\}}$ BDD	`AgErr` BDD	`EnvErr` BDD
Blocks	6/50	55	78	46	10	403	29
BlocksExt	3/50	19	81	33	8	327	32
Triangle	9/40	298	467	169	12	2119	111
Rect.	8/15	25	15481	2272	1057	74757	884
Elev.	7/15	44	68	28	10	281	28
Total	**33**/170	-	-	-	-	-	-

We evaluated the performance of SYFT4FOND in reducing FOND planning to synthesis. Table 2 shows that SYFT4FOND is able to solve a reasonable number of instances. Indeed, SYFT4FOND was able to solve triangle-tire planning instances with side 19, though based on plain backward search. The bottleneck of the synthesis was constructing the BDDs of the agent winning moves and region during the fixpoint computation, which are mostly affected by the size of the agent error BDD. In general, we consider this result adequate to show the practical feasibility of reducing FOND planning to synthesis.

6 Conclusion

In this paper, we have presented an effective method for translating PDDL into symbolic DFA. We implemented our method in a new tool, PDDL2DFA, and applied it to solving planning problems through reduction to synthesis in the tool SYFT4FOND. Testing these tools on various case studies demonstrated the practicality and performance of our approach. Indeed, we demonstrated that our method successfully constructs DFAs for considerably large domains and solves a practical number of planning instances, performing significantly better than straightforward approaches based on translating PDDL in LTL_f. Our work makes a step towards integrating planning and synthesis more closely. Future research can build on this foundation, aiming to integrate FOND domains into advanced synthesis techniques developed for temporally extended goals [27, 19], structured environment specifications [20, 9, 30, 3], multiple goal specifications [14, 38], and for handling goal unrealizability [4, 5, 24, 23].

References

1 Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning and synthesis under assumptions. *CoRR*, abs/1807.06777, 2018. `arXiv:1807.06777`.

2 Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning under LTL environment specifications. In *ICAPS*, pages 31–39, 2019. URL: `https://ojs.aaai.org/index.php/ICAPS/article/view/3457`.

3 Benjamin Aminof, Giuseppe De Giacomo, Gianmarco Parretti, and Sasha Rubin. Effective approach to ltlf best-effort synthesis in multi-tier environments. In *IJCAI*, 2024.

4 Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Best-effort synthesis: Doing your best is not harder than giving up. In *IJCAI*, pages 1766–1772, 2021. doi:10.24963/IJCAI.2021/243.

5 Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Reactive synthesis of dominant strategies. In *AAAI*, pages 6228–6235. AAAI Press, 2023. doi:10.1609/AAAI.V37I5.25767.

6 Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning under LTL environment specifications. In *ICAPS*, pages 31–39. AAAI Press, 2019. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3457.

7 Krzysztof R. Apt and Erich Grädel, editors. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.

8 Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008.

9 Suguman Bansal, Giuseppe De Giacomo, Antonio Di Stasio, Yong Li, Moshe Y. Vardi, and Shufang Zhu. Compositional safety LTL synthesis. In *VSTTE*, volume 13800 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2022. doi:10.1007/978-3-031-25803-9_1.

10 Suguman Bansal, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, pages 9766–9774, 2020. doi:10.1609/AAAI.V34I06.6528.

11 Suguman Bansal, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, pages 9766–9774, 2020. doi:10.1609/AAAI.V34I06.6528.

12 Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997. doi:10.1016/S0004-3702(96)00047-1.

13 Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992. doi:10.1145/136035.136043.

14 Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Finite LTL synthesis with environment assumptions and quality measures. In *KR*, pages 454–463. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/18072.

15 Alberto Camacho, Meghyn Bienvenu, and Sheila A McIlraith. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, pages 58–67, 2019. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3460.

16 Alonzo Church. Logic, arithmetic and automata. In *Proc. International Congress of Mathematicians*, 1963.

17 Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147:35–84, 2003. doi:10.1016/S0004-3702(02)00374-0.

18 Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong Planning in Non-Deterministic Domains Via Model Checking. In *AIPS*, pages 36–43. AAAI, 1998. URL: http://www.aaai.org/Library/AIPS/1998/aips98-005.php.

19 Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. Pure-past linear temporal and dynamic logic on finite traces. In *IJCAI*, pages 4959–4965, 2020. doi:10.24963/IJCAI.2020/690.

20 Giuseppe De Giacomo, Antonio Di Stasio, Moshe Y Vardi, and Shufang Zhu. Two-stage technique for LTL_f synthesis under LTL assumptions. In *KR*, pages 304–314, 2020. doi:10.24963/KR.2020/31.

21 Giuseppe De Giacomo and Marco Favorito. Compositional approach to translate $\mathrm{LTL}_f/\mathrm{LDL}_f$ into deterministic finite automata. In *ICAPS*, pages 122–130, 2021. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/15954.

22 Giuseppe De Giacomo and Yves Lespérance. The nondeterministic situation calculus. In *KR*, pages 216–226, 2021. doi:10.24963/KR.2021/21.

23 Giuseppe De Giacomo, Gianmarco Parretti, and Shufang Zhu. LTL_f best-effort synthesis in nondeterministic planning domains. In *ECAI*, pages 533–540, 2023. doi:10.3233/FAIA230313.

24 Giuseppe De Giacomo, Gianmarco Parretti, and Shufang Zhu. Symbolic LTL_f best-effort synthesis. In *EUMAS*, pages 228–243, 2023.

25 Giuseppe De Giacomo and Sasha Rubin. Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals. In *IJCAI*, pages 4729–4735, 2018. doi:10.24963/IJCAI.2018/657.

26 Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

27 Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564, 2015. URL: http://ijcai.org/Abstract/15/223.

28 Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. BDD-based Boolean functional synthesis. In *CAV*, pages 402–421, 2016. doi:10.1007/978-3-319-41540-6_22.

29 Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool, 2013.

30 Giuseppe De Giacomo, Antonio Di Stasio, Lucas M. Tabajara, Moshe Y. Vardi, and Shufang Zhu. Finite-trace and generalized-reactivity specifications in temporal synthesis. *Formal Methods Syst. Des.*, 61(2):139–163, 2022. doi:10.1007/S10703-023-00413-2.

31 Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool, 2019.

32 Christian Muise, Sheila A McIlraith, and J Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*, 2012.

33 Gianmarco Parretti. Syft4Fond. Software (visited on 2025-09-18). URL: https://github.com/GianmarcoDIAG/syft4fond, doi:10.4230/artifacts.24786.

34 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

35 Amir Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.

36 Fabio Somenzi. CUDD: CU Decision Diagram Package 3.0.0. Universiy of Colorado at Boulder, 2016.

37 Shufang Zhu and Marco Favorito. Lydiasyft: A compositional symbolic synthesis framework for $\mathrm{ltl_f}$ specifications. In *TACAS 2025*, pages 295–302, 2025. doi:10.1007/978-3-031-90643-5_15.

38 Shufang Zhu and Giuseppe De Giacomo. Act for your duties but maintain your rights. In *KR*, 2022.

39 Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTL_f synthesis. In *IJCAI*, pages 1362–1369, 2017. doi:10.24963/IJCAI.2017/189.

40 Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTL_f synthesis. In *IJCAI*, pages 1362–1369, 2017. doi:10.24963/IJCAI.2017/189.

Heuristics for Covering the Timeline in Temporal Graphs

Riccardo Dondi
Università degli Studi di Bergamo, Italy

Rares-Ioan Mateiu
Department of Computer Science, University of Bucharest, Romania

Alexandru Popa
Department of Computer Science, University of Bucharest, Romania

Abstract

We consider a variant of the Vertex Cover problem on temporal graphs, called Minimum Timeline Cover (k-MinTimelineCover). Temporal graphs are used to model complex systems, describing how edges (relations) change in a discrete time domain. The k-MinTimelineCover problem has been introduced in complex data summarization and synthesis jobs. Given a temporal graph G, k-MinTimelineCover asks to define k activity intervals for each vertex, such that each temporal edge is covered by at least one active interval. The objective function is the minimization of the sum of interval lengths. k-MinTimelineCover is NP-hard and even hard to approximate within any factor for $k > 1$. While the literature has mainly focused on the cases $k = 1$, in this contribution we consider the case $k > 1$. We first present an ILP formulation that is able to solve the problem on moderate size instances. Then we develop an efficient heuristic, based on local search which is built on top of the solution of an existing literature method.

Finally, we present an experimental evaluation of our algorithms on synthetic data sets, that shows in particular that our heuristic has a consistent improvement on the state-of-the art method.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Design and analysis of algorithms; Theory of computation → Mathematical optimization; Theory of computation → Discrete optimization

Keywords and phrases Temporal Networks, Activity Timeline, Vertex Cover, Heuristic, Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.8

1 Introduction

Complex systems are usually modeled through graphs or networks in order to understand their properties and behaviors. The availability of temporal information of relations between vertices has led to the introduction of temporal graphs [7, 13, 9], where edges, called temporal edges, represent interactions between two entities at a given time. Note that different variations of the temporal graph model have been introduced; here we consider a model where the time domain, over which the temporal edges are defined, is discrete [7, 13, 9].

Temporal graphs have been considered in several domains, e.g., social network analysis [17], computational biology [8] and epidemiology [1]. We consider an approach defined for the summarizations of temporal interactions in social networks [12, 16, 15, 14]. In this context, a user activity in a social network (called activity timeline) is represented with one or more time intervals; the goal is to represent users interaction, so that if an interaction is observed at time t, then at least one of the user activity timeline include t [14, 15, 6, 3, 2, 4, 5]. The objective function, following a parsimonious approach, is the minimization of the overall length (called *span*) of timeline activities or the minimization of the maximum length of the timeline activity. Here, we focus on the first objective function.

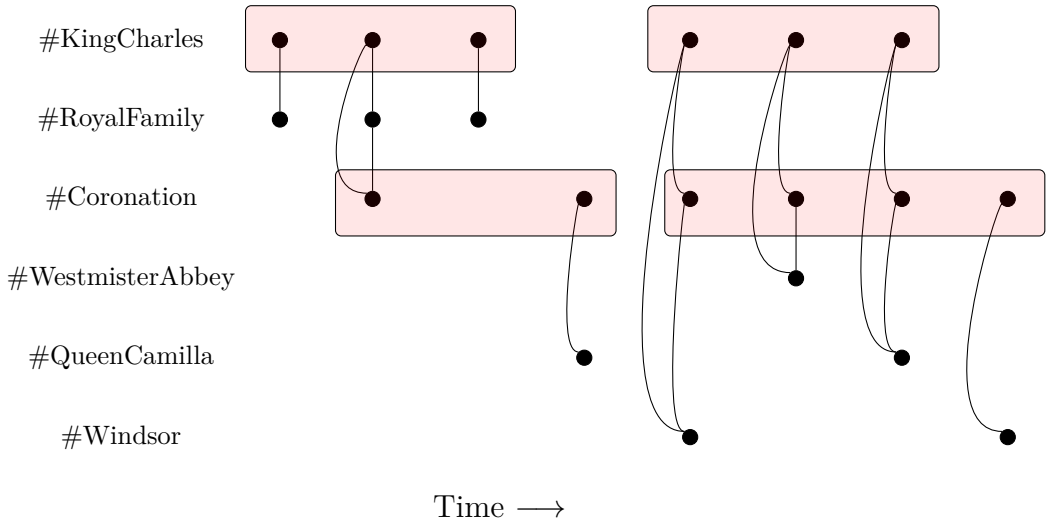


Figure 1 An example of k-MinTimelineCover with $k = 2$. Activity timelines (pink) show two intervals that are useful in summarizing temporal interactions, identifying the two main hashtags that explain the event.

We present an example to show how activity timelines can be useful in understanding evolution of events. Consider the King Charles Coronation in May 2023. Figure 1 shows temporal co-occurrence of hashtag related to this event. The timelines, associated indicated in pink, helps to identify the main topics in the online discussions.

Several papers have studied the complexity of k-MinTimelineCover [15, 6, 3, 2], also in some restricted variant. In particular, the case $k = 1$, where the activity timeline of each vertex is defined as a single interval, has been deeply studied, due also to the link with a foundational problem in theoretical computer science, that is Minimum Vertex Cover. For $k = 1$ k-MinTimelineCover is NP-hard and hard to approximate within constant factor [15, 6, 3, 2]. For $k \geq 2$, the problem is even harder, as even deciding if there exists a solution (of any span) is an NP-complete problem [15].

Due to the hardness of the problem and of its restrictions, some heuristic methods have been developed, for the general problem [15, 14] and for the restriction with $k = 1$ [15, 14, 11]. Only the paper that has introduced k-MinTimelineCover has designed practical methods for the problem [15] when $k \geq 2$, thus in this contribution we focus on designing methods for it. First, we design an ILP method that is able to solve the problem for moderate size instances, then we present a polynomial-time heuristic for k-MinTimelineCover, based on local search, so that it is applicable even on larger temporal graphs. In order to design our heuristic, we present a dynamic programming algorithm for a related problem, called k-MinIntervalCover, that runs in polynomial time.

We consider the performance of our methods on synthetic datasets, both in for the efficiency and the quality of the solutions returned. For this latter aspect we show that our heuristic, is able to shrink significantly the activity timeline lengths, with respect to the methods presented in [15].

1.1 Related Works

The complexity of k-MinTimelineCover, including the parameterized and approximation complexity, has been considerably investigated. k-MinTimelineCover is NP-hard, even for many restrictions. Rozenstein et al. have shown that it is NP-complete to decide whether

the exists a solution of any span [15]; this implies that is not possible to approximate the problem within any factor. Also for $k = 2$ it is NP-complete to decide whether there exists a solution of k-MinTimelineCover that has span equal to 0 [6].

The case $k = 1$ is known NP-hard also when temporal graphs have some restrictions: at most one temporal edge is defined in each timestamp [2], the time domain consists of three timestamps and each vertex has degree bounded by two [2], the time domain consists of two timestamps [6].

k-MinTimelineCover have also been considered through the lens of parameterized complexity, for parameters the number of vertices, the number of timestamps in the time domain, and k [6]. For $k = 1$, k-MinTimelineCover has been show to be fixed-parameter tractable, for parameter the span of a solution, when the time domain consists of two timestamps [6] and later for any number of timestamps [3]. It is unlikely that the previous result can be extended to larger values of k, since for $k = 2$ it is NP-complete to decide whether there exists a solution of k-MinTimelineCover that has span equal to 0 [6].

As for the approximation complexity, we have already noted that k-MinTimelineCover cannot be approximated within any factor. For $k = 1$, while the problem is known to be hard to approximate within constant factor assuming the Unique Game Conjecture [6, 3], it admits a $O(T \log |V|)$-approximation algorithm [4, 5] (T is the number of timestamps of the time domain, V is the set of vertices).

As for heuristics, Rozenshtein et al. [14] have presented **k-Inner**, which is based on solving a problem, called k-Coalesce, that requires to cover some predetermined timestamps. These timestamps represent an estimation of the activity timeline of a vertex. In our contribution, we present a local search heuristic which is meant to be built on top of this heuristic, in order possibly improve the computed solutions (that is we are able to reduce the span). Other heuristics have been proposed recently for the case $k = 1$ [11, 10]. We note that other complexity results and a heuristic, called **k-Budget**, have been given for the problem that has as objective function the minimization of the maximum span [15, 6].

The rest of the paper is organized as follow. In Section 2 we give some definitions and we formally introduce the k-MinTimelineCover problem. In Section 3 we present the ILP formulation for k-MinTimelineCover, then in Section 4 we describe our local search heuristic. In Section 5 we give an experimental evaluation of our methods on synthetic datasets. We conclude the paper with some future directions described in Section 6.

2 Preliminaries

A temporal graph $G = (V, E)$, with V a set of vertices and E a set of temporal edges, which are defined on time domain $[T] = [1, T]$ of timestamps (note that in the model we consider the time domain is discrete and T represents the maximum timestamp value). Each temporal edge is represented by a timestamped triplet (u, v, t) such that $u, v \in V$ and $t \in [T]$ represents the time when the interaction between u and v occurred. Note that in the temporal graphs we consider, temporal edges are undirected and unweighted.

Consider a vertex v and two timestamps $s_v \in [T]$ and $e_v \in [T]$ with $s_v \leq e_v$, $I_v = [s_v, e_v]$ is an activity interval of v. An *activity timeline* $\mathcal{T}_k(v)$ of a vertex $v \in V$ is a set of k (where $k \geq 1$) activity intervals $I_{v_i} = [s_{v_i}, e_{v_i}], i \in [k]$, where $s_{v_i} \leq e_{v_i}$, with the constraint that $e_{v_i} < s_{v_{i+1}}$ for each $i \in [k-1]$. The last constraint guarantees that the activity intervals of an activity timeline of a vertex are time disjoint. Given a vertex $v \in V$ and an activity timeline $\mathcal{T}_k(v)$ of v, we say that v is *active* in a timestamp t that is included in an interval of $\mathcal{T}_k(v)$; if t is not included in an interval of $\mathcal{T}_k(v)$, v is said to be *inactive* in t.

Moreover, we define the *span* of interval I_{v_i} $\delta(I_{v_i}) = e_{v_i} - s_{v_i}$ as the duration of the interval i for the vertex v. We define the span of an activity timeline $\mathcal{T}_k(v)$ of v, consisting of interval $I_{v_i} = [s_{v_i}, e_{v_i}], i \in [k]$, as

$$S(\mathcal{T}_k(v)) = \sum_{i=1}^{k} \delta(I_{v_i}).$$

The activity timeline of temporal graph G is defined as:

$$\mathcal{T}_k = \bigcup_{v \in V} \mathcal{T}_k(v)$$

The *sum span* of the activity timeline $\mathcal{T}_k$ of a graph G is defined as the sum spam over all vertices in the temporal graph:

$$S(\mathcal{T}_k) = \sum_{v \in V} S(\mathcal{T}_k(v)) = \sum_{v \in V} \sum_{j \in [1 \ldots k]} \delta(I_{v_i})$$

An important fact to note is that a vertex can have at most k activity intervals, so the problem setting admits empty intervals.

Now, we define how an activity timeline covers temporal edges of a temporal graph.

▶ **Definition 1.** *Given a temporal graph $G = (V, E)$ and an activity timeline $\mathcal{T}_k$ of G, $\mathcal{T}_k$ covers G if $\forall (u, v, t) \in E$, there is an interval $I_v \in \mathcal{T}_k(v)$ or $I_u \in \mathcal{T}_k(u)$ such that $t \in I_v$ or $t \in I_u$.*

Now, we are able to formally define the problem we are interested in.

▶ **Problem 1.** *k-Minimum Timeline Cover (k-MinTimelineCover)*
Input: *A temporal graph $G = (V, E)$, $k \in \mathbb{N}$.*
Output: *An activity timeline $\mathcal{T}_k$ that covers G and minimizes the total sum span $S(\mathcal{T}_k)$.*

Note that in the following, we will focus on the case $k \geq 2$.

Figure 2 shows an example of timeline covering over a 2-timestamps graph with 5 vertices.

3 An ILP Formulation for k-MinTimelineCover

In the following, we present an ILP model formulation for k-MinTimelineCover. Let $G = (V, E)$ be a temporal graph, and k be the number of activity intervals per vertex. The ILP model is based on the following variables. Binary variables x_v^t, $v \in V, t \in [T]$, are used to define the timestamp that belongs to an activity timeline of v; $x_v^t = 1$ if and only if vertex v is active at time t. The ILP formulation also defines variables y_v^t, $v \in V, t \in [T]$, which are used to guarantee that the activity timeline of v consists of k intervals. $y_v^t = 1$ if and only if vertex $v \in V$ is defined active (inactive, respectively) at time t, $t \in [T-1]$, and becomes inactive (active, respectively) at time $t+1$.

▶ **Theorem 2.** *Let $G = (V, E)$ be a temporal graph, and k be the number of maximum activity intervals per node. The activity timeline $\mathcal{T}_k$ that covers G and minimizes the total sum span $S(\mathcal{T}_k)$ is the solution to the following integer linear programming formulation:*

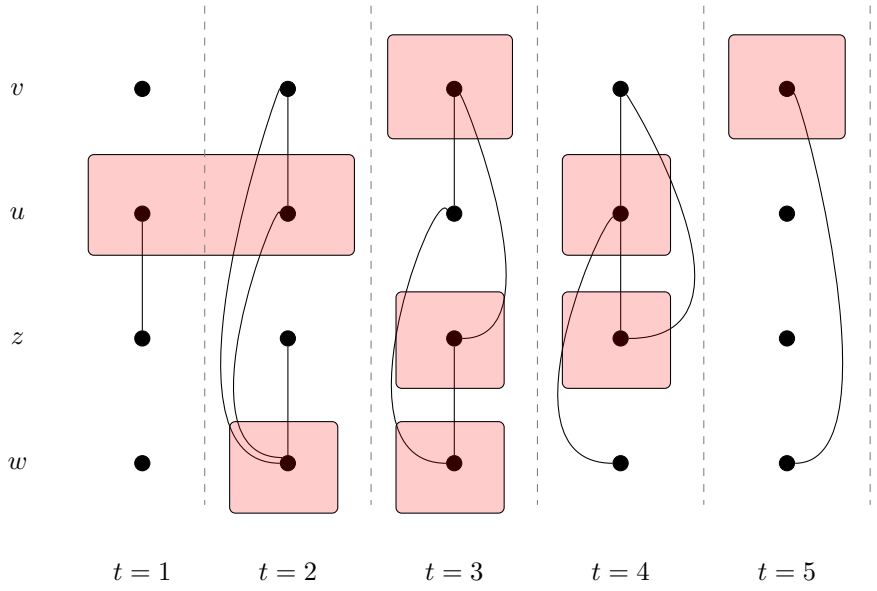


Figure 2 An example of k-MinTimelineCover with $k = 2$. The activity timeline of each vertex is represented with two pink rectangles. Note that the only vertex that has positive span is u, all the other vertices have span equal to 0. Every temporal edge is covered by at least an activity timeline.

$$min(\sum_{v \in V}(\sum_{t=1}^{T} x_v^t - k)) \tag{1}$$

s. t.

$$x_v^t + x_u^t \geq 1, \forall (u, v, t) \in E \tag{2}$$

$$\sum_{t=1}^{T} y_v^t \leq 2 * k, \forall v \in V \tag{3}$$

where $y_v^t = 0$ *if* x_v^t *and* x_v^{t+1} *have the same state and* $y_v^t = 1$ *otherwise*

$$\sum_{t=1}^{T} x_v^t \geq k, \forall v \in V \tag{4}$$

Proof. We now prove the correctness of the ILP model. The first constraint of the ILP formulation (which is represented by Equation 2) guarantees that for every temporal edge $(u, v, t) \in E$, at least one of the endpoints is active at timestamp t, which shows that the timeline activity output by the ILP covers each temporal edge (u, v, t). Constraint 3 introduces another binary variable y_v^t which does not allow us to have more than k disjoint intervals in a timeline activity of each vertex. Constraint 4 is explained more formally by the following inequalities:

$y_v^t \geq x_v^t - x_v^{t+1} \forall v \in V, t \in [T-1]$
$y_v^t \geq x_v^{t+1} - x_v^t \forall v \in V, t \in [T-1]$
$y_v^0 = 0$
$y_v^T = 0$

Each interval of an activity timeline of a vertex is defined by a start and an end timestamp, consisting of two switches. A timestamp is defined as a switch - represented by $y_v^t = 1$ - if the vertex either becomes active or inactive at that respective timestamp. Thus, the total number of switches for every vertex v can be at most $2k$, because each vertex will become active - and then inactive k times.

The last constraint (Equation 4) ensures that for each vertex, its activity interval contains at least k different active timestamps. This ensures that in the objective function (Equation 1) no term is negative. Note that the assumption that each activity interval contains at least k different active timestamps is not restrictive, as an activity interval containing one timestamp has span 0, thus we can possibly create one-timestamp intervals without increasing the span. Also, note that we assume that we have two timestamps, 0 and $T + 1$, where no temporal edge is defined, and we added fixed values $y_v^0 = y_v^{T+1} = 0$. This makes sure that, for cases when it happens, activation of the first timestamp and the deactivation of the last timestamp are considered in the sum of Constraint 3. The objective function corresponds to minimize the sum of the lengths of all active intervals across all vertices which is equivalent to the sum span defined in the preliminaries section:

$$S(\mathcal{T}_k) = \sum_{v \in V} S(\mathcal{T}_k(v))) = \sum_{j \in [1 \ldots k]} \sum_{v \in V} \delta(I_{v_i})$$

Basically, for every vertex v we create k timestamp intervals; we sum up the number of timestamps of these k intervals from which we subtract k in order ensure that the ILP model respects the definition of span. For example if we need to have a vertex active at all times, that will be interpreted by the y activation variable as a single interval I. But by subtracting k, we are sure that this interval is split into k subintervals; indeed while I has a span of $T - 1$, the k subintervals have a total span of $T - k$, which has a lower span with respect to I. To sum up, a solution to the ILP provides a minimum temporal activity timeline for each vertex, constrained to at most k intervals, which ensures that all temporal edges are covered. ◀

4 A Local Search Heuristic

In this section, we present a heuristic based on local search and on a polynomial-time algorithm for a subproblem of k-MinTimelineCover. The motivation behind the use of the local search technique is that for large graph instances and a large k, we have observed that along the intervals produced by the heuristic solutions there were many which could be left out of the solution without affecting the correctness of k-MinTimelineCover.

Before we introduce the steps of the local search procedure (see below), we introduce the concept of *loss* inspired by the paper of Lazzarinetti et al. [10]. For each vertex $v \in V$ at timestamp $t \in [T]$ we define its associated loss $loss(v, t)$ as the number of edges covered that would become uncovered if v becomes inactive at time t. Given an interval $I(v)$ of an activity timeline $\mathcal{T}(v)$ of a vertex v, the interval loss $loss(I(v))$ is defined as:

$$loss(I(v)) = \sum_{t \in I(v)} loss(v, t).$$

Our improvement algorithm improves the solution provided by the heuristics in [15] and refines it using local search. The local search algorithm: (1) finds for each vertex v if there exists an activity interval of $loss = 0$ in the solution; (2) it removes the activity timeline $\mathcal{T}_k(v)$ from the solution; (3) it computes the timestamps associated with the temporal edges incident in v left uncovered by the pruning of step (2); (4) it computes by dynamic programming algorithm an activity timeline of v of minimum span, containing the timestamps of step (3).

The motivation for this strategy is that, if there is an interval $I(v)$ of $loss = 0$, it is not useful to cover temporal edges, instead of removing only $I(v)$, we decide to remove all the activity timeline $\mathcal{T}_k(v)$. This allows us to define a new activity timeline for v with k intervals. Basically, by reconfiguring the whole activity interval, we can find a new structure for ordering the activity timestamps which would allow for a smaller span than the one which would result by only deleting a single activity interval. Even though the strategy produces additional cost regarding computation, it can produce a much better solution because it saves not only the span of the deleted interval, but also the span saved by reshuffling the activity timestamps into new intervals.

For example, assume that $\mathcal{T}_k(v)$ contains another interval $I'(v)$ of positive loss, such that $I'(v)$ covers a temporal edge (u, v, t). If (u, v, t) is covered also by an interval of vertex u, we don't require to cover (u, v, t) when we recompute the timeline activity of v. Note that the activity timeline computed at point (4) does not increase the span with respect to $\mathcal{T}_k(v)$.

We begin by describing how our heuristic deals with steps (1) - (3), by constructing a specialized data structure, to which we will refer as loss dictionary, based on the initial solution. This structure associates each activity interval of every vertex with a corresponding loss value, which is initialized to zero. For each temporal edge in G, we examine whether it is covered in the initial solution by both incident vertices or only by one. If the edge is covered by both vertices, no modification is made in the loss dictionary. However, if it is covered by only one vertex, this implies that the vertex must be active at that specific timestamp, otherwise the temporal edge would become uncovered. Consequently, we increment the loss value $loss(I(v))$ of the activity interval of the corresponding vertex by one. After iterating through all temporal edges and updating the loss dictionary accordingly, we identify all vertices that possess at least one activity interval with a loss value of zero. These intervals are deemed redundant and subsequently, for that vertex we recalculate its activity timeline from scratch, by parsing through the timestamps and looking for the uncovered timestamped edges.

Following this pruning step, we recompute the activity timeline for that vertex using a dynamic programming algorithm. We must specify the fact that indeed, the pruning step can be done also for vertices not associated with an activity interval of $loss = 0$. We tested this on relatively small synthetically generated datasets and the result was very similar to the result of our approach, because for most of the vertices which do not contain $loss = 0$ intervals, the recalculation process is not able to shrink the activity timeline.

Now, we formally present the dynamic programming step (4), which solves the following problem.

▶ **Problem 2.** *k-Min-Interval-Cover (k-MinIntervalCover)*
Input: *An ordered set $\mathcal{U}$ of timestamps, $k \in \mathbb{N}$.*
Output: *A sequence $\mathcal{I}$ of k intervals of minimum span such that each timestamp in $\mathcal{U}$ is covered .*

Note that, given $i \in [z]$, we denote by $\mathcal{U}[i]$ the i-th timestamp in $\mathcal{U}$. Now, we give the dynamic programming recurrence to solve in polynomial time k-MinIntervalCover. Let $z = |U|$. Define $DP[i][j]$, where $i \in [z]$ and $j \in [k]$, as the minimum span of a sequence of j intervals in $[\mathcal{U}[i],\mathcal{U}[z]]$ that covers each timestamp in $\mathcal{U}$ from the i-th to the z-th one. Furthermore, we assume that z is a timestamp that contains no temporal edge and thus not need to be covered. This is used to allow some interval to be empty. Also note that empty intervals do not need to be time disjoint. The recurrence of compute $DP[i][j]$, where $i \in [z]$ and $j \in [k]$, is defined in the following.

$$DP[i][j] = min_{i<l\leq z}\{DP[l][j-1] + \mathcal{U}[l-1] - \mathcal{U}[i]) \quad \text{with } j \geq 1 \tag{5}$$

For the base cases $DP[i][0] = +\infty$, for each $i \in z$; $DP[z][0] = 0$.

Next, we prove the correctness of Recurrence 5.

▶ **Theorem 3.** *$DP[i][j] = s$ if and only if there exists sequence of j intervals in $[i, T]$ that covers each timestamp in $\mathcal{U}$ from $\mathcal{U}[i]$ to z and has minimum span s.*

Proof. We prove the theorem by induction on j. The lemma holds for $j = 0$, as an empty sequence of intervals does not cover any timestamp.

Assume that the lemma holds for $j-1$, we prove it for j. Consider a sequence of j intervals in $[\mathcal{U}[i],\mathcal{U}[z]]$ that covers timestamps from $\mathcal{U}[i]$ to $\mathcal{U}[z]$ and has minimum span s. There is an interval $I = [\mathcal{U}[i],\mathcal{U}[l-1]]$, with $l > i$, that covers all the timestamps between $\mathcal{U}[i]$ and $\mathcal{U}[l-1]$; moreover, there is a sequence of $j-1$ intervals that covers timestamps between $\mathcal{U}[l]$ and $\mathcal{U}[z]$ and has total span s', where $s = s' + h - i$. By induction $DP[h+1][j-1] = s'$, and thus $DP[i][j] = s$.

Assume that $DP[i][j] = s$. Then since $j \geq 1$, it follows that there exists a value $l \geq j$ such that $DP[i][j] = min_{l>i}\{DP[l][j-1] + \mathcal{U}[l-1] - \mathcal{U}[i])$. By induction hypothesis, there is a sequence of $j-1$ intervals in $[\mathcal{U}[i][l],\mathcal{U}[z]]$ that covers each timestamp between $\mathcal{U}[l]$ to $\mathcal{U}[z]$ and have span $DP[l][j-1]$. By adding the interval $[\mathcal{U}[i],\mathcal{U}[l-1]]$ to the sequence, we obtain a sequence of intervals that covers each timestamp in $\mathcal{U}[i]$ to $\mathcal{U}[z]$ and has minimum span s. ◀

The pseudocode of the dynamic programming is described in Algorithm 1.

Next, we prove two properties of the local search procedure: (1) that it covers each temporal edge and (2) that it never increases the span of a timeline activity.

▶ **Lemma 4.** *Consider a vertex v and a sequence of intervals $I_1, I_2, \dots I_m$, $m \leq k$ of span s. From those intervals we extract an ordered set $\mathcal{U}$ of timestamps, used as input to Algorithm 1. Then Algorithm 1 produces a sequence of intervals $I'_1, I'_2, \dots I'_k$ of span at most s, such that all timestamps are covered.*

Proof. Let $\mathcal{T}_k$ be a solution of k-MinTimelineCover on instance G and a vertex v having an interval of loss equal to 0 in $\mathcal{T}_k$. The local search heuristic is applied to v. First note that $DP[z][k]$ outputs the span of a sequence of intervals of minimum span that covers all the timestamps in $\mathcal{U}$. Since each timestamp of $\mathcal{U}$ is covered, then the corresponding intervals define an activity timeline of v, thus with the activity timelines of other vertices, a feasible solution of k-MinTimelineCover, as each temporal edge is covered. Furthermore, note that the span of intervals $I_1, I_2, \dots I_m$ must be at least s, otherwise they will induce a sequence of intervals covering the timestamps in $\mathcal{U}$ and having span less than $DP[z][k]$, contradicting Theorem 3. Finally, each temporal edge incident in v and not defined in a timestamp of $\mathcal{U}$ is covered by the solution $\mathcal{T}_k$ of k-MinTimelineCover, thus concluding the proof. ◀

Finally, we present the complexity of the Dynamic Programming algorithm which can solve k-MinIntervalCover.

▶ **Lemma 5.** *Algorithm 1 computes the optimal intervals and minimum total interval length in $\mathcal{O}(n^2 * k)$ time, where n represents the total number of timestamps in the input list $\mathcal{U}$, and k is the maximum number of intervals.*

Proof. Let n be the total number of timestamps in $\mathcal{U}$. The DP table dp thus will have dimensions $(n+1)*(k+1)$. The algorithm can be broken into 3 main components: (1) initialization, (2) base case setup and (3) the DP Computation. (1) involves iterating through all $(n+1)(k+1)$ elements, taking $\mathcal{O}(n*k)$ time. (2) involves asigning base case variables $dp[n][i], i \in k$ and takes $\mathcal{O}(k)$ time. (3) consists of three nested loops, which form the core of the computation. Because each operation within the innermost loop takes $\mathcal{O}(1)$, our total running time will be $\mathcal{O}(n^2 * k)$. Summing the time complexities of these three stages will be equal to the dominant term in the sum, which is $\mathcal{O}(n^2 * k)$ and represents a polynomial time algorithm. ◀

Algorithm 1 Dynamic programming solution for ordering the timestamps for vertices with 0 loss.

Input: A sorted list of timestamps $t[1 \dots n]$, k
Output: Optimal intervals and minimum total interval length

```
dp[0..T][0..k] ← ∞                                   ▷ Initialize DP table
prev[0..T][0..k] ← 0                                 ▷ For solution reconstruction
dp[T][0] ← 0                                         ▷ Base case: empty sequence
for j ← 1 to k do
    dp[T][j] ← 0                                     ▷ Empty sequence with any intervals
end for
for i ← T − 1 down to 0 do
    for j ← 1 to k do
        for l ← i + 1 to T do
            if l = i + 1 then
                intervalLength ← 0
            else
                intervalLength ← timestamps[l − 1] − timestamps[i] + 1
            end if
            totalLength ← intervalLength + dp[l][j − 1]
            if totalLength < dp[i][j] then
                dp[i][j] ← totalLength
                prev[i][j] ← l
            end if
        end for
    end for
end for
Reconstruct the dp matrix
return intervals, dp[0][k]
```

5 Experiments

We begin this section by describing how we create the synthetically generated dataset on which we will test both the ILP model and the heuristic. It is based on the generated dataset of [14] and for each iteration of our algorithms we consider a static underlying graph structure $G = (V, E)$. For each $v \in V$ we simulate a set of temporal interaction intervals at different timestamps. The parameters controlling the generation process include the number of interaction intervals per vertex, the distance between events, the inter-interval spacing, and the degree of temporal overlap between consecutive intervals. For simplicity

and replicability, parameters were held constant across iterations. The generator ensures stochastic yet reproducible behavior through a fixed random seed. The resulting dataset consists of a chronologically ordered list of timestamped interactions (u, v, t), as well as a mapping of vertices to their corresponding k active time intervals. Experiments were conducted on a Macbook Pro machine with 11 CPU cores, 18 GB RAM, and $M3$ *ARM* processor to ensure consistency in runtime comparisons.

We evaluate the performance of our Integer Linear Programming (ILP) formulation on synthetically generated temporal networks. We used a Gurobi solver in its current configuration, the model scales effectively to instances comprising up to $50,000$ temporal edges, with total runtime remaining under 25 minutes. The formulation demonstrates optimal performance in scenarios characterized by a relatively small number of activity intervals per vertex. For our standard benchmarking experiments, we fixed the number of activity intervals at $k = 10$ and used an overlap coefficient of 0.5. Additionally, to assess the scalability of the approach under alternative structural conditions, we conducted a separate test with a reduced k value and a larger graph containing more vertices and edges. Empirically, we observed that lower overlap coefficients lead to faster computational times; nonetheless, we maintained a minimum overlap of 0.5 in most experiments to better reflect real-world temporal dynamics. Across all tested configurations, the ILP formulation consistently achieved high solution accuracy, with both precision and recall exceeding 99%, indicating the correctness and robustness of the approach.

Figure 3 ILP Execution Time vs the total number of timestamps.

In the following, we compare the results of the *k-Baseline* and *k-Inner* heuristics of [14] with their improved versions using the heuristic algorithm. We will do a separate analysis for each of the considered algorithms. Starting with k-Baseline, we observe that, precision-wise the results are similar. This apparent discrepancy occurs because both sets of intervals are being compared against the ground-truth results, which remain considerably smaller in magnitude. The standard comparison metrics do not fully capture the practical significance of our improvements because both the initial approach and our method necessarily produce solutions that deviate substantially from the optimal (given the NP-hard nature of the problem and its known approximation hardness of any factor). Nevertheless, when examining the total interval length across all vertex variations, we observe a clear and consistent improvement. This improvement stems directly from our algorithm (as detailed in Section 5), which calculates the loss of every interval for each vertex and, for vertices containing intervals with zero loss, recalculates the activity timestamps and optimally reorders them using dynamic programming.

Comparative analysis of our approach against the baseline reveals consistent improvements in precision and recall metrics, which are represented by an average improvement that exceeds 2.5% and peak differentials of more than 5% in maximum values. We use Figure 4 for reference. Other relevant average comparisons are for recall, which stays the same for the Baseline and the Heuristic Improvement algorithm at 0.65. More significantly, our method demonstrates substantial optimization of the total interval length, reducing it from $3.0 * 1e6$ to $2.5 * 1e6$, which constitutes a reduction of 16.7%. This considerable decrease in total length, coupled with the consistent precision improvements, underscores the robustness and practicality of our algorithmic refinements.

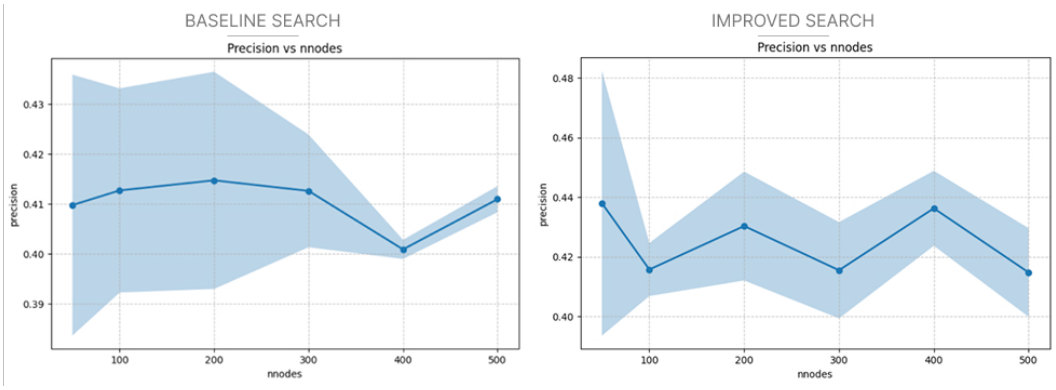


Figure 4 Precision Comparison between Baseline and Improved Heuristic.

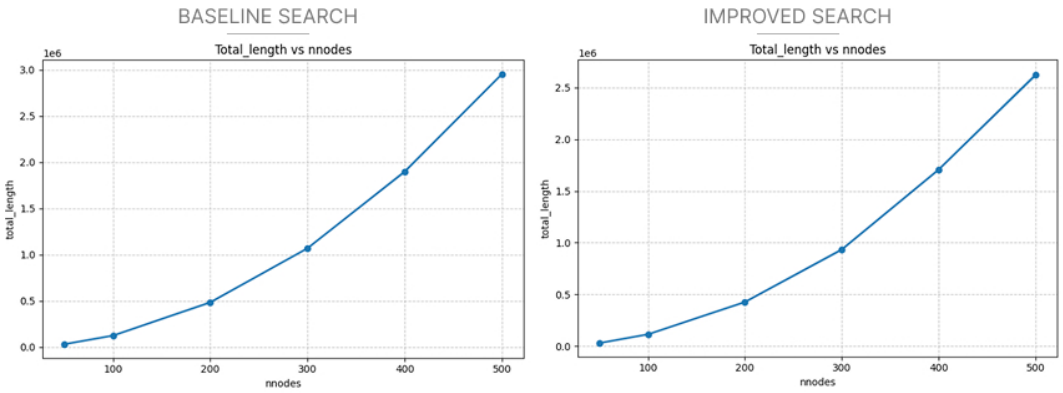


Figure 5 Total Length Comparison between Baseline and Improved Heuristic.

In the experimental phase aimed at enhancing the *k-Inner* algorithm, we adopted a slightly modified approach. Specifically, we conducted a dry run for all vertices $v \in V$ with a loss value of zero, retaining the newly generated activity intervals only if they resulted in a lower total cost compared to the baseline established by the original *k-Inner* algorithm. This conservative strategy was employed as a safeguard, given that the original heuristic already achieves a high precision of approximately 75%. In earlier iterations, we observed instances where modifying certain vertices adversely affected the interval structure, leading to significant reductions in overall precision. Despite these challenges, our modified heuristic yields slight but consistent improvements. However, given the already high baseline precision, these gains typically fall within a margin of approximately 1%. To ensure robustness in

evaluation, we extended the number of iterations and increased the total number of vertices analyzed. As shown in Figure 6, our approach surpasses the 75% precision threshold for a limited number of vertices, after which the performance aligns closely with that of the original *k-Inner* algorithm. This convergence is expected, as the algorithm already performs near-optimally, leaving few opportunities for zero-loss interval modifications. Consequently, the impact on the total interval length remains minimal.

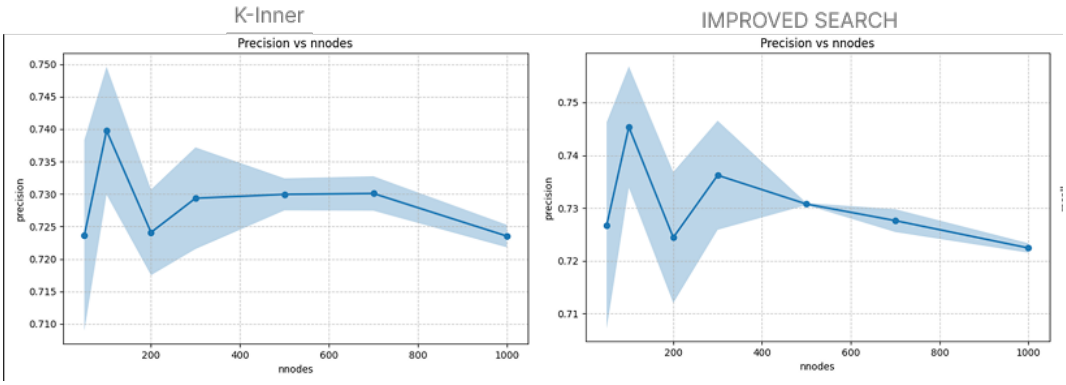


Figure 6 Total Length Comparison between Baseline and Improved Heuristic.

6 Conclusion and Future Works

We have considered k-MinTimelineCover, a problem defined for complex data summarization. We have presented an ILP formulation for moderate size instances and we have designed an efficient heuristic based on local search. We have presented an experimental evaluation on synthetic data sets for both methods. Future works include an experimental evaluation on real-word datasets and an investigation of the performance of variants of our heuristics that extend the local search approach, for example when more than one vertex containing an interval of loss equal to 0 is considered in this step.

References

1 Argyrios Deligkas, Michelle Döring, Eduard Eiben, Tiger-Lily Goldsmith, and George Skretas. Being an influencer is hard: The complexity of influence maximization in temporal graphs with a fixed source. *Inf. Comput.*, 299:105171, 2024. doi:10.1016/J.IC.2024.105171.

2 Riccardo Dondi. Untangling temporal graphs of bounded degree. *Theor. Comput. Sci.*, 969:114040, 2023. doi:10.1016/J.TCS.2023.114040.

3 Riccardo Dondi and Manuel Lafond. An FPT algorithm for temporal graph untangling. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.IPEC.2023.12.

4 Riccardo Dondi and Alexandru Popa. Timeline cover in temporal graphs: Exact and approximation algorithms. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2023. doi:10.1007/978-3-031-34347-6_15.

5 Riccardo Dondi and Alexandru Popa. Exact and approximation algorithms for covering timeline in temporal graphs. *Annals of Operations Research*, April 2024. doi:10.1007/s10479-024-05993-8.

6 Vincent Froese, Pascal Kunz, and Philipp Zschoche. Disentangling the computational complexity of network untangling. *CoRR*, abs/2204.02668, 2022. doi:10.48550/arXiv.2204.02668.

7 Petter Holme and Jari Saramäki. Temporal networks. *CoRR*, abs/1108.1780, 2011. arXiv:1108.1780.

8 Mohammad Mehdi Hosseinzadeh, Mario Cannataro, Pietro Hiram Guzzi, and Riccardo Dondi. Temporal networks in biology and medicine: a survey on models, algorithms, and tools. *Netw. Model. Anal. Health Informatics Bioinform.*, 12(1):10, 2023. doi:10.1007/S13721-022-00406-X.

9 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002. doi:10.1006/JCSS.2002.1829.

10 Giorgio Lazzarinetti, Riccardo Dondi, Sara Manzoni, and Italo Zoppis. DLMinTC+: A deep learning based algorithm for minimum timeline cover on temporal graphs. *Algorithms*, 18(2):113, 2025.

11 Giorgio Lazzarinetti, Sara Manzoni, Italo Zoppis, and Riccardo Dondi. FastMinTC+: A fast and effective heuristic for minimum timeline cover on temporal networks. In Pietro Sala, Michael Sioutis, and Fusheng Wang, editors, *31st International Symposium on Temporal Representation and Reasoning, TIME 2024, October 28-30, 2024, Montpellier, France*, volume 318 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.TIME.2024.20.

12 Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM Comput. Surv.*, 51(3):62:1–62:34, 2018. doi:10.1145/3186727.

13 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.

14 Polina Rozenshtein, Francesco Bonchi, Aristides Gionis, Mauro Sozio, and Nikolaj Tatti. Finding events in temporal networks: segmentation meets densest subgraph discovery. *Knowl. Inf. Syst.*, 62(4):1611–1639, 2020. doi:10.1007/S10115-019-01403-9.

15 Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. The network-untangling problem: from interactions to activity timelines. *Data Min. Knowl. Discov.*, 35(1):213–247, 2021. doi:10.1007/S10618-020-00717-5.

16 Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1055–1064. ACM, 2015. doi:10.1145/2783258.2783321.

17 John Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Temporal distance metrics for social network analysis. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 31–36, 2009.

Temporal GraphQL: A Tree Grammar Approach

Curtis E. Dyreson
Department of Computer Science, Utah State University, Logan, UT, USA

Bishal Sarkar
Department of Computer Science, Utah State University, Logan, UT, USA

Abstract

This paper presents a novel system, called Temporal GraphQL, for supporting temporal data in web services. A temporal web service is a service that provides a *temporal* view of data, that is, a view of the current data as well as past or future states of the data. Capturing the history of the data is important in data forensics, data auditing, and subscriptions, where an application continuously reads data. GraphQL is a technology for improving the development and management of web services. Originally developed by Facebook and widely used in industry, GraphQL is a query language for web services. This paper introduces Temporal GraphQL. We show how to use tree grammars to model GraphQL schemas, data, and queries, and propose temporal tree grammars to model Temporal GraphQL. We extend GraphQL with temporal snapshot, slice, and delta operators. To the best of our knowledge, this is the first work on Temporal GraphQL and temporal tree grammars.

2012 ACM Subject Classification Information systems → Temporal data; Information systems → Service discovery and interfaces; Information systems → Query languages

Keywords and phrases Temporal databases, temporal queries, GraphQL, web services

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.9

1 Introduction

Web applications rely on *web services* for data management. A web service is a an *application programming interface* (API) endpoint that allows a client to interact with a back-end database over the web. Web services are ubiquitous; when on-line users shop, make dinner reservations, buy airline tickets, vote, or post social media updates each interaction typically invokes several web services. Web services read and write data formatted in Javascript Objection Notation (JSON). JSON is a lightweight, text notation for representing objects. Though JSON DBMSs are rising in popularity, e.g., MongoDB ranks fifth in a recent ranking of DBMS popularity [15], JSON is the most widely used data exchange language. JSON is tightly integrated into many modern programming languages, e.g., Python, Java, Typescript, all have libraries to quickly convert objects formatted in JSON to objects in the host language and vice-versa.

GraphQL is a technology for improving the development and management of web services [22]. Originally developed by Facebook and widely used in industry, GraphQL is a query language for a web service, or more generally, an API. GraphQL supports *queries* that read data as well as *mutations* that update data on the types provided by the schema. A GraphQL query is evaluated to produce JSON data requested by a user.

GraphQL, however, lacks support for temporal data. Temporal data is data annotated with time metadata. This paper presents a novel system, called Temporal GraphQL, for supporting temporal data in web services. A temporal web service is a service that provides a *temporal* view of data, that is, a view of the current data as well as past or future states of the data. Capturing the history of the data is important in data forensics, data auditing, and subscriptions, where an application continuously reads data. For a subscription, instead of returning all of the data in each snapshot, only the differences between snapshots can be provided. This "delta" is usually much smaller than the entire dataset.

This paper makes the following contributions.

- We show how to use tree grammars to model GraphQL schemas, data, and queries, and propose temporal tree grammars to model temporal GraphQL. To the best of our knowledge, this is the first work on temporal tree grammars.
- We extend GraphQL with temporal snapshot, slice, and delta operators.

This paper is organized as follows. The next section reviews GraphQL. Section 3 introduces Temporal GraphQL. We then describe tree grammars and how they are used in supporting Temporal GraphQL in Section 4. The final two sections cover related work and conclusions.

2 Review of GraphQL

In this section we review GraphQL. The starting point for GraphQL is a *schema*, which describes the types provided by an API as described in Section 2.1. GraphQL supports queries and mutations (updates) using the API. For our purposes mutations are a variation on queries, so this paper focuses exclusively on queries, which are presented in Section 2.2.

2.1 GraphQL Schemas

A GraphQL schema describes the types provided by an API. As an example, consider the GraphQL schema specification shown in Figure 1. The schema is taken from GraphQL's tutorial for learning GraphQL [23] and has three object *types*: `Character`, `Planet`, and `Species`. The miniworld for the types is a science fiction world where a character originates on some planet, is of some species, and has friends who are characters. Each type has one or more properties. A property represents a key-value pair in JSON. The name of the property is the key and the schema records type constraints on the value. The `Character` type has `name`, `friends`, `homeworld`, and `species` properties. The `Character` `name` is a `String` type and must be non-null (indicated by the "`!`"). The `friends` property is a list (indicated by the enclosing brackets "`[]`") of references to `Character`s.

The data is checked during query evaluation to ensure that it conforms to the schema, if not, an error is generated. As an example Figure 3 shows a fragment of a data instance that conforms to the schema given in Figure 1. Each JSON object implicitly contains an `id` property that uniquely identifies the object within the data collection, not just the type (collection-wide unique identifiers are used to implement client-side caching of objects). Sub-objects are represented as references, e.g., the `homeworld` property in the `Character` type is a reference to a `Planet` object.

2.2 Queries

Queries are at the core of GraphQL. Every GraphQL schema must also support an entry point for queries, this type is known as the `Query` type. The `Query` type specifies the root entry point(s) for the database. An example is given in Figure 2. The `Query` type has three entry points:

1. `hero` reads a `Character`,
2. `characters` yields a list `Character`s, and
3. `planets`, which returns a list of `Planet`s.

The entry point is the starting point for the query, more fields can be added to flesh out objects and sub-objects in a query result. An example is given in Figure 4. In the example, the `hero` entry point is expanded to include the `name` and `friends` properties. The `friends`

```
type Character {
  name: String!
  friends: [Character]
  homeworld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

Figure 1 A GraphQL schema for a science fiction database.

```
type Query {
  hero: Character
  characters: [Character]
  planets: [Planet]
}
```

Figure 2 The `Query` type in GraphQL defines query "entry points".

```
Character = [
  { "id": "id_r2_d2",
    "name": "R2-D2",
    "friends": [
      "characterId": "id_luke_skywalker",
      "characterId": "id_han_solo"
    ],
    "homeworldId": "id_naboo",
    "species": "id_droid" },
   { "id": "id_luke_skywalker",
     "name": "Luke Skywalker",
     ... },
   ...
]
Planet = [
  { "id": "id_naboo",
    "name": "Naboo",
    "climate": "Temperate" },
  ...
]
Species = [
  { "id": "id_droid",
     "name": "Droid",
    ... }
  ...
]
```

Figure 3 JSON that conforms to the GraphQL schema in Figure 1 for a science fiction database.

property is a list of `Characters`, and for each sub-object in the list the `name` and `homeworld` (which is a `Planet` type object) properties are selected. An example of the result of evaluating the query is given in Figure 5.

Queries can also include *filters*, which are predicates to test the data for membership in the result. For example, suppose that we want to select the character named Luke Skywalker. Then we could filter the `hero` entry point in a query as follows:

```
hero(filter: { name: { eq: "Luke Skywalker" } }) {
    name
    ...
}
```

Entry points can also be specified to take arguments. For instance, we could modify the `hero` entry point to match a specific name as follows.

```
type Query {
    hero(name: String) : Character
    ...
}
```

```
query {
  hero {
    name
    friends {
      name
      homeworld {
        name
        climate
      }
    }
  }
}
```

Figure 4 A query using the `hero` entry point.

```
{ "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {  "name": "Luke Skywalker",
            "homeworld": {
              "name": "Tatooine",
              "climate": "Desert"
            }
        },
        ....
      ]
  }
}
```

Figure 5 A fragment of the result of the GraphQL query in Figure 4.

The query to fetch Luke Skywalker would then be as follows.

```
hero(name: "Luke Skywalker") {
    name
    ...
}
```

3 Temporal GraphQL

Science fiction data changes over time as edits to the data are made and new data is inserted. This section describes how to capture the the changing history. In Temporal GraphQL the data is assumed to be annotated with time metadata that records the lifetime of the data in some temporal dimension. Common dimensions are transaction time and valid time. We first show how to add support for time to a schema type, we then describe several temporal query operators that let users travel in time and select past versions of data. In the next section we discuss how to support Temporal GraphQL in a layered approach where a temporal schema/operator is translated into the corresponding GraphQL schema/operator. This implementation strategy leverages non-temporal GraphQL to support GraphQL.

3.1 Temporal Types

In a temporal GraphQL schema a type can be made temporal by adding a GraphQL *directive* as shown in Figure 6. A GraphQL directive is prefixed with the "`@`" character. Directives are essentially decorators in many popular programming languages. Directives can be added to both schemas and queries. The `@temporal` directive indicates that the type is now a temporal type, that is the schema type will represent data annotated with temporal metadata. For the purposes of this paper we assume a single, transaction-time temporal dimension as it is common for systems to record the time data is created and deleted. Extensions to valid time, belief time, or bitemporal times are future work. We will further assume that the transaction time is recorded as a period or interval timestamp, rather than a temporal element, that

```
# @temporal directive declaration
directive @temporal() on OBJECT | SCHEMA

# Character is a temporal type
type Character @temporal {
  name: String
  friends: [Character]
  homeworld: Planet
  species: Species
}
...
```

Figure 6 A GraphQL schema for a temporal science fiction database.

```
query @slice({start: 3, stop: 4}) ( {
  hero {
    name
  }
}
```

Figure 7 An example `@slice` query.

is as a set of periods, or as an indeterminate time [2, 19]. Extending to handle temporal elements or temporal indeterminacy are left to future work. The schema can be a mix of temporal and nontemporal types, though for this running example we will assume that all of the object types have been similarly annotated.

3.2 Temporal Queries

We assume that an API or web service supplies the temporal data in a temporal GraphQL instance. Temporal GraphQL supports several kinds of *temporal queries*. Each kind of query is specified by a GraphQL *query directive* that modifies the behavior of the query as described below.

- `directive @snapshot(time: Int!) on QUERY` - A snapshot query takes as input a time, t, and returns the non-temporal data as of t.
- `directive @slice(time: Timestamp!) on QUERY` - A slice query takes a `Timestamp` object and returns the temporal history as of the given timestamp. The `Timestamp` type is defined in Figure 9.
- `directive @current() on QUERY` - The current query returns the snapshot as of the current time.
- `directive @delta(time: Timestamp!) on QUERY` - The delta query takes a `Timestamp` and returns all of the data that changed during the `Timestamp`.

Figure 7 shows how a slice query would be specified.

4 A Layered Approach to Supporting Temporal GraphQL

In this section we describe a layered approach to mapping a temporal schema (query) into a *representational* schema (query). The key to the approach is to model the schema as a *temporal tree grammar*. Section 4.1 provides background on tree grammars, which are extended in Section 4.2 to include support for time. The temporal tree grammar models the representational schema and data as described in Section 4.2.1.

4.1 A Tree Grammar Approach to Modeling GraphQL Schemas

Introduced in 1969, a tree grammar is a (context-free) grammar for generating (or parsing) trees [33].

▶ **Definition 1** (Simple Unordered Tree Grammar). *A simple unordered tree grammar is a four-tuple (Σ, R, N, Δ) where:*

- Σ *is the alphabet, a finite set of terminals;*
- N *is a finite set of nonterminals;*
- $R \in N$ *is a set of root (start) nonterminals, and*
- Δ *is a finite set of productions, with the following properties. Each production is of the form* $n \to x_1[\alpha_1]\beta_1 \ \ldots \ x_k[\alpha_k]\beta_k$ *where* $n \in N$*, the body of the production is unordered (the production represents all possible permutations of the body of the production), and for all* i*,*
  - $x_i \in \Sigma$*;*
  - α_i *is a well-formed formula of terminals or nonterminals, square brackets to indicate tree nesting, and metalanguage symbols from the EBNF (e.g.,* $*$ *represents Kleene closure);*
  - *and* β_i *is the empty string,* ? *to indicate 0 or 1 occurrences, or* $*$ *for Kleene closure.*

The grammar is for *unranked* trees [4]. In a *ranked* tree grammar each terminal or nonterminal would have a fixed arity or "rank", but GraphQL trees (and JSON) are best modeled by unranked tree grammars, so for instance a `homeworld` node may have between one and two descendants since `name` and `climate` are optional. The grammar is also context-free and simple because every clause in the body starts with a terminal (the root of a subtree) so a parser or generator for the grammar can deterministically choose the clause in the body of the production based on a single lookahead token (the evaluation only has to keep track of which clause was chosen).

One use of the tree grammar is to ensure that a GraphQL query is *valid*. Validation is the first step in query evaluation. Validation consists of checking the query against the schema to determine if all of the fields correspond to a type property and that the types are nested correctly. It is straightforward to convert a GraphQL schema to a tree grammar. The conversion introduces one production for every type (including the `Query` type). For example the converted tree grammar for the GraphQL schema of Figure 1 is given below. Note that a query is agnostic about JSON lists in the result so this distinction is not made in the grammar to validate a query.

- $\Sigma = \{$`id`, `name`, `friends`, `homeworld`, `species`, `climate`, `lifespan`, `origin`, `characters`, `hero`, `planets`$\}$
- $R = Q$
- $N = \{Q, C, P, S\}$
- Δ contains the following productions (for each $x[\alpha]\beta$ in the body β is a "?" because the clause is optional, so for clarity we will omit the "?").

  $Q \to$ `hero`$[C]$ `characters`$[S]$ `planets`$[P]$
  $C \to$ `id`[ ] `name`[ ] `friends`$[C]$ `homeworld`$[P]$ `species`$[S]$
  $P \to$ `id`[ ] `name`[ ] `climate`[ ]
  $S \to$ `id`[ ] `name`[ ] `lifespan`[ ] `origin`$[P]$

The grammar can be used to determine whether a query, such as that given in Figure 4 is valid. Figure 8 shows the tree representation of the query in Figure 4. The grammar is used to validate from the root down by first choosing to expand Q (the start nonterminal) by `hero`$[C]$ based on the root of the query matching `hero`. C is then expanded by `name`[ ] and `friends`$[C]$, and so on.

A GraphQL schema can also be converted to a *result* grammar that describes the result of a query. Below is the converted result grammar. Note that lists in the result are represented in a tree as repeated children (using Kleene closure in the grammar) and that *null* values could be present.

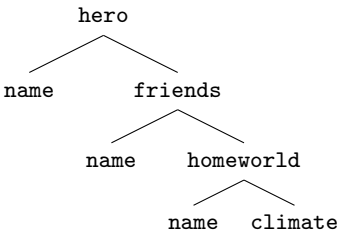


Figure 8 The tree corresponding to the query in Figure 4.

- $\Sigma = \{\texttt{id}, \texttt{name}, \texttt{friends}, \texttt{homeworld}, \texttt{species}, \texttt{climate}, \texttt{lifespan}, \texttt{origin}, \texttt{String}, \texttt{data}, \texttt{ID}, \mathit{null}\}$
- $R = D$
- $N = \{D, Q, C, P, S\}$
- Δ contains the following productions (for each $x_i[\alpha_i]\beta_i$ in the body there is a "?" because the clause is optional, so for clarity we will omit the "?").

  $D \rightarrow \texttt{data}[Q]$
  $Q \rightarrow [\ C \mid S* \mid P*\ ]$
  $C \rightarrow \texttt{id}[\texttt{ID}]\ \ \texttt{name}[\ \texttt{String} \mid \mathit{null}\ ]\ \ \texttt{friends}[C]*\ \ \texttt{homeworld}[P]\ \ \texttt{species}[S]$
  $P \rightarrow \texttt{id}[\texttt{ID}]\ \ \texttt{name}[\ \texttt{String} \mid \mathit{null}\ ]\ \ \texttt{climate}[\ \texttt{String} \mid \mathit{null}\ ]$
  $S \rightarrow \texttt{id}[\texttt{ID}]\ \ \texttt{name}[\ \texttt{String} \mid \mathit{null}\ ]\ \ \texttt{lifespan}[\ \texttt{String} \mid \mathit{null}\ ]\ \ \texttt{origin}[P]$

The result grammar is used to generate the result shown in Figure 5.

4.2 Temporal Tree Grammar

To support temporal data, we introduce a *temporal tree grammar*. There are (at least) two ways in which a grammar can be temporal.

1. The grammar itself evolves - The grammar changes over time as rules are updated, inserted, and deleted. For Temporal GraphQL this is akin to schema evolution or versioning [32].
2. The data changes over time - The grammar describes a snapshot of the data, but the data itself is temporal, capturing the entire timeline of its evolution (in some time dimension(s)). The non-temporal grammar parses each snapshot rather than the entire history of the data.

We consider evolution of the grammar next, and in Section 4.2.2 representing the data's history.

4.2.1 Grammar Evolution

To record the changing history of the schema, each terminal and nonterminal in the body of a production is annotated with a transaction time lifetime (transaction time since edits to the schema are transactions). The lifetimes are updated as follows when edits are made to the productions (to the schema).

- Deletion of terminal or nonterminal in the body - When part of a production is deleted the lifetime of the deleted parts is ended at the time at which it was deleted.
- Insertion of a terminal or nonterminal - An insertion creates a new lifetime starting at the time at which the part of the production was inserted and terminating at time *uc* (until changed).

- Insertion or deletion of a production - Each terminal or nonterminal in the body is updated as either inserted (lifetime starts) or deleted (lifetime ends).
- Terminal or nonterminal marked as *deprecated* - GraphQL supports a `@deprecated` annotation in the schema. Rather than deleting properties or types, they can be annotated as deprecated. Since deprecated properties/types are not deleted, their lifetime will continue (since they contine to be present in the schema). Since deprecated properties and types are part of the GraphQL standard, Temporal GraphQL will continue to support such types, so a slice at the current time will include deprecated fields present as of the time of the slice. However, in Temporal GraphQL fields and properties (including deprecated fields) can be (logically) deleted since rollback to previous versions of the schema is supported. So in some sense, Temporal GraphQL "fixes" the need for having a `@deprecated` annotation, but for compatibility, it continues to support the annotation.
- Change to a production - We model the change to a production as the deletion of the changed part, followed by an insertion of the new part.

For example, suppose that at time 1 the temporal type given in Figure 6 is inserted in the schema. Then at time 2 the `name` field is deleted and at time 3 a `moniker` field is added. By time 4 the production for the type in the query grammar would be as shown below, the temporal annotations are shown as subscripts for each terminal and nonterminal in the body of the production.

$$C \rightarrow \texttt{id}_{1-uc}[\ ]\ \texttt{name}_{1-2}[\ ]\ \texttt{moniker}_{3-uc}[\ ]\ \texttt{friends}_{1-uc}[C]\ \texttt{homeworld}_{1-uc}[P]\ \texttt{species}_{1-uc}[S]$$

The temporal annotations specify the time(s) at which the body of the production is valid. For instance the rule would be used as follows in validating a query, such as that given in Figure 4. Initially, the lifetime of the query is 1-uc. But the query contains the `Character name` property. So as the query `name` construct is parsed, the lifetime becomes 1-$uc \bigcap$ 1-2 = 1-2, which is the time interval in which the `name` property existed. If the lifetime becomes empty, then the parsing fails (at no transaction time is the query valid).

The lifetime computed by the query grammar can be propagated to the task of generating the result using the result grammar, in particular it constrains the result to only data that was alive during the computed lifetime as described next

4.2.2 A Representational Model for Data Evolution

A result tree grammar can be converted to a temporal result tree grammar that includes timestamps in the data and supports multiple versions of a property as follows. First, for each nonterminal, X, in the grammar corresponding to an `@temporal` object type we add add two nonterminals: a *version* nonterminal, represented as X_V, which represents a single version of the data, and a *history* nonterminal, represented as X_T, which is a list of versions. Next, we add a production, $X_T \rightarrow \texttt{versions}X[X_V*]$, to indicate that a history nonterminal is a list of versions of type X. We also add a production, $X_V \rightarrow \texttt{timestamp}[T]\ \texttt{data}[X]$, to state that a version is the paring of a timestamp (represented by `Timestamp` type, T) and the data for that version, which is an instance of X. Finally, in the body of each production we replace X with X_T to indicate that X is temporal.

As an example, the productions in the temporal result tree grammar is given below.

- $\Sigma = \{\texttt{id}, \texttt{name}, \texttt{friends}, \ldots \texttt{lifespan}, \texttt{origin}, \texttt{String}, \texttt{data}, \texttt{ID}, \mathit{null}\}$
- $R = Q$
- $N = \{Q, C, C_T, C_V, P, P_T, P_V, S, S_T, S_V\}$
- Δ contains the following productions.

$D \rightarrow$ `data`$[Q]$
$Q \rightarrow$ [C_T | S_T* | P_T*]
$C_T \rightarrow$ `versionsCharacter`$[C_V*]$
$C_V \rightarrow$ `timestamp`$[T]$ `data`$[C]$
$C \rightarrow$ `id`[`ID`] `name`[`String` | *null*] `friends`$[C]*$ `homeworld`$[P]$ `species`$[S]$
$P_T \rightarrow$ `versionsPlanet`$[P_V*]$
$P_V \rightarrow$ `timestamp`$[T]$ `data`$[P]$
$P \rightarrow$ `id`[`ID`] `name`[`String` | *null*] `climate`[`String` | *null*]
$S_T \rightarrow$ `versionsSpecies`$[S_V*]$
$S_V \rightarrow$ `timestamp`$[T]$ `data`$[S]$
$S \rightarrow$ `id`[`ID`] `name`[`String` | *null*] `lifespan`[`String` | *null*] `origin`$[P]$
$T \rightarrow$ `start`[`Int` | *null*] `stop`[`Int` | *null*]

A key feature of the result tree grammar is that it can be expressed with a non-temporal GraphQL schema, enabling GraphQL itself to support Temporal GrapQL. The *representational schema* for the temporal result grammar is given n Figure 9. The representational schema is a GraphQL schema that represents the data and temporal metadata. In the representational schema the `Character` type is converted to a `CharacterTemporal` type. The `CharacterTemporal` type replaces `Character` everywhere else in the schema. A temporal type is a list of *versions*. Each version is a `Timestamp` paired with a snapshot of a non-temporal `Character` object. The `Timestamp` object is simply a transaction time interval, but in practice could be bitemporal, a temporal element, or indeterminate; the object is defined to represent the kinds and nature of times that annotate the data.

The temporal result grammar describes the types produced by the API. A fragment of the representational data is shown in Figure 10. The data represents one change to the "`R2-D2`" `Character`, the `homeworld` was updated from "`Tatooine`" to "`Naboo`". The change creates a new version of the `Character` object.

We note that this representational grammar takes a *temporal-centric* approach to querying (described in the next section). In a temporal centric approach filtering is done primarily by temporal constraints rather than value-specific constraints. There are alternative, potential representations that we leave to future work that could better support a *value-centric* approach. For instance, a `Timestamp` property could be added to each type in the schema to record its lifetime. We envision a system in which a designer could choose the representational schema that best suits their needs or alternatively choose support for more than one kind of representational schema simultaneously. Note that the schema is not used for storage of the data, rather it provides a view for query access, so supporting alternative representations should be possible.

4.2.3 Evaluating Temporal Queries

The temporal result grammar is also used to construct the result of temporal queries. Just as in the non-temporal case, the grammar is used to *generate* a result by repeated application of the productions starting from the start nonterminal. But in the temporal case two additional features are needed, the timestamps have to be processed and the result can be temporal or non-temporal (depending on the operation). Let's consider the `@slice` operator first.

We introduce the notion of *sequenced generation* (or parsing) to process `@slice`. It is sequenced because for each use of a production in the generation (parse) there is an associated *lifetime* that represents when the data is alive. The lifetime is important to track since in the generated tree a child cannot exist without its parent, so each child's lifetime is constrained

```
...

type CharacterTemporal {
  versionsCharacter: [CharacterVersion]
}

type CharacterVersion {
  timestamp: Timestamp
  snapshot: Character
}

type Character {
  name: String
  friends: [CharacterTemporal]
  homeworld: PlanetTemporal
  species: SpeciesTemporal
}

...

type Timestamp {
  start: String
  stop: String
}
```

Figure 9 A representational GraphQL schema for the temporal schema of Figure 1.

```
...
CharacterTemporal = [
  { "id": "id_r2_d2",
    "versionsCharacter": ["id_r2_d2_v1",
                          "id_r2_d2_v2"] },
  ...
]
CharacterVersion = [
  { "id": "id_r2_d2_v1",
    "timestamp": "id_t1",
    "snapshot": "id_r2_d2_s1 },
  { "id": "id_r2_d2_v2",
    "timestamp": "id_t2",
    "snapshot": "id_r2_d2_s2 },
]
Character = [
  { "id": "id_r2_d2_s1",
    "name": "R2-D2",
    "friends": [
      "characterId": "id_luke_skywalker_v1",
      "characterId": "id_han_solo_v1"  ],
    "homeworldId": "id_tatooine_v1",
    "species": "id_droid_v1" },
  { "id": "id_r2_d2_s2",
    "name": "R2-D2",
    "friends": [
      "characterId": "id_luke_skywalker_v1",
      "characterId": "id_han_solo_v1" ],
    "homeworldId": "id_naboo_v1",
    "species": "id_droid_v1" },
   ...
]
...
```

Figure 10 JSON that conforms to the GraphQL schema in Figure 1 for a science fiction database.

by the lifetime of its parent. A slice grabs the part of a history within a time interval specified by the user. So initially, the lifetime of the root is given in the slice. Moreover, when the `@slice` is validated using an evolving grammar, the validation produces a timestamp that represents the times at which the query is valid (matches the temporal schema grammar). So the lifetime is the intersection of the time specified in the slice and the time produced by the validation. If this time is empty, then the `@slice` generates an error (the schema does not match the query). As the generation proceeds from the root to each leaf in the result, the lifetime is maintained along each branch in the tree by taking the intersection of the branch's lifetime with the data's lifetime in any version object.

As an example, consider the `@slice` query given in Figure 7 using the grammar of Section 4.2 including the change made to the grammar in Section 4.2.1 on the data of Figure 10. The query specifies a slice from 3-4 but validation produces a time of 1-2 since `name` only exists at time 1-2. The intersection is empty, hence an error would be generated. But suppose the slice was from 1-1. The result depicted in Figure 11 would be generated. Initially 1-1 would be passed from the root (the `hero` node) along each branch

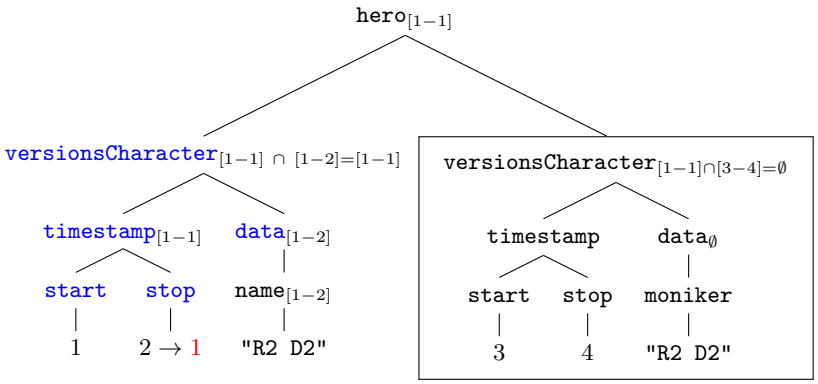


Figure 11 The tree corresponding to the query in Figure 4.

(each `Character` version) in the constructed tree. For each version in the data the intersection of the version's time and the branch's time is computed, which becomes the branch's time for descendants along the branch, and the timestamp is updated. If the time is empty then the branch generation is terminated. As an example, for the first (leftmost) `versionsCharacter` node in Figure 11 the `stop` property in the timestamp is changed from 2 to 1 as indicated in a red font. For the second `versionsCharacter` branch the intersection of 1-1 and 3-4 (the time of the version) is empty, so the framed branch in the tree is not generated in the result.

The evaluation of `@delta` starts the same as the evaluation of `@slice` with the initial computation of the lifetime of the root. But the intersection of times along a branch is not performed, instead branches are pruned if the version lifetime does not start or end during the branch's lifetime. So if we replace `@slice` with `@delta` in the query in Figure 7 and use a `start` time of 1, then the validation would produce a root timestamp of times 1-2. When applied to Figure 11 the leftmost branch (without changing the timestamp) would be selected, but the right child has a time of 3-4, so is outside the interval 1-2 and would not be included in the result.

Finally, the evaluation of `@snapshot` (and `@current`) is similar to that of `@slice` insofar as branches are pruned that fall outside of the slice time. But all non-data nodes are removed from the result. As example using Figure 11 the `@snapshot` at time 1 would prune the framed subtree rooted at the rightmost `versionsCharacter` node (since its lifetime is 3-4), and also prune from the result the non-data nodes by placing the `name` node as the only child of the `hero` node (removing the nodes in blue font in the leftmost `versionsCharacter` subtree).

5 Related Work

To the best of our knowledge there have been no previous papers on Temporal GraphQL or temporal tree grammars. There has, however, been extensive previous research to supporting temporal data [3, 20, 30]. This research has fallen into two broad categories: versioning and timestamp-based support. Timestamp-based queries are common in temporal relational databases. A temporal relational database [25] stores data that is annotated with time metadata. The time metadata records when the data was alive in some time domain, e.g., transaction time [27], valid time [28], or both. Such databases can be queried in various ways. For instance in TSQL2 [34] a query can be evaluated to retrieve the data's history e.g., a timeslice query [26], or retrieve the data as of some time instant, e.g., a snapshot

query [35], or perform a query at every time instant in the data's history, e.g., a sequenced query [5]. But TSQL2 does not support queries that ask for versions of data, e.g., get the second version of an employment record or retrieve the changes to the employment record. Data versioning is more common in temporal object-oriented databases [13] or temporal documents where each edit or change creates a new version of an object or document [21]. Users can navigate among the versions and restore old versions if necessary.

Semi-structured data representations such as JSON, XML, and YAML are used to represent both data and documents and thus need to support both timestamp and version histories [1, 7–11, 16, 31, 37]. Semi-structured data changes over time, sometimes frequently, as new data is inserted and existing data is edited and deleted [12, 24, 29]. Previous research in temporal XML and JSON called elements that maintain their identity over time *items* [14, 17, 18, 36]. Items are timestamped with a lifetime and as an element can be moved within a document. Each change to an item creates a version, which is also timestamped. Previous research showed how to represent, query, describe with a schema and validate temporal semi-structured data. Differences in XML and JSON spawned further research in schema validation and versioning for JSON data [6].

6 Conclusions

GraphQL is a widely used technology for making it easier to develop and maintain web applications. GraphQL queries and mutations are used to read and write data to a back-end database through a web services API. But data and schemas change over time and capturing and querying this history is important in many applications. In this paper we presented Temporal GraphQL, a technology that adds support for time to GraphQL. We observed that tree grammars can model GraphQL schemas, data, and queries and we proposed temporal tree grammars to model temporal GraphQL. And we extended GraphQL with temporal snapshot, slice, and delta operators. The key advantage of our design is that it leverages non-temporal GraphQL to support GraphQL.

Our short-term future work is targeted to implementing Temporal GraphQL as a layer for a GraphQL system. But such a system has to be coupled with techniques for converting web services to temporal web services, to supply the data for the temporal types in the query, which in term needs temporal support in a back-end database. We are currently working on a PostGraphile layer; PostGraphile is a GraphQL system for Postgres databases. We also plan to specialize the `@temporal` annotation to support different kinds of time, e.g., `@validTime`. And we plan to add temporal elements and support for indeterminacy to the `Timestamp` type. Indeterminacy will also require some changes to queries to support indeterminate queries. Finally, we plan to generalize the support for temporal metadata outlined in this paper to include other kinds of metadata, such as quality metadata.

References

References

1 Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. A Data Model for Temporal XML Documents. In *Database and Expert Systems Applications, 11th International Conference, DEXA 2000, London, UK, September 4-8, 2000, Proceedings*, pages 334–344, 2000. doi:10.1007/3-540-44469-6_31.

2 Luca Anselma, Luca Piovesan, and Paolo Terenziani. Dealing with temporal indeterminacy in relational databases: An AI methodology. *AI Commun.*, 32(3):207–221, 2019. doi:10.3233/AIC-190619.

3 Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Ontology-mediated query answering over temporal data: A survey (invited talk). In *24th International Symposium on Temporal Representation and Reasoning, TIME 2017, October 16-18, 2017, Mons, Belgium*, pages 1:1–1:37, 2017. doi:10.4230/LIPIcs.TIME.2017.1.

4 Michael Benedikt, Leonid Libkin, and Frank Neven. Logical definability and query languages over ranked and unranked trees. *ACM Trans. Comput. Logic*, 8(2):11–es, April 2007. doi:10.1145/1227839.1227843.

5 Michael H. Böhlen and Christian S. Jensen. Sequenced semantics. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_1053.

6 Safa Brahmia, Zouhaier Brahmia, Fabio Grandi, and Rafik Bouaziz. τjschema: A framework for managing temporal json-based nosql databases. In *Database and Expert Systems Applications - 27th International Conference, DEXA 2016, Porto, Portugal, September 5-8, 2016, Proceedings, Part II*, pages 167–181, 2016. doi:10.1007/978-3-319-44406-2_13.

7 Zouhaier Brahmia, Fabio Grandi, Safa Brahmia, and Rafik Bouaziz. A graphical conceptual model for conventional and time-varying json data. *Procedia Computer Science*, 184:823–828, 2021. The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops. doi:10.1016/j.procs.2021.03.102.

8 Zouhaier Brahmia, Fabio Grandi, Safa Brahmia, and Rafik Bouaziz. τjupdate: An update language for time-varying JSON data. *J. Comput. Lang.*, 79:101258, 2024. doi:10.1016/J.COLA.2024.101258.

9 Zouhaier Brahmia, Hind Hamrouni, and Rafik Bouaziz. XML data manipulation in conventional and temporal XML databases: A survey. *Comput. Sci. Rev.*, 36:100231, 2020. doi:10.1016/J.COSREV.2020.100231.

10 Sudarshan S. Chawathe, Serge Abiteboul, and Jennifer Widom. Representing and Querying Changes in Semistructured Data. In *Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23-27, 1998*, pages 4–13, 1998. doi:10.1109/ICDE.1998.655752.

11 Shu-Yao Chien, Vassilis J. Tsotras, and Carlo Zaniolo. Efficient Schemes for Managing Multiversion XML Documents. *VLDB J.*, 11(4):332–353, 2002. doi:10.1007/s00778-002-0079-4.

12 Junghoo Cho and Hector Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler. In *VLDB*, pages 200–209, 2000. URL: http://www.vldb.org/conf/2000/P200.pdf.

13 Carlo Combi. Temporal object-oriented databases. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_404.

14 Faiz Currim, Sabah Currim, Curtis E. Dyreson, Richard T. Snodgrass, Stephen W. Thomas, and Rui Zhang. Adding Temporal Constraints to XML Schema. *IEEE Trans. Knowl. Data Eng.*, 24(8):1361–1377, 2012. doi:10.1109/TKDE.2011.74.

15 DB-Engines Ranking. https://db-engines.com/en/ranking. Accessed: 2025-05-10.

16 Curtis E. Dyreson and Fabio Grandi. Temporal XML. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_411.

17 Curtis E. Dyreson and Kalyan G. Mekala. Prefix-Based Node Numbering for Temporal XML. In *Web Information System Engineering - WISE 2011 - 12th International Conference, Sydney, Australia, October 13-14, 2011. Proceedings*, pages 172–184, 2011. doi:10.1007/978-3-642-24434-6_13.

18 Curtis E. Dyreson, Amani M. Shatnawi, Sourav S. Bhowmick, and Vishal Sharma. Temporal JSON keyword search. *Proc. ACM Manag. Data*, 2(3):177, 2024. doi:10.1145/3654980.

19 Curtis E. Dyreson and Richard Thomas Snodgrass. Supporting valid-time indeterminacy. *ACM Trans. Database Syst.*, 23(1):1–57, March 1998. doi:10.1145/288086.288087.

20 Opher Etzion, Sushil Jajodia, and Suryanarayana M. Sripada, editors. *Temporal Databases: Research and Practice. (the book grow out of a Dagstuhl Seminar, June 23-27, 1997)*, volume 1399 of *Lecture Notes in Computer Science*. Springer, 1998. doi:10.1007/BFb0053695.

21 Aayush Goyal and Curtis E. Dyreson. Temporal JSON. In *5th IEEE International Conference on Collaboration and Internet Computing, CIC 2019, Los Angeles, CA, USA, December 12-14, 2019*, pages 135–144. IEEE, 2019. doi:10.1109/CIC48465.2019.00025.

22 GraphQL: A Query Language for Your API. https://graphql.org. Accessed: 2025-05-10.

23 Introduction to GraphQL. https://graphql.org/learn. Accessed: 2025-05-10.

24 Panagiotis G. Ipeirotis, Alexandros Ntoulas, Junghoo Cho, and Luis Gravano. Modeling and Managing Content Changes in Text Databases. In *ICDE*, pages 606–617, 2005. doi:10.1109/ICDE.2005.91.

25 Christian S. Jensen and Richard T. Snodgrass. Temporal database. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_395.

26 Christian S. Jensen and Richard T. Snodgrass. Timeslice operator. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_1426.

27 Christian S. Jensen and Richard T. Snodgrass. Transaction time. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_1064.

28 Christian S. Jensen and Richard T. Snodgrass. Valid time. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_1066.

29 Venkata N. Padmanabhan and Lili Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *SIGCOMM*, pages 111–123, 2000. doi:10.1145/347059.347413.

30 Vangipuram Radhakrishna, P. V. Kumar, and V. Janaki. A survey on temporal databases and data mining. In *Proceedings of the The International Conference on Engineering & MIS 2015*, ICEMIS '15, pages 52:1–52:6, New York, NY, USA, 2015. ACM. doi:10.1145/2832987.2833064.

31 Flavio Rizzolo and Alejandro A. Vaisman. Temporal XML: Modeling, Indexing, and Query Processing. *VLDB J.*, 17(5):1179–1212, 2008. doi:10.1007/s00778-007-0058-x.

32 John F. Roddick. A survey of schema versioning issues for database systems. *Inf. Softw. Technol.*, 37(7):383–393, 1995. doi:10.1016/0950-5849(95)91494-K.

33 William C. Rounds. Context-free grammars on trees. In *Proceedings of the 1st Annual ACM Symposium on Theory of Computing, May 5-7, 1969, Marina del Rey, CA, USA*, pages 143–148. ACM, 1969. doi:10.1145/800169.805428.

34 Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

35 Richard T. Snodgrass, Michael H. Böhlen, Christian S. Jensen, and Andreas Steiner. Transitioning temporal support in TSQL2 to SQL3. In *Temporal Databases: Research and Practice. (the book grow out of a Dagstuhl Seminar, June 23-27, 1997)*, pages 150–194, 1997. doi:10.1007/BFb0053702.

36 Richard T. Snodgrass, Curtis E. Dyreson, Faiz Currim, Sabah Currim, and Shailesh Joshi. Validating Quicksand: Temporal Schema Versioning in tauXSchema. *Data Knowl. Eng.*, 65(2):223–242, 2008. doi:10.1016/j.datak.2007.09.003.

37 Fusheng Wang and Carlo Zaniolo. An XML-Based Approach to Publishing and Querying the History of Databases. *World Wide Web*, 8(3):233–259, 2005. doi:10.1007/s11280-005-1317-7.

Safety and Liveness on Finite Words

Luca Geatti
University of Udine, Italy

Stefano Pessotto
University of Udine, Italy

Stefano Tonetta
Fondazione Bruno Kessler, Italy

Abstract

The study of safety and liveness is crucial in the context of formal languages on infinite words, providing a fundamental classification of system properties. They have been studied extensively as fragments for regular languages and Linear Temporal Logic (LTL), both from the theoretical and practical point of view, especially in the context of model checking. In contrast, despite the growing interest in Linear Temporal Logic over finite traces (LTLf) as a specification formalism for finite-length executions, the notions of safety and liveness for finite words have remained largely unexplored.

In this work, we address this gap by defining the safety and liveness fragments of languages on finite words, mirroring the definition used for infinite words. We show that safety languages are exactly those that are prefix-closed, from which a bounded model property for all safety languages follows. We also provide criteria for determining whether a given language belongs to the safety or liveness fragment and analyze the computational complexity of this classification problems. Moreover, we show that certain LTL formulas classified as safety or liveness over infinite words may not preserve this classification when interpreted over finite words, and *vice versa*. We further establish that the safety-liveness decomposition theorem – asserting that every ω-regular language can be expressed as the intersection of a safety language and a liveness language – also holds in the finite-word setting. Finally, we examine the implications of these results for the model checking problem in LTLf.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Safety, Liveness, Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.10

Related Version *Extended Version*: `https://users.dimi.uniud.it/~luca.geatti/data/time-25/time25.pdf` [14]

Funding Luca Geatti acknowledges the support from the project "ENTAIL – intEgrazioNe tra runTime verification e mAchlne Learning" – funded by the European Union – NextGenerationEU, under the PNRR- M4C2I1.5, Program "iNEST – interconnected nord-est innovation ecosystem" – Creazione e rafforzamento di "Ecosistemi dell'Innovazione per la sostenibilità" – ECS_00000043, CUP G23C22001130006 – R.S. Geatti.

1 Introduction

The concepts of *safety* and *liveness* form a fundamental classification of system properties that describe how systems behave over time. The distinction was first introduced by Leslie Lamport in his 1977 paper on proving the correctness of concurrent programs [20], where he informally characterized safety properties as those stipulating that *"nothing bad happens"*, and liveness properties as those ensuring that *"something good eventually happens"*. The formalization of these concepts was provided by Alpern and Schneider in [1], by defining safety and liveness properties as ω-regular languages (*i.e.* sets of infinite words) such that:

- safety: all violations of the property are irremediable, that is, there exists a finite prefix of each violation (bad prefix) such that all its extensions violate the property;
- liveness: there are no irremediable violations, that is, for any finite prefix, there exists a continuation that satisfies the property.

They also proved the *safety-liveness decomposition theorem* for ω-regular languages: every property on infinite words can be expressed as the intersection of a safety property and a liveness property. This result has been recently extended also to quantitative properties [16].

The classification of properties into safety and liveness has become fundamental in the formal verification of reactive systems, particularly within the context of *model checking* for Linear Temporal Logic (LTL [23]) properties, that is, the problem of checking whether all executions of a system satisfy an LTL formula. This distinction enables the application of specialized verification techniques: once a property is identified as safety or liveness, efficient algorithms tailored to each class – such as the early proof systems by Manna and Pnueli [21, 19] or `IC3` for safety properties [7, 8] and `K-Liveness` for liveness properties [11] – can be employed. Furthermore, the identification of the safety fragment of LTL has led to syntactic characterizations that capture exactly the safety properties expressible in LTL [9, 24, 10], thereby allowing one to express safety properties directly, without the need to verify whether a given formula satisfies the safety condition.

In the last decade, Linear Temporal Logic on finite words (LTLf [13]) has emerged as a useful formalism for reasoning about systems with inherently finite executions (for example, in the context of planning). LTLf preserves the syntax of standard LTL but interprets formulae over finite words. Despite the growing interest in LTLf, the adaptation of safety and liveness to finite words has received little attention. In [10], a logical characterization of the safety fragment of LTLf has been proved complete. However, most research questions remain still open.

This work addresses this gap by studying safety and liveness for languages over finite words, revisiting classical results from the infinite-word setting – such as the decomposition theorem – and establishing new results specific to the finite-word case. Our main contributions are as follows.

First, we examine key properties of the safety fragment over finite words. We show that safety languages in this setting are precisely the *prefix-closed* languages; that is, if a word belongs to the language, then all of its prefixes must also belong to it. This characterization allows us to establish a bounded model property: a safety language is non-empty if and only if it contains the empty word. Furthermore, we demonstrate that, analogous to the infinite-word setting, a regular language over finite words is a safety language if and only if its *closure* – defined as the automaton obtained by marking all states as final – is equivalent to the original automaton. We then study the complexity of deciding whether a regular language is a safety language. We prove that this problem is PSPACE-complete, both when the language is given by a Nondeterministic Finite Automaton (NFA) and when it is specified by an LTLf formula – matching the known complexity for the infinite-word setting [24].

Second, we analyze the liveness fragment of languages over finite words. We prove that, similarly to the case of ω-regular languages, a regular language is liveness if and only if its closure recognizes the universal language. This characterization allows us to derive complexity results for the liveness recognition problem: it is PSPACE-complete when the language is specified by an NFA, and in EXPSPACE when given by an LTLf formula. Moreover, we highlight a subtle but important distinction that arises when transitioning between infinite-word and finite-word semantics in temporal logic. Specifically, we show that certain formulae classified as safety (resp., liveness) under the infinite-word interpretation (LTL) are not safety (resp., liveness) under the finite-word interpretation (LTLf), and *vice versa*.

Third, we prove that every regular language can be expressed as the intersection of a safety language and a liveness language, extending the classical Alpern-Schneider decomposition to the finite-word setting.

Last but not least, we investigate the model checking of LTLf properties over safety systems – namely, systems represented by NFAs recognizing safety properties. We demonstrate that, within this context, model checking requires careful consideration. In numerous instances, formulae that are semantically meaningful and nontrivial in the infinite-word setting lead to degenerate cases in the finite-word framework. In such cases, we show that the model checking problem becomes trivial – either invariably false or reducible to a simple condition, such as verifying whether the empty word ϵ is accepted by the property, or to the model checking of a substantially simpler formula.

Related Work

In [18], Kupferman and Vardi were the first to show that the problem of determining whether an LTL formula defines a liveness language is EXPSPACE-complete. The complexity of this problem – open for over three decades – highlights that recognizing liveness is substantially more difficult when starting from LTL formulae than from NFAs. This challenge is further exacerbated by the absence of syntactic characterizations for liveness properties, in contrast to the case of safety.

In [4], Basin *et al.* address the problem of deciding whether a formula of Timed Linear Temporal Logic (TLTL) expresses a safety or a liveness property. They prove that the problem is EXPSPACE-complete for safety and 2EXPSPACE-complete for liveness, thereby effectively adding one exponential of complexity compared to the untimed LTL case.

In [3], the model checking problem for LTLf formulae is examined. It is shown that, when restricting the model checking problem to traces that both start from an initial state and end in a final state, the problem is PSPACE-complete – matching the complexity of standard LTL model checking. Conversely, when considering all traces originating from an initial state – regardless of whether they reach a final state – the problem becomes EXPSPACE-complete.

The work presented in [22] refines the safety-liveness classification of LTL properties by considering their monitorability. It focuses on runtime verification, thus considering finite words (sequence of observations) as prefixes of infinite executions. For this reason, the definitions of safety and liveness are the classical ones based on prefixes of infinite words. In contrast, our work directly defines and explores safety and liveness fragments for languages on finite words, and specifically for LTLf. This distinction is crucial as it leads to many different properties compared to the infinite word setting.

Outline of the paper

Section 2 reviews the necessary background on LTL, LTLf, and automata. Sections 3 and 4 investigate the properties and computational complexity of safety and liveness languages over finite words, respectively. Section 5 establishes the safety-liveness decomposition theorem in the finite-word setting. Section 6 examines the implications of our results for the model checking of LTLf specifications. Section 7 summarizes the achieved results and outlines directions for future work. Proofs omitted from the main text are provided in Appendix A or in the extended version of this paper [14].

2 Background

In this section, we give the necessary background.

From now on, let $\Sigma = \{a, b, c, \ldots\}$ be an alphabet, *i.e.* a finite set of symbols. A *finite word* (over Σ) is any finite, possibly empty, sequence of symbols in Σ. An *infinite word* (over Σ) is any infinite sequence of symbols in Σ. We denote with Σ^* (resp., Σ^ω) the set of all finite and possibly empty (resp., infinite) words over Σ. We denote with ε the empty word. We define the *length* of σ as $|\sigma| = 0$ if $\sigma = \varepsilon$, as $|\sigma| = n$ if $\sigma = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle \in \Sigma^*$, and as $|\sigma| = \omega$ if $\sigma \in \Sigma^\omega$. A *language of finite words* $\mathcal{L}$ is a subset of Σ^*, while a *language of infinite words* $\mathcal{L}$ is a subset of Σ^ω. We denote with $\overline{\mathcal{L}}$ the complement of $\mathcal{L}$.

2.1 Linear Temporal Logic

We start by giving the syntax of *Linear Temporal Logic on finite words* (LTLf [13]) and of *Linear Temporal Logic* (LTL [23]), both of which are defined by the same grammar. From now on, let $\mathcal{AP} = \{p, q, r, \ldots\}$ be a set of atomic propositions.

▶ **Definition 1.** *A formula ϕ of* LTLf *and of* LTL *over $\mathcal{AP}$ is inductively defined as follows:*

$$\phi := \top \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{X}\phi \mid \phi_1 \,\mathsf{U}\, \phi_2$$

where $p \in \mathcal{AP}$.

The temporal operators X and U are called respectively *next* and *until*. We define the following classic shortcut operators: (i) $\bot := \neg\top$; (ii) $\phi_1 \wedge \phi_2 := \neg(\neg\phi_1 \vee \neg\phi_2)$; (iii) $\widetilde{\mathsf{X}}\phi := \neg\mathsf{X}\neg\phi_1$; (iv) $\mathsf{F}\phi := \top \,\mathsf{U}\, \phi$; (v) $\mathsf{G}\phi := \neg\mathsf{F}\neg\phi$; (vi) $\phi_1 \,\mathsf{R}\, \phi_2 := \neg((\neg\phi_1) \,\mathsf{U}\, (\neg\phi_2))$. The temporal operators $\widetilde{\mathsf{X}}$, F, G, and R are called respectively *weak next*, *eventually*, *globally*, and *release*. The *size* of ϕ is the size of its parse tree.

A notable fragment of LTLf and of LTL is SafetyLTL. Formulae in this fragment restrict the use of temporal operators to $\widetilde{\mathsf{X}}$, G and R, and allow negation only in front of atomic propositions. The syntax of SafetyLTL is presented below.

▶ **Definition 2.** *A formula ϕ of* SafetyLTL *over $\mathcal{AP}$ is inductively defined as follows:*

$$\phi := \top \mid \bot \mid p \mid \neg p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \widetilde{\mathsf{X}}\phi \mid \mathsf{G}\phi \mid \phi_1 \,\mathsf{R}\, \phi_2$$

where $p \in \mathcal{AP}$.

We now define the semantics of LTLf and LTL. Formulae of LTLf (resp., of LTL) are interpreted over finite (resp., infinite) words. More precisely, LTLf is interpreted over finite (possibly empty) words in $(2^{\mathcal{AP}})^*$, while LTL is interpreted over infinite words in $(2^{\mathcal{AP}})^\omega$. The satisfaction of a formula ϕ of LTLf (resp., of LTL) by a finite word (resp., by an infinite word) $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ at position i, denoted by $\sigma, i \models \phi$, is inductively defined as follows:

- $\sigma, i \models \top$ is always true;
- $\sigma, i \models p$ iff $0 \leq i < |\sigma|$ and $p \in \sigma_i$;
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$;
- $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;
- $\sigma, i \models \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models \phi$;
- $\sigma, i \models \phi_1 \,\mathsf{U}\, \phi_2$ iff $\exists j \,.\, i \leq j < |\sigma| (\sigma, j \models \phi_2 \wedge \forall k \,.\, i \leq k < j (\sigma, k \models \phi_1))$.

We write $\sigma \models \phi$ to denote $\sigma, 0 \models \phi$. The language of an LTLf formula ϕ over the set of atomic propositions $\mathcal{AP}$, denoted as $\mathcal{L}(\phi)$, is defined as the set $\{\sigma \in (2^{\mathcal{AP}})^* \mid \sigma \models \phi\}$. Similarly, the language of an LTL formula ϕ, denoted with $\mathcal{L}^\infty(\phi)$, is defined as the set $\{\sigma \in (2^{\mathcal{AP}})^\omega \mid \sigma \models \phi\}$. We say that ϕ is valid on finite words (resp., on infinite words) if and only if $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$ (resp., $\mathcal{L}^\infty(\phi) = (2^{\mathcal{AP}})^\omega$).

We highlight an important aspect of the difference between LTLf and LTL, which will be relevant in the next section. Consider the formula $\widetilde{\mathsf{X}}\bot$. Under finite words semantics, we can use $\widetilde{\mathsf{X}}\bot$ to hook the final position of a word: for all finite words σ, it holds that $\sigma, i \models \widetilde{\mathsf{X}}\bot$ if and only if $i = |\sigma| - 1$. This allows us to express formulae like $\mathsf{G}(q)$ as $q \,\mathsf{U}\, (\widetilde{\mathsf{X}}\bot \wedge q)$, without the need of the *globally* operator. However, under infinite words semantics, the formula $\widetilde{\mathsf{X}}\bot$ is always false, and thus formulae like $q \,\mathsf{U}\, (\widetilde{\mathsf{X}}\bot \wedge q)$ are always false as well.

2.2 The safety and the liveness fragments of infinite words

Let Σ be an alphabet. Given an infinite word $\sigma \in \Sigma^\omega$, we define $\texttt{pref}(\sigma) \coloneqq \{\sigma' \in \Sigma^* \mid \exists \sigma'' \in \Sigma^\omega \text{ such that } \sigma = \sigma' \cdot \sigma''\}$.

The definition of safety language of infinite words is given as follows.

▶ **Definition 3.** *A language $\mathcal{L} \subseteq \Sigma^\omega$ is* safety *if and only if, for all $\sigma \notin \mathcal{L}$, there exists $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \notin \mathcal{L}$, for all $\sigma'' \in \Sigma^\omega$. Such prefix σ' is called a* bad prefix *for $\mathcal{L}$.*

Given a formula ϕ of LTL over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a safety formula iff $\mathcal{L}^\infty(\phi)$ is a safety language over the alphabet $2^{\mathcal{AP}}$.

In [9], it is proved that SafetyLTL, when interpreted on infinite words, captures exactly the set of all safety languages that are definable in LTL.

▶ **Proposition 4** ([9])**.** *Let $\mathcal{L} \subseteq \Sigma^\omega$ be a language definable in* LTL*. It holds that $\mathcal{L}$ is safety if and only if $\mathcal{L} = \mathcal{L}^\infty(\phi)$, for some formula $\phi \in$* SafetyLTL*.*

Liveness languages of infinite words are defined as follows.

▶ **Definition 5.** *A language $\mathcal{L} \subseteq \Sigma^\omega$ is* liveness *if and only if, for all $\sigma \in \Sigma^*$, there exists $\sigma' \in \Sigma^\omega$ such that $\sigma \cdot \sigma' \in \mathcal{L}$.*

Given a formula ϕ of LTL over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a liveness formula iff $\mathcal{L}^\infty(\phi)$ is a liveness language over the alphabet $2^{\mathcal{AP}}$.

2.3 Finite Automata

We define the classic notion of Nondeterministic Finite Automaton.

▶ **Definition 6.** *A* Nondeterministic Finite Automaton *(*NFA*) is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ such that: (i) Q is a finite set of states; (ii) Σ is a finite alphabet; (iii) $I \subseteq Q$ is the set of initial states; (iv) $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation; and (v) $F \subseteq Q$ is the set of final states.*

Given an NFA $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, a state $q \in Q$ and a finite word $\sigma = \langle \sigma_0, \dots, \sigma_{n-1} \rangle \in \Sigma^*$, we define $\hat{\Delta}(q, \sigma)$ as the set

$$\{q' \in Q \mid \exists \langle q_0, q_1, \dots, q_n \rangle \,.\, q_0 = q,\ q_n = q',\ (q_i, \sigma_i, q_{i+1}) \in \Delta,\ \forall 0 \leq i < n\}.$$

We say that $\mathcal{A}$ *reaches state* q' *reading* σ iff $q' \in \bigcup_{q_0 \in I} \hat{\Delta}(q_0, \sigma)$, for some $q_0 \in I$. A word $\sigma \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if $\bigcup_{q \in I} \hat{\Delta}(q, \sigma) \cap F \neq \varnothing$. The language of $\mathcal{A}$, denoted with $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. We say that two NFAs $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. If a language $\mathcal{L}$ is such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$, for some NFA $\mathcal{A}$, then $\mathcal{L}$ is called a *regular language*.

Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an NFA. $\mathcal{A}$ is called a *partial Deterministic Finite Automaton* (DFA) if $|I| \leq 1$ and, for all $q \in Q$ and for all $a \in \Sigma$, there exists at most one $q' \in Q'$ such that $(q, a, q') \in \Delta$.

The notion of reduced automata [2] is defined as follows.

▶ **Definition 7** (Reduced NFA). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an* NFA. *We say that $\mathcal{A}$ is* reduced *if, for all $q \in Q$, there exists $\sigma \in \Sigma^*$ such that $\hat{\Delta}(q, \sigma) \cap F \neq \varnothing$. We denote with $\mathcal{R}(\mathcal{A})$ the* NFA *obtained from $\mathcal{A}$ by removing all states q that do* not *satisfy the condition $\hat{\Delta}(q, \sigma) \cap F \neq \varnothing$ for any $\sigma \in \Sigma^*$.*

Clearly, it always holds that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{R}(\mathcal{A}))$. Notice that, if $\mathcal{L}(\mathcal{A}) = \varnothing$, then also all the initial states (along with all the states reachable from them) are removed from $\mathcal{R}(\mathcal{A})$, leading to a degenerate case of an automaton without initial state ($I = \varnothing$). Moreover, it is worth pointing out that every NFA can be transformed into an equivalent reduced DFA, first by applying determinization and then by removing all states that do not lead to any final state. Finally, we define the *closure* of an NFA [2] as follows.

▶ **Definition 8** (Closure automaton). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an* NFA. *The closure of $\mathcal{A}$, denoted with $\mathcal{C}(\mathcal{A})$, is the automaton obtained from $\mathcal{A}$ by setting all states as final,* i.e. *$(Q, \Sigma, I, \Delta, Q)$.*

The interesting property of closures in NFAs is that they reject words solely by attempting undefined transitions. This form of rejection is *irremediable*, as no subsequent extension of the input word can lead the automaton to accept it.

3 The safety fragment on finite words

In this section, we define the *safety* fragment of languages of finite words. We then study some properties of this fragment as well as some complexity-related issues. Finally we compare safety languages definable in LTLf with those definable in LTL.

3.1 Safety languages of finite words

From now on, let Σ be a finite alphabet. Given a finite word $\sigma \in \Sigma^*$, we define $\texttt{pref}(\sigma) \coloneqq \{\sigma' \in \Sigma^* \mid \exists \sigma'' \in \Sigma^* \,.\, \sigma = \sigma' \cdot \sigma''\}$. We define safety languages of finite words as follows.

▶ **Definition 9** (Safety languages of finite words). *A language $\mathcal{L} \subseteq \Sigma^*$ is* safety *if and only if, for all $\sigma \notin \mathcal{L}$, there exists $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \notin \mathcal{L}$, for all $\sigma'' \in \Sigma^*$. Such prefix σ' is called a* bad prefix *for $\mathcal{L}$.*

Given a formula ϕ of LTLf over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a safety formula if $\mathcal{L}(\phi)$ is a safety language over the alphabet $2^{\mathcal{AP}}$.

In contrast with safety languages of infinite words, in this setting we require that all the *finite* continuations of a bad prefix do not belong the language. From the definition, it immediately follows that every safety language $\mathcal{L}$ is such that $\overline{\mathcal{L}} = K \cdot \Sigma^*$, where $K \subseteq \Sigma^*$ is the set of bad prefixes.

Examples. Let $\Sigma = \{a, b\}$. The language b^* is a safety language, because every violation (*i.e.* every word not belonging to the language) contains a prefix ending with the symbol a such that all possible continuations of the prefix are violations. In this case, the set of bad prefixes is $b^* \cdot a \cdot \Sigma^*$. On the contrary, the language $b \cdot b^*$ is *not* of safety, since $\varepsilon \notin \mathcal{L}$ but it is not true that $\varepsilon \cdot \sigma' \notin \mathcal{L}$ for all $\sigma' \in \Sigma^*$. The LTLf formula $\mathsf{G}(p \rightarrow \widetilde{\mathsf{X}} q)$ recognizes a safety language, because any violating trace contains two adjacent positions where p is true in the first one and q is false in the next one. Any continuation of the trace starting from this point is a violation of the formula.

3.2 Properties of the safety fragment on finite words

First, we show that the safety condition is not limited to regular languages. In fact, there exist safety languages over finite words that are not regular.

▶ **Proposition 10.** *Let $\Sigma = \{a, b\}$ and let $\mathcal{L} = \{a^n \cdot b^n \mid n > 0\} \cdot \Sigma^*$. It holds that $\overline{\mathcal{L}}$ is safety and not regular.*

Proof. We provide the complete proof in [14]. ◀

Second, we show that safety languages over finite words are precisely those languages that are prefix-closed, *i.e.* if a word belongs to the language, then all of its prefixes are also included. The notion of prefix-closure is formally defined as follows.

▶ **Definition 11** (Prefix-closure). *Let $\mathcal{L} \subseteq \Sigma^*$. We say that $\mathcal{L}$ is* prefix-closed *if, for all $\sigma \in \mathcal{L}$, it holds that $\texttt{pref}(\sigma) \subseteq \mathcal{L}$.*

The following proposition proves that all safety languages of finite words are prefix-closed and *vice versa*. Therefore, prefix-closure provides an alternative and equivalent characterization of safety languages of finite words.

▶ **Proposition 12.** *Let $\mathcal{L} \subseteq \Sigma^*$. It holds that $\mathcal{L}$ is safety if and only if $\mathcal{L}$ is prefix-closed.*

Proof. We begin proving the left-to-right direction. Let $\mathcal{L} \subseteq \Sigma^*$ be a safety language. Suppose by contradiction that $\mathcal{L}$ is *not* prefix-closed, that is, there exists $\sigma \in \mathcal{L}$ and a prefix $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \notin \mathcal{L}$. Since $\mathcal{L}$ is safety, by Definition 9, it holds that there exists a prefix $\sigma'' \in \texttt{pref}(\sigma')$ of σ' such that $\sigma'' \cdot \sigma''' \notin \mathcal{L}$, for all $\sigma''' \in \Sigma^*$. In the particular case in which σ''' is the suffix of σ obtained from σ by removing its prefix σ'', we have that $\sigma'' \cdot \sigma''' = \sigma$ and $\sigma \notin \mathcal{L}$, which is a contradiction since we supposed that $\sigma \in \mathcal{L}$. Therefore, it must be that $\mathcal{L}$ is prefix-closed.

We now prove the right-to-left direction. Let $\mathcal{L} \subseteq \Sigma^*$ be a prefix-closed language. Suppose by contradiction that $\mathcal{L}$ is *not* safety, that is, there exists $\sigma \notin \mathcal{L}$ such that, for all $\sigma' \in \texttt{pref}(\sigma)$, there exists a $\sigma'' \in \Sigma^*$ such that $\sigma' \cdot \sigma'' \in \mathcal{L}$. In the particular case in which $\sigma' = \sigma$, we have that $\sigma \cdot \sigma'' \in \mathcal{L}$, for some $\sigma'' \in \Sigma^*$. But, since by hypothesis $\mathcal{L}$ is prefix-closed, all prefixes of $\sigma \cdot \sigma''$, and in particular σ, must belong to $\mathcal{L}$. However, this is a contradiction since we supposed that $\sigma \notin \mathcal{L}$. Therefore, $\mathcal{L}$ must be a safety language. ◀

As a corollary of Proposition 12, a bounded model property for safety languages over finite words follows directly, showing that any nonempty language of this kind necessarily includes the empty word (its proof is provided in Appendix A).

▶ **Corollary 13** (Small and Bounded Model Property for safety languages). *Let $\mathcal{L} \subseteq \Sigma^*$ be a safety language. It holds that $\mathcal{L} \neq \varnothing$ if and only if $\varepsilon \in \mathcal{L}$.*

Third, we show an effective way to establish whether the language recognized by a given NFA is safety. The procedure consists in checking whether the reduced version of the given NFA and its closure are equivalent.

▶ **Proposition 14.** *Let $\mathcal{A}$ be an* NFA. *$\mathcal{L}(\mathcal{A})$ is safety if and only if $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$.*

Proof. We consider first the right-to-left direction. Suppose that $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$ and consider the automaton $\mathcal{C}(\mathcal{R}(\mathcal{A}))$. Since the closure of any automaton, by definition, is such that all of its states are final, it is straightforward to see that its language is prefix-closed. Therefore, we have that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$ is prefix-closed, and so is $\mathcal{L}(\mathcal{R}(\mathcal{A}))$. Then, by Proposition 12, $\mathcal{L}(\mathcal{R}(\mathcal{A}))$ is a safety language.

To prove the left-to-right direction, notice that, since $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ is obtained from $\mathcal{A}$ by setting all its states as final, it holds that $\mathcal{L}(\mathcal{R}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$. Therefore, only the inclusion $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$ has to be proved.

To prove that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$, we divide in cases depending on whether the set I of initial states of $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ is empty.

If $I = \varnothing$, then $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \varnothing$ and clearly $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$.

If $I \neq \varnothing$, let q_0 be one of the initial states of $\mathcal{C}(\mathcal{R}(\mathcal{A}))$, let $\sigma \in \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$, and let q be one of the final states reached by $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ after reading σ. Since $\mathcal{R}(\mathcal{A})$ shares with its closure the same set of states and transition relation, q is a state of $\mathcal{R}(\mathcal{A})$ and is reached by $\mathcal{R}(\mathcal{A})$ after reading σ. Moreover, since $\mathcal{R}(\mathcal{A})$ is reduced, by definition of reduced automaton, there exists a $\sigma' \in \Sigma^*$ such that $\hat{\Delta}(q, \sigma') \cap F \neq \varnothing$, and thus $\sigma \cdot \sigma' \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$. Since by hypothesis $\mathcal{L}(\mathcal{R}(\mathcal{A}))$ is safety, it is also prefix-closed (Proposition 12), and thus $\sigma \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$. ◀

3.3 The complexity of recognizing safety languages of finite words

Now, we investigate the complexity of determining whether a language of finite words is a safety language, depending on the form in which the language is represented – either by automata or temporal logic. In both cases, we prove that the problem is PSPACE-complete. Interestingly, this is the same complexity as for the case of infinite words [25].

▶ **Proposition 15.** *Establishing whether the language accepted by an* NFA *is safety is* PSPACE*-complete.*

Proof (sketch). The membership in PSPACE follows from Proposition 14, and from the fact that the equivalence problem of two NFAs is a PSPACE problem. For the PSPACE-hardness, we use a reduction from the *universality problem* for NFAs, which is PSPACE-complete. The complete proof is provided in Appendix A. ◀

▶ **Proposition 16.** *Establishing whether the language of an* LTLf *formula is safety is* PSPACE*-complete.*

Proof (sketch). The PSPACE upper bound follows from the singly exponential construction of equivalent NFAs starting from LTLf formulas [13] and from the fact that the equivalence problem of two NFAs is a PSPACE problem. For proving the PSPACE-hardness, we use a reduction from the LTLf validity problem, which is PSPACE-complete. The complete proof is provided in Appendix A. ◀

3.4 Comparison of the safety fragments of LTL and LTLf

In this part, we compare the safety fragments of LTLf and LTL, and observe that certain formulae in LTLf, when interpreted over infinite words, are no longer safety, and conversely, some formulae in LTL cease to be safety when interpreted over finite words. This highlights that, with respect to the safety fragment, transitioning between LTLf and LTL must be done with care.

▶ **Proposition 17.** *It holds that:*
- *there exists an* LTL *formula* ϕ *such that* $\mathcal{L}^\infty(\phi)$ *is safety but* $\mathcal{L}(\phi)$ *is* not *safety; and*
- *there exists an* LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety but* $\mathcal{L}^\infty(\phi)$ *is* not *safety.*

Proof. To prove the first point, we take $\phi \coloneqq \mathsf{G}(p \to \mathsf{X}q)$. When interpreted over finite words, the language of ϕ is *not* safety. In fact, consider the word $\sigma \coloneqq \langle\{p\}\rangle$ of length 1. It holds that $\sigma \notin \mathcal{L}(\phi)$ because there is an occurrence of p that is not followed by any q. But none of its prefixes $\sigma' \in \mathtt{pref}(\sigma)$ is such that $\sigma' \cdot \sigma'' \notin \mathcal{L}(\phi)$, for all $\sigma'' \in (2^{\mathcal{AP}})^*$. In fact, if $\sigma' = \varepsilon$ then $\sigma' \cdot \{p\} \cdot \{q\} \in \mathcal{L}(\phi)$, while if $\sigma' = \langle\{p\}\rangle$ then $\sigma' \cdot \{q\} \in \mathcal{L}(\phi)$. It follows that $\mathcal{L}(\phi)$ is not a safety language. However, it is worth pointing out that, when interpreted over infinite words, the language of ϕ is safety. In fact, ϕ belongs to the syntactic safety fragment of LTL, *i.e.* SafetyLTL, and, by Proposition 4, $\mathcal{L}^\infty(\phi)$ is a safety language of infinite words.

To prove the second point, consider the following formula: $\phi \coloneqq \mathsf{G}(q\,\mathsf{U}\,(p \wedge q) \vee q\,\mathsf{U}\,(\widetilde{\mathsf{X}}\bot \wedge q))$. Under the interpretation over finite words, ϕ is equivalent to $\mathsf{G}(q\,\mathsf{U}\,(p \wedge q) \vee \mathsf{G}q)$, which in turn is equivalent to $\mathsf{G}(p\,\mathsf{R}\,q)$. To prove that $\mathcal{L}(\mathsf{G}(p\,\mathsf{R}\,q))$ is a safety language, observe that, for all $\sigma \in (2^{\mathcal{AP}})^*$, $\sigma \not\models \mathsf{G}(p\,\mathsf{R}\,q)$ if and only if $\sigma \models \mathsf{F}(\neg p\,\mathsf{U}\,\neg q)$. Now, for each of these words σ, there exists a prefix $\sigma' \in \mathtt{pref}(\sigma)$ which does not contain a q in its last position and thus it is irremediable, that is, $\sigma' \cdot \sigma'' \models \mathsf{F}(\neg p\,\mathsf{U}\,\neg q)$, for all $\sigma'' \in (2^{\mathcal{AP}})^*$. Therefore, $\mathcal{L}(\phi)$ is a safety language. However, under the interpretation over infinite words, ϕ is equivalent to $\mathsf{G}(q\,\mathsf{U}\,(p \wedge q))$, whose language is *not* safety: the infinite word $\{q\}^\omega$ does not satisfy $\mathsf{G}(q\,\mathsf{U}\,(p \wedge q))$ because p is never true, but each of its prefixes $\langle\{q\}, \ldots, \{q\}\rangle$ can be extended to an infinite word satisfying the formula, for example, by concatenating $\{p, q\}^\omega$. ◀

4 The liveness fragment on finite words

In this section, we define the *liveness* fragment of languages of finite words. We then show an effective way to recognize whether a language is liveness and we study the complexity of the problem. Finally, we compare liveness languages definable in LTLf with those definable in LTL.

4.1 Liveness languages of finite words

We define the liveness condition for languages of finite words as follows.

▶ **Definition 18.** *A language* $\mathcal{L} \subseteq \Sigma^*$ *is* liveness *if and only if, for all* $\sigma \in \Sigma^*$*, there exists* $\sigma' \in \Sigma^*$ *such that* $\sigma \cdot \sigma' \in \mathcal{L}$*.*

Given a formula ϕ of LTLf over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a liveness formula if $\mathcal{L}(\phi)$ is a liveness language over the alphabet $2^{\mathcal{AP}}$.

Examples. The LTLf formula $\mathsf{F}(p)$ over $\mathcal{AP} = \{p\}$ recognizes a liveness language, because any finite word $\sigma \in (2^{\mathcal{AP}})^*$ can be extended to a trace in the language of $\mathsf{F}(p)$ by concatenating $\langle\{p\}\rangle$. The same holds for the formula $\mathsf{GF}(p)$ and for all formulae of type $\mathsf{F}(\psi)$, where ψ

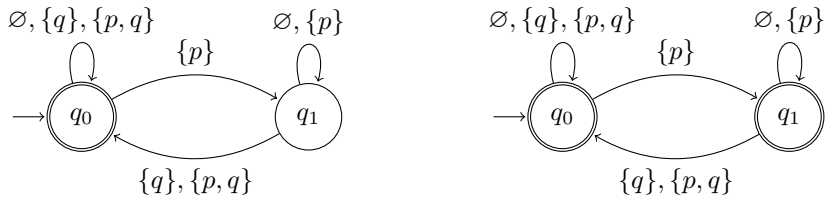


Figure 1 On the left, an automaton recognizing the LTLf formula $\mathsf{G}(p \to \mathsf{F}q)$. On the right, its closure, which accepts every finite word. By Proposition 19, the language of $\mathsf{G}(p \to \mathsf{F}q)$ fulfills the liveness condition.

is an LTLf formula. Conversely, the formula $\mathsf{G}(\neg p)$ does not recognize a liveness language, because the word $\langle\{p\}\rangle$ cannot be extended to a trace in the language of $\mathsf{G}(\neg p)$. Consider now the LTLf formula $\mathsf{F}(p \wedge \widetilde{\mathsf{X}}\bot)$ over $\mathcal{AP} = \{p\}$, stating that p holds in the final position of a word. Under finite word interpretation, we have $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^* \cdot \{p\}$. Clearly, $\mathcal{L}(\phi)$ is a liveness language. Under infinite word interpretation, instead, we have that $\mathcal{L}^\infty(\phi) = \varnothing$, because there exists no word containing a position whose successor satisfies the formula $\bot$. It follows that $\mathcal{L}^\infty(\phi)$ is *not* a liveness language of infinite words.

4.2 Recognizing liveness languages of finite words

In this section, we present an effective method for determining whether a language of finite words satisfies the liveness property. We subsequently analyze the computational complexity of this decision problem in two scenarios: when the language is represented by an NFA and when it is specified using an LTLf formula.

▶ **Proposition 19.** *Let $\mathcal{A}$ be an* NFA *over the alphabet Σ. $\mathcal{L}(\mathcal{A})$ is liveness if and only if $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \Sigma^*$.*

Proof. We start proving the left-to-right direction. Suppose that $\mathcal{L}(\mathcal{A})$ is liveness. Clearly, it always holds that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \Sigma^*$, thus we need only to prove that $\Sigma^* \subseteq \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$. Let $\sigma \in \Sigma^*$ be a finite word. Since $\mathcal{L}(\mathcal{A})$ is a liveness language and since $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{R}(\mathcal{A}))$, by Definition 18, there there exists $\sigma' \in \Sigma^*$ such that $\sigma \cdot \sigma' \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$, *i.e.* σ is the prefix of a word accepted by $\mathcal{L}(\mathcal{R}(\mathcal{A}))$. Therefore, there exists a state q of $\mathcal{R}(\mathcal{A})$ reached by $\mathcal{R}(\mathcal{A})$ after reading σ. By definition of closure automaton, state q is final in $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ and is reached by $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ after reading σ. It follows that $\sigma \in \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$.

We now prove the right-to-left direction. Suppose that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \Sigma^*$. We prove that the condition of liveness is satisfied by $\mathcal{L}(\mathcal{A})$. Let $\sigma \in \Sigma^*$. Since $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \Sigma^*$, it follows that there exists a state q reached by $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ after reading σ. Since, $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ and $\mathcal{R}(\mathcal{A})$ share the same set of states and the same transition relation, q is also a state of $\mathcal{R}(\mathcal{A})$ and is reached by $\mathcal{R}(\mathcal{A})$ after reading σ. Since $\mathcal{R}(\mathcal{A})$ is reduced, a final state of $\mathcal{R}(\mathcal{A})$ is reachable from state q, impling that there exists a word $\sigma' \in \Sigma^*$ such that $\sigma \cdot \sigma' \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$, proving that the liveness condition holds for $\mathcal{L}(\mathcal{R}(\mathcal{A}))$. Since the NFAs $\mathcal{R}(\mathcal{A})$ and $\mathcal{A}$ are equivalent, this means that also $\mathcal{L}(\mathcal{A})$ is a liveness language. ◀

As an example of application of Proposition 19, we consider the LTLf formula $\phi \coloneqq \mathsf{G}(p \to \mathsf{F}q)$ over the set of atomic propositions $\mathcal{AP} \coloneqq \{p, q\}$. The automaton $\mathcal{A}_\phi$, which is equal to its reduced version $\mathcal{R}(\mathcal{A}_\phi)$, recognizing $\mathcal{L}(\phi)$ is depicted in Figure 1 (left). The right side of Figure 1 displays its closure $\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))$. Since $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))) = (2^{\mathcal{AP}})^*$, by Proposition 19, the language of $\mathsf{G}(p \to \mathsf{F}q)$ is a liveness language.

We now investigate the computational complexity of deciding whether a language satisfies the liveness condition, in the cases where the language is given by an NFA and by an LTLf formula. In the former case, the problem is PSPACE-complete, as stated by the following proposition.

▶ **Proposition 20.** *Establishing whether the language accepted by an* NFA *is liveness is* PSPACE*-complete.*

Proof (sketch). Both the membership in PSPACE (*cf.* Proposition 19) and the PSPACE-hardness follows from the universality problem of NFAs, which is PSPACE-complete. The complete proof is provided in [14]. ◀

Interestingly, the best algorithm we have devised so far for deciding whether an LTLf formula is liveness requires exponential space. This is due to the fact that, given an LTLf formula ϕ, the algorithm first constructs the NFA corresponding to ϕ, which requires exponential space in $|\phi|$, and then checks the universality of its closure (*cf.* Proposition 19), a step that requires polynomial space in the size of the automaton. Overall, the algorithm operates in exponential space with respect to $|\phi|$. Whether this algorithm is optimal (*i.e.* EXPSPACE-hard), or whether a more efficient solution exists, remains an open problem – even in the case of infinite words.

▶ **Proposition 21.** *Establishing whether the language of an* LTLf *formula is liveness is in* EXPSPACE.

Proof. Let ϕ be an LTLf formula of size n over $\mathcal{AP}$. The algorithm to check whether ϕ is liveness proceeds as follows. First, it builds the NFA $\mathcal{A}_\phi$ equivalent to ϕ, which is of size $2^{\mathcal{O}(n)}$ [13]. Then, it constructs $\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))$ and checks whether $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))) = (2^{\mathcal{AP}})^*$, in space polynomial in the size of $\mathcal{A}_\phi$, that is, $2^{\mathcal{O}(n)}$. If this is the case, then ϕ is liveness, otherwise ϕ is not liveness. Overall, the algorithm requires space exponential in n, and thus the problem is in EXPSPACE. ◀

4.3 Comparison of the liveness fragments of LTL and LTLf

In the proposition below, we show that, as in the case of the safety fragment, there exist liveness formulae of LTLf that, when interpreted over infinite words, no longer accept liveness languages. Conversely, there also exist liveness formulae of LTL that, when interpreted over finite words, do not define a liveness language.

▶ **Proposition 22.** *It holds that:*

- *there exists an* LTL *formula ϕ such that $\mathcal{L}(\phi)$ is liveness but $\mathcal{L}^\infty(\phi)$ is not liveness; and*
- *there exists an* LTL *formula ϕ such that $\mathcal{L}^\infty(\phi)$ is liveness but $\mathcal{L}(\phi)$ is not liveness.*

Proof. To prove the first point, it suffices to take the formula $\mathsf{F}(p \wedge \widetilde{\mathsf{X}}\bot)$. As we shown above, when interpreted over finite words, the formula recognize a liveness language, while over infinite words it recognizes a language which is not liveness.

To prove the second point, consider the following formula: $\phi \coloneqq \mathsf{G}((\widetilde{\mathsf{X}}\bot \wedge q) \rightarrow \mathsf{F}(\widetilde{\mathsf{X}}\bot \wedge \neg q))$. Over finite words, it holds that $\mathcal{L}(\phi) = \varnothing$, because the formula forces the final time point of any word to satisfy both q and $\neg q$. It follows that $\mathcal{L}(\phi)$ is *not* a liveness language. However, over the infinite word interpretation, we have that $\mathcal{L}^\infty(\phi) = (2^{\mathcal{AP}})^\omega$, because the antecedent of the implication is always false. Therefore, $\mathcal{L}^\infty(\phi)$ is liveness. ◀

5 Decomposition of regular languages

In this section, we present a decomposition result for regular languages that leverages the characterization of the safety and liveness fragments. Specifically, we show that for any regular language $\mathcal{L}$, one can construct a safety language and a liveness language whose intersection is precisely $\mathcal{L}$.

From this point onward, we restrict our attention to DFAs. This assumption simplifies certain proofs, most notably that of Proposition 25. This restriction comes with no loss of generality, as every regular language $\mathcal{L}$ admits an equivalent DFA.

As a first step, we give the following definitions.

▶ **Definition 23** (The `Safe` and the `Live` versions of a DFA). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a* DFA. *We define* $\texttt{Safe}(\mathcal{A})$ *as* $\mathcal{C}(\mathcal{R}(\mathcal{A}))$. *We define* $\texttt{Live}(\mathcal{A})$ *as the automaton* $\mathcal{R}(\mathcal{A})$ *augmented with a new final state $q_f \notin Q$, and the following transitions: (i) (q_f, a, q_f), for all $a \in \Sigma$; (ii) (q, a, q_f), for all $q \in Q$ and for all $a \in \Sigma$ such that $\Delta(q, a) = \varnothing$.*

The proposition below states that, for any DFA $\mathcal{A}$, the automata $\texttt{Safe}(\mathcal{A})$ and $\texttt{Live}(\mathcal{A})$ recognize a safety and a liveness language, respectively. We provide the proof in Appendix A.

▶ **Proposition 24.** *Let $\mathcal{A}$ be a* DFA. *It holds that $\mathcal{L}(\texttt{Safe}(\mathcal{A}))$ (resp., $\mathcal{L}(\texttt{Live}(\mathcal{A}))$) is a safety language (resp., a liveness language).*

The final step before presenting the decomposition algorithm is to show that, for any DFA $\mathcal{A}$, the automaton $\texttt{Live}(\mathcal{A})$ recognizes exactly the set of words that are either accepted by $\mathcal{A}$ or not accepted by $\texttt{Safe}(\mathcal{A})$. This property is essential to ensure that the intersection of $\texttt{Safe}(\mathcal{A})$ and $\texttt{Live}(\mathcal{A})$ recognizes precisely the language $\mathcal{L}(\mathcal{A})$.

▶ **Proposition 25.** *Let $\mathcal{A}$ be a* DFA. $\texttt{Live}(\mathcal{A})$ *recognizes the language $\mathcal{L}(\mathcal{A}) \cup (\Sigma^* \setminus \mathcal{L}(\texttt{Safe}(\mathcal{A})))$.*

Proof. We provide the proof in [14]. ◀

The following theorem states and proves the decomposition theorem for regular languages, establishing that every regular language over an alphabet Σ is expressible as an intersection of a safety language over Σ and a liveness language over Σ.

▶ **Theorem 26.** *Let $\mathcal{L} \subseteq \Sigma^*$ be a regular language. There exist two regular languages $\mathcal{L}_{safe} \subseteq \Sigma^*$ and $\mathcal{L}_{live} \subseteq \Sigma^*$ such that: (i) $\mathcal{L}_{safe}$ is a safety language; (ii) $\mathcal{L}_{live}$ is a liveness language; (iii) $\mathcal{L} = \mathcal{L}_{safe} \cap \mathcal{L}_{live}$.*

Proof. Since $\mathcal{L}$ is a regular language, there exists a DFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. Let $\mathcal{L}_{safe} = \mathcal{L}(\texttt{Safe}(\mathcal{A}))$ and $\mathcal{L}_{live} = \mathcal{L}(\texttt{Live}(\mathcal{A}))$.

By Proposition 24, $\mathcal{L}_{safe}$ and $\mathcal{L}_{live}$ are a safety language and a liveness language, respectively. Moreover, by Proposition 25, it holds that:

$$\begin{aligned}\mathcal{L}_{safe} \cap \mathcal{L}_{live} &= \mathcal{L}(\texttt{Safe}(\mathcal{A})) \cap \mathcal{L}(\texttt{Live}(\mathcal{A})) = \mathcal{L}(\texttt{Safe}(\mathcal{A})) \cap (\mathcal{L}(\mathcal{A}) \cup (\Sigma^* \setminus \mathcal{L}(\texttt{Safe}(\mathcal{A}))))\\ &= \mathcal{L}(\texttt{Safe}(\mathcal{A})) \cap \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A})\end{aligned}$$

Therefore, $\mathcal{L} = \mathcal{L}_{safe} \cap \mathcal{L}_{live}$. ◀

6 Implications on LTLf Model Checking

In this section, we examine certain implications of the safety and liveness fragments on the *model checking* problem [12] over finite words. Specifically, we address the problem of verifying whether an NFA M satisfies an LTLf formula ϕ, *i.e.* whether $\mathcal{L}(M) \subseteq \mathcal{L}(\phi)$, denoted as $M \models \phi$.

The identification of safety and liveness fragments within both LTLf and the class of regular languages over finite words opens avenues for designing specialized model checking procedures and conducting a more granular complexity analysis – paralleling existing results in the setting of infinite words [19, 11].

However, we show that when $\mathcal{L}(M)$ is assumed to be a nonempty safety language – a common scenario in practice, particularly in invariant checking benchmarks [5] – the model checking problem for LTLf formulae often becomes uninformative. In many such cases, formulae that are semantically meaningful and nontrivial in the infinite-word setting give rise to degenerate instances in the finite-word setting, where the model checking task becomes trivial: either always false, or reducible to a simple condition such as whether the empty word ε belongs to $\mathcal{L}(M)$ or the model checking of a much simpler formula.

6.1 The cosafety case

We begin by analyzing the case where $\mathcal{L}(\phi)$ is a *cosafety* language – that is, the complement of a safety language. Consider, for instance, the formula $\phi \coloneqq \mathsf{F}p$. Since $\mathcal{L}(M)$ is assumed to be a safety language and thus is prefix-closed (*cf.* Definition 11 and Proposition 12), it necessarily contains the empty word ε. However, since ϕ is a cosafety formula, $\overline{\mathcal{L}(\phi)}$ is a safety language, and it holds that $\varepsilon \in \overline{\mathcal{L}(\phi)}$, that is, $\varepsilon \notin \mathcal{L}(\phi)$. Consequently, $\mathcal{L}(M) \not\subseteq \mathcal{L}(\phi)$, *i.e.* $M \not\models \phi$. More generally, there does not exist an NFA M such that $\mathcal{L}(M)$ is a safety, nonempty language and $M \models \phi$, with $\phi \coloneqq \mathsf{F}p$. It is worth noting that this conclusion remains valid even if one were to disregard ε from the language, in which case the model checking task reduces to a trivial condition: verifying whether the initial state satisfies p.

This reasoning extends to arbitrary cosafety properties. Let ϕ be a cosafety LTLf formula and let M be an NFA such that $\mathcal{L}(M)$ is safety and nonempty. One has that $M \models \phi$ if and only if $\mathcal{L}(M) \cap \overline{\mathcal{L}(\phi)} = \varnothing$. Since $\mathcal{L}(M)$ is safety and thus prefix-closed, by Corollary 13 we have that $\varepsilon \in \mathcal{L}(M)$, and thus:

1. if $\varepsilon \not\models \phi$ (that is, $\varepsilon \in \overline{\mathcal{L}(\phi)}$), then $\mathcal{L}(M) \cap \overline{\mathcal{L}(\phi)} \neq \varnothing$, and thus $M \not\models \phi$;
2. if $\varepsilon \models \phi$ (that is, $\varepsilon \in \mathcal{L}(\phi)$), then $\varepsilon \notin \overline{\mathcal{L}(\phi)}$; but since $\overline{\mathcal{L}(\phi)}$ is a safety language, by Corollary 13 it holds that $\overline{\mathcal{L}(\phi)} = \varnothing$, implying that $\mathcal{L}(M) \cap \overline{\mathcal{L}(\phi)} = \varnothing$ and $M \models \phi$.

Therefore, checking whether $M \models \phi$ is equivalent to checking whether $\varepsilon \not\models \phi$, for all cosafety formulas ϕ and for all safety, nonempty NFAs M.

6.2 The general case

We now establish that for any LTLf formula ϕ and any NFA M such that $\mathcal{L}(M)$ is a safety language, the model checking problem $M \models \phi$ reduces to verifying whether $\mathcal{L}(M)$ is contained within a substantially simpler language than $\mathcal{L}(\phi)$. Specifically, it suffices to check inclusion in the language recognized by the automaton corresponding to ϕ after removing all non-final states. As an example, suppose that $\phi \coloneqq \mathsf{GF}p$. For any finite word σ, we have that $\sigma \models \mathsf{GF}p$ if and only if σ satisfies p in the last position. Moreover, since $\mathcal{L}(M)$ is prefix-closed, it means that, for every $\sigma \in \mathcal{L}(M)$, proposition p holds in *every* position of σ. Therefore, $M \models \mathsf{GF}p$ is equivalent to $M \models \mathsf{G}p$.

We define the *subclosure* of an NFA as the automaton resulting from the removal of all non-final states along with their associated incoming and outgoing transitions.

▶ **Definition 27** (Subclosure automaton). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an* NFA*. The subclosure of $\mathcal{A}$, denoted with $\mathcal{S}(\mathcal{A})$, is the automaton obtained from $\mathcal{A}$ by removing non-final states,* i.e. *$(Q \cap F, \Sigma, I \cap F, \Delta \cap (F \times \Sigma \times F), F)$.*

The following proposition establishes that, for any NFA M such that $\mathcal{L}(M)$ is safety and any LTLf formula ϕ, the model checking problem $M \models \phi$ is equivalent to checking the language inclusion $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$, where $\mathcal{A}_\phi$ is the DFA corresponding to ϕ.

▶ **Proposition 28.** *Let ϕ be an* LTLf *formula over $\mathcal{AP}$ and let $\mathcal{A}_\phi$ be its equivalent* DFA*. Let M be an* NFA *such that $\mathcal{L}(M) \subseteq (2^{\mathcal{AP}})^*$ is a safety language. It holds that $M \models \phi$ if and only if $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$.*

Proof. The right-to-left direction is trivial since $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) \subseteq \mathcal{L}(\mathcal{A}_\phi)$ and thus, if $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$, then $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A}_\phi)$, that is $M \models \phi$.

To prove the left-to-right direction, assume that $M \models \phi$, that is $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A}_\phi)$, and let $\sigma \in \mathcal{L}(M)$. We prove that $\sigma \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$ by induction on the length of σ.

Let $|\sigma| = 0$. Thus $\sigma = \varepsilon$. Since σ belongs also to $\mathcal{L}(\mathcal{A}_\phi)$, there exists an initial state of $\mathcal{A}_\phi$ that is final. By Definition 27, such state is also initial and final in $\mathcal{S}(\mathcal{A}_\phi)$. Thus, $\varepsilon \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$.

Let $|\sigma| = n > 0$. Since $\sigma \in \mathcal{L}(\mathcal{A}_\phi)$, there exists a path $\langle q_0, q_1, \ldots, q_n \rangle$ accepting σ. Since $\mathcal{L}(M)$ is safety and thus prefix-closed, also the prefix σ' of size $n-1$ is in $\mathcal{L}(\mathcal{A}_\phi)$. Moreover, since $\mathcal{A}_\phi$ is a DFA, $\langle q_0, q_1, \ldots, q_{n-1} \rangle$ must be accepting, having that σ' belongs to $\mathcal{L}(\mathcal{A}_\phi)$. By inductive hypothesis, $\sigma' \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$. Since both q_n and q_{n-1} are final states in $\mathcal{A}_\phi$, by the definition of $\mathcal{S}$ there is a transition from q_{n-1} to q_n, having that $\langle q_0, q_1, \ldots, q_n \rangle$ must be a sequence of $\mathcal{S}(\mathcal{A}_\phi)$ accepting σ. Therefore, $\sigma \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$. ◀

It is important to note that Proposition 28 remains valid regardless of whether the empty word ε is included in the semantics. In particular, if the model checking problem $M \models \varphi$ is to be interpreted with the exclusion of ε, one can equivalently verify $M \models (\phi \vee \mathsf{G}\bot)$, since $\mathcal{L}(\mathsf{G}\bot) = \{\varepsilon\}$.

Some important consequences of Proposition 28 are highlighted by the following representative examples, which hold for any NFA M such that $\mathcal{L}(M)$ is safety and nonempty:

- if $\phi = \mathsf{F}p$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \varnothing$ and therefore it *never* holds that $M \models \phi$;
- if $\phi = \mathsf{F}p \vee \mathsf{G}\bot$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \{\varepsilon\} \cup \{p\} \cdot \Sigma^*$ and, since $\varepsilon \in L(M)$, it holds that $M \models \phi$ iff $M \models p$;
- if $\phi = \mathsf{GF}p$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \mathcal{L}(\mathsf{G}p)$ and $M \models \phi$ iff $M \models \mathsf{G}p$;
- if $\phi = \mathsf{FG}p$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \varnothing$ and thus it *never* holds that $M \models \phi$;
- if $\phi = \mathsf{FG}p \vee \mathsf{G}\bot$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \mathcal{L}(\mathsf{G}p)$, and thus $M \models \phi$ iff $M \models \mathsf{G}p$;

7 Conclusions

In this work, we investigated the notions of safety and liveness languages over finite words. We established several fundamental properties of these classes, including the prefix-closed nature of safety languages. Furthermore, we presented effective procedures to determine whether the language recognized by an NFA or specified by an LTLf formula is safety or liveness. We also discussed key distinctions between the finite-word and infinite-word settings. In addition, we proved that, analogously to the infinite-word case, every regular language can

be represented as the intersection of a safety language and a liveness language. We further examined implications for model checking of LTLf formulae, particularly when the program under verification recognizes a safety regular language.

As a direction for future work, we conjecture that the problem of deciding whether the language defined by an LTLf formula is a liveness language is EXPSPACE-complete, a result that remains open in this paper. Furthermore, a natural extension of our results involves the study of *relative safety* and *relative liveness*, as introduced by T. Henzinger in [15] and studied further in [6], that refine the classical definitions by characterizing safety and liveness properties with respect to a given environmental assumption. Investigating these refined notions in the finite-word setting represents an interesting direction for future research.

References

1 Bowen Alpern and Fred B Schneider. Defining liveness. *Information processing letters*, 21(4):181–185, 1985. doi:10.1016/0020-0190(85)90056-0.

2 Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

3 Suguman Bansal, Yong Li, Lucas M. Tabajara, Moshe Y. Vardi, and Andrew M. Wells. Model checking strategies from synthesis over finite traces. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I*, volume 14215 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2023. doi:10.1007/978-3-031-45329-8_11.

4 David A. Basin, Carlos Cotrini Jiménez, Felix Klaedtke, and Eugen Zalinescu. Deciding safety and liveness in TPTL. *Inf. Process. Lett.*, 114(12):680–688, 2014. doi:10.1016/J.IPL.2014.06.005.

5 Armin Biere, Nils Froleyks, and Mathias Preiner. Hardware model checking competition 2024. In Nina Narodytska and Philipp Rümmer, editors, *Formal Methods in Computer-Aided Design, FMCAD 2024, Prague, Czech Republic, October 15-18, 2024*, page 1. IEEE, 2024. doi:10.34727/2024/ISBN.978-3-85448-065-5_6.

6 Alberto Bombardelli, Alessandro Cimatti, Stefano Tonetta, and Marco Zamboni. Symbolic Model Checking of Relative Safety LTL Properties. In *iFM*, volume 14300 of *Lecture Notes in Computer Science*, pages 302–320. Springer, 2023. doi:10.1007/978-3-031-47705-8_16.

7 Aaron R Bradley. SAT-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 70–87. Springer, 2011. doi:10.1007/978-3-642-18275-4_7.

8 Aaron R Bradley. Understanding IC3. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 1–14. Springer, 2012. doi:10.1007/978-3-642-31612-8_1.

9 Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In Werner Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 474–486. Springer, 1992. doi:10.1007/3-540-55719-9_97.

10 Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. A first-order logic characterisation of safety and co-safety languages. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13242 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2022. doi:10.1007/978-3-030-99253-8_13.

11 Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In *2012 Formal Methods in Computer-Aided Design (FMCAD)*, pages 52–59. IEEE, 2012. URL: https://ieeexplore.ieee.org/document/6462555/.

12 Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of model checking*, volume 10. Springer, 2018. doi:10.1007/978-3-319-10575-8.

13 Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13*, pages 854–860, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

14 Luca Geatti, Stefano Pessotto, and Stefano Tonetta. Safety and liveness on finite words (extended version). https://users.dimi.uniud.it/~luca.geatti/data/time-25/time25.pdf, 2025. Extended version of the paper.

15 Thomas A. Henzinger. Sooner is safer than later. *Inf. Process. Lett.*, 43(3):135–141, 1992. doi:10.1016/0020-0190(92)90005-G.

16 Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Quantitative Safety and Liveness. In *FoSSaCS*, volume 13992 of *Lecture Notes in Computer Science*, pages 349–370. Springer, 2023. doi:10.1007/978-3-031-30829-1_17.

17 John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001. doi:10.1145/568438.568455.

18 Orna Kupferman and Gal Vardi. On relative and probabilistic finite counterability. *Formal Methods Syst. Des.*, 52(2):117–146, 2018. doi:10.1007/S10703-017-0277-8.

19 Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001. doi:10.1023/A:1011254632723.

20 Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977. doi:10.1109/TSE.1977.229904.

21 Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

22 Doron Peled and Klaus Havelund. Refining the Safety-Liveness Classification of Temporal Properties According to Monitorability. In *Models, Mindsets, Meta*, volume 11200 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2018. doi:10.1007/978-3-030-22348-9_14.

23 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.

24 A Prasad Sistla. On characterization of safety and liveness properties in temporal logic. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 39–48, 1985. doi:10.1145/323596.323600.

25 A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994. doi:10.1007/BF01211865.

A Omitted proofs

▶ **Corollary 13** (Small and Bounded Model Property for safety languages). *Let $\mathcal{L} \subseteq \Sigma^*$ be a safety language. It holds that $\mathcal{L} \neq \varnothing$ if and only if $\varepsilon \in \mathcal{L}$.*

Proof. The right-to-left direction is trivial. To prove the left-to-right direction, suppose that $\mathcal{L}$ is not empty and let $\sigma \in \mathcal{L}$. Since by hypothesis $\mathcal{L}$ is safety, by Proposition 12 we have that $\mathcal{L}$ is prefix-closed. Since $\sigma \in \mathcal{L}$ and since ε is a prefix of σ, by Definition 11 we have that $\varepsilon \in \mathcal{L}$. ◀

▶ **Proposition 15.** *Establishing whether the language accepted by an* NFA *is safety is* PSPACE*-complete.*

Proof. For the membership in PSPACE, we show the following procedure to check whether the language of a given reduced NFA is equal to the language of its closure (Proposition 14). Let $\mathcal{A}$ be an NFA of size n. By means of a sequence of reachability checks, in nondeterministic

logarithmic space, we can turn $\mathcal{A}$ into an equivalent reduced NFA $\mathcal{R}(\mathcal{A})$ of size $\mathcal{O}(n)$. The procedure checks whether $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$. Notice that, since $\mathcal{L}(\mathcal{R}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$, only the opposite inclusion needs to be checked. Checking whether $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$ is equivalent to check whether:

$$\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \cap \overline{\mathcal{L}(\mathcal{R}(\mathcal{A}))} = \varnothing$$

Building an automaton for $\overline{\mathcal{L}(\mathcal{R}(\mathcal{A}))}$ requires $2^{\mathcal{O}(n)}$ space [17], computing the intersection between two automata requires polynomial space with respect to the size of the two automata, and checking the emptiness can be performed *on-the-fly* in nondeterminstic logarithmic space. The total complexity is thus nondeterministic polynomial space, and thus the problem is in PSPACE.

For proving the PSPACE-hardness, we consider the *universality problem* for NFAs, *i.e.* the problem of checking whether the language of a given NFA over the alphabet Σ is Σ^*, which is known to be PSPACE-complete.

Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an NFA. We define the automaton $\mathcal{B}$ as the NFA $(Q', \Sigma', I', \Delta', F')$ such that:

- $Q' \coloneqq Q \cup \{q_{sink}, q_{loop}\}$;
- $\Sigma' \coloneqq \Sigma \cup \{f\}$, where $f \notin \Sigma$;
- $I' \coloneqq I$;
- $\Delta' \coloneqq \Delta \cup \{(q, f, q_{loop}) \mid q \in Q'\} \cup$
  $\{(q, a, q_{sink}) \mid q \in Q,\ a \in \Sigma,\ \Delta(q, a) = \varnothing\} \cup$
  $\{(q, a, q) \mid a \in \Sigma,\ q \in \{q_{sink}, q_{loop}\}\}$;
- $F' \coloneqq F \cup \{q_{loop}\}$.

Automaton $\mathcal{B}$ has two crucial properties: (i) it has no undefined transitions; (ii) $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup (\Sigma')^* \cdot f \cdot (\Sigma')^*$; note that this implies that $\mathcal{L}(\mathcal{B}) \cap (\Sigma)^* = \mathcal{L}(\mathcal{A})$.

We prove that $\mathcal{L}(\mathcal{A}) = \Sigma^*$ if and only if $\mathcal{L}(\mathcal{B})$ is safety.

Suppose that $\mathcal{L}(\mathcal{A}) = \Sigma^*$ and let σ be any word in $(\Sigma')^*$. We divide in cases, depending on whether σ contains at least one f.

Case 1. If σ does not contain any f, then $\sigma \in \Sigma^*$ and thus $\sigma \in \mathcal{L}(\mathcal{A})$. Since by construction $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup (\Sigma')^* \cdot f \cdot (\Sigma')^*$, we have that $\sigma \in \mathcal{L}(\mathcal{B})$.

Case 2. If σ contains at least one f, then $\sigma \in (\Sigma')^* \cdot f \cdot (\Sigma')^*$, and thus, also in this case, $\sigma \in \mathcal{L}(\mathcal{B})$.

Therefore $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$ and, clearly, $\mathcal{L}(\mathcal{B})$ is a safety language over the alphabet Σ'.

Suppose that $\mathcal{L}(\mathcal{B})$ is a safety language. We first prove that $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$. Suppose by contradiction that this is not case and let $\sigma \notin \mathcal{L}(\mathcal{B})$. Since $\mathcal{L}(\mathcal{B})$ is a safety language, there exists a $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \notin \mathcal{L}(\mathcal{B})$, for all $\sigma'' \in (\Sigma')^*$. Now, let q be any state reached by $\mathcal{B}$ after reading σ' (notice that, since $\mathcal{B}$ does not contain undefined transitions, state q always exists). By construction of $\mathcal{B}$, reading the symbol f, $\mathcal{B}$ transitions from state q to state q_{loop}. Since q_{loop} is a final state, this means that $\sigma' \cdot f \in \mathcal{L}(\mathcal{B})$. But this is a contradiction with the fact that $\sigma' \cdot \sigma'' \notin \mathcal{L}(\mathcal{B})$, for all $\sigma'' \in (\Sigma')^*$. Therefore, $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$.

As observed above, by construction, $\mathcal{L}(\mathcal{B}) \cap (\Sigma)^* = \mathcal{L}(\mathcal{A})$. Thus, the fact that $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$ implies that $\mathcal{L}(\mathcal{A}) = \Sigma^*$. ◀

▶ **Proposition 16.** *Establishing whether the language of an* LTLf *formula is safety is* PSPACE*-complete.*

Proof. We follow the same approach as Sistla [25] for proving that the problem of establishing whether the language recognized by an LTL formula is safety is PSPACE-complete.

As for the membership in PSPACE, we follow this procedure. Given an LTLf formula ϕ of size n over the atomic propositions $\mathcal{AP}$, the procedure builds two NFAs over the alphabet $2^{\mathcal{AP}}$, both of size $2^{\mathcal{O}(n)}$ [13]: the automaton $\mathcal{A}_\phi$ equivalent to the formula ϕ, and the automaton $\mathcal{A}_{\neg\phi}$ equivalent to the formula $\neg\phi$. To check whether $\mathcal{L}(\phi)$ is a safety language, it suffices to check whether $\mathcal{L}(\mathcal{R}(\mathcal{A}_\phi)) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi)))$ (Proposition 14), which is equivalent to check whether $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))) \cap \mathcal{L}(\mathcal{R}(\mathcal{A}_{\neg\phi})) = \varnothing$. This check can be done in nondeterministic logarithmic space, *on-the-fly* while building the two automata. Therefore, the problem is in PSPACE.

To prove the PSPACE-hardness, we use the validity problem for LTLf formulae, *i.e.* checking whether the language of a given LTLf formula over the set of atomic propositions $\mathcal{AP}$ is $(2^{\mathcal{AP}})^*$. Let ϕ be an LTLf formula over $\mathcal{AP}$. We prove that ϕ is valid (*i.e.* $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$) if and only if $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety, where $p \notin \mathcal{AP}$.

Proving that if $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$ then $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety is straightforward. Since $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$ and p does not appear in ϕ, it also holds that $\mathcal{L}(\phi \vee \mathsf{F}p) = (2^{\mathcal{AP}\cup\{p\}})^*$, and thus $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety.

We now prove the opposite direction. Suppose that $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety. Suppose by contradiction that $\phi \vee \mathsf{F}p$ is not valid and let $\sigma \in (2^{\mathcal{AP}\cup\{p\}})^*$ be a word such that $\sigma \not\models \phi \vee \mathsf{F}p$. Since by hypothesis the language of $\phi \vee \mathsf{F}p$ is safety, there exists a $\sigma' \in \mathtt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \not\models \phi \vee \mathsf{F}p$, for all $\sigma'' \in (2^{\mathcal{AP}\cup\{p\}})^*$. However, this is a contradition because the $\sigma' \cdot \{p\} \models \phi \vee \mathsf{F}p$. Therefore, $\phi \vee \mathsf{F}p$ is valid, *i.e.* $\mathcal{L}(\phi \vee \mathsf{F}p) = (2^{\mathcal{AP}\cup\{p\}})^*$. Since $p \notin \mathcal{AP}$, this means that $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$. ◀

▶ **Proposition 24.** *Let $\mathcal{A}$ be a* DFA*. It holds that $\mathcal{L}(\mathtt{Safe}(\mathcal{A}))$ (resp., $\mathcal{L}(\mathtt{Live}(\mathcal{A}))$) is a safety language (resp., a liveness language).*

Proof. We first prove the case for $\mathtt{Safe}(\mathcal{A})$. Since, by Definition 23, $\mathtt{Safe}(\mathcal{A})$ is defined as $\mathcal{C}(\mathcal{R}(\mathcal{A}))$, it holds that $\mathcal{C}(\mathcal{R}(\mathtt{Safe}(\mathcal{A})))$ is exactly $\mathtt{Safe}(\mathcal{A})$. In particular $\mathcal{L}(\mathtt{Safe}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathtt{Safe}(\mathcal{A}))))$. By Proposition 14, we have that $\mathcal{L}(\mathtt{Safe}(\mathcal{A}))$ is a safety language.

We now prove the case for $\mathtt{Live}(\mathcal{A})$. The definition of $\mathtt{Live}(\mathcal{A})$ states that all undefined transitions of $\mathcal{R}(\mathcal{A})$ are replaced with a transition into a new, final state (to which the automaton is forced to remain reading any symbol). This means that the transition relation of $\mathtt{Live}(\mathcal{A})$ is *complete*: $\Delta(q, a) \neq \varnothing$, for all states q of $\mathtt{Live}(\mathcal{A})$ and for all symbols $a \in \Sigma$. Since all the states in $\mathcal{C}(\mathcal{R}(\mathtt{Live}(\mathcal{A})))$ are set to final, it follows that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathtt{Live}(\mathcal{A})))) = \Sigma^*$. By Proposition 19, it means that $\mathcal{L}(\mathtt{Live}(\mathcal{A}))$ is a liveness language. ◀

A Better Algorithm for Converting an STNU into Minimal Dispatchable Form

Luke Hunsberger
Vassar College, Poughkeepsie, NY, USA

Roberto Posenato
University of Verona, Italy

Abstract

A Simple Temporal Network with Uncertainty (STNU) is a data structure for representing and reasoning about temporal constraints on activities, including those with uncertain durations. An STNU is dispatchable if it can be flexibly and efficiently executed in real time while guaranteeing that all relevant constraints are satisfied. Typically, dispatchability requires inserting conditional *wait* constraints, thereby forming an *Extended* STNU (ESTNU). The number of edges in an ESTNU affects the computational work that must be done during real-time execution. The MinDispESTNU problem is that of finding an equivalent dispatchable ESTNU having a minimal number of edges. Recent work presented an $O(kn^3)$-time algorithm for solving the MinDispESTNU problem, where n is the number of timepoints and k is the number of actions with uncertain durations. A subsequent paper presented a faster $O(n^3)$-time algorithm, but it has been shown to be incomplete. This paper presents a new $O(mn + n^2k + n^2 \log n)$-time algorithm for solving the MinDispESTNU problem, where m is the number of constraints in the network. The correctness of the algorithm is based on a novel theory of the *canonical form of nested diamond structures.* An empirical evaluation demonstrates the order-of-magnitude improvement in performance.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Theory of computation → Dynamic graph algorithms

Keywords and phrases Temporal constraint networks, dispatchable networks

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.11

1 Background

Temporal constraint networks facilitate representing and reasoning about temporal constraints on activities. *Simple Temporal Networks with Uncertainty* (STNUs) allow the explicit representation of actions with uncertain durations [13]. An STNU is *dispatchable* if it can be executed by a flexible and efficient real-time execution algorithm while guaranteeing that all of its constraints will be satisfied. This paper modifies an existing algorithm for converting a dispatchable network into an equivalent dispatchable network having a minimal number of edges, making it an order of magnitude faster, as demonstrated by an empirical evaluation.

Simple Temporal Networks. A *Simple Temporal Network* (STN) is a pair $(\mathcal{T}, \mathcal{C})$ where $\mathcal{T}$ is a set of real-valued variables called timepoints; and $\mathcal{C}$ is a set of *ordinary* constraints, each of the form $(Y - X \leq \delta)$ for $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$ [3]. An STN is *consistent* if it has a solution as a constraint satisfaction problem (CSP). Each STN has a corresponding graph where the timepoints serve as nodes and the constraints correspond to labeled, directed edges. In particular, each constraint $(Y - X \leq \delta)$ corresponds to an edge $X \xrightarrow{\delta} Y$ in the graph. Such edges may be notated as (X, δ, Y) or, if context permits, simply XY. A path from X to Y may be notated by listing its timepoints (e.g., $XUVWY$) or, if the context permits, just XY.

A flexible and efficient *real-time execution* (RTE) algorithm has been defined for STNs that maintains a time window for each timepoint X and, as each X is executed, propagates constraints only *locally,* to X's *neighbors* in the graph [19, 15]. An STN is *dispatchable* if that RTE algorithm is guaranteed to satisfy all of the STN's constraints no matter how the flexibility afforded by the algorithm is exploited during execution. A consistent STN is dispatchable if and only if each pair of timepoints connected by a path in the graph are connected by a shortest *vee-path* (i.e., a shortest path comprising zero or more negative edges followed by zero or more non-negative edges) [12]. Efficient algorithms for generating equivalent dispatchable STNs having a *minimal number of edges* have been presented [19, 15]. Having fewer edges is important since it lessens real-time computations done during execution.

Simple Temporal Networks with Uncertainty. A *Simple Temporal Network with Uncertainty* (STNU) augments an STN to include *contingent links* that represent actions with uncertain, but bounded durations [13]. An STNU is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ where $(\mathcal{T}, \mathcal{C})$ is an STN, and $\mathcal{L}$ is a set of contingent links, each of the form (A, x, y, C), where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$. The semantics of STNU execution ensures that regardless of when the *activation timepoint* A is executed, the *contingent timepoint* C will occur such that $C - A \in [x, y]$. Each STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ has a corresponding graph $\mathcal{G} = (\mathcal{T}, \mathcal{E}_{\rm o}, \mathcal{E}_{\rm lc}, \mathcal{E}_{\rm uc})$, where $(\mathcal{T}, \mathcal{E}_{\rm o})$ is the graph for the STN $(\mathcal{T}, \mathcal{C})$, and $\mathcal{E}_{\rm lc}$ and $\mathcal{E}_{\rm uc}$ are sets of *labeled* edges corresponding to the contingent durations in $\mathcal{L}$. In particular, each contingent link (A, x, y, C) in $\mathcal{L}$ has a *lower-case* (LC) edge $A \xrightarrow{c:x} C$ in $\mathcal{E}_{\rm lc}$ that represents the uncontrollable possibility that the duration might take on its minimum value x; and an *upper-case* (UC) edge $C \xrightarrow{C:-y} A$ in $\mathcal{E}_{\rm uc}$ that represents the possibility that it might take on its maximum value y. For convenience, edges such as $A \xrightarrow{c:x} C$ and $C \xrightarrow{C:-y} A$ may be notated as $(A, c{:}x, C)$ and $(C, C{:}-y, A)$, respectively.

An STNU is *dynamically controllable* (DC) if there exists a dynamic, real-time execution strategy that guarantees that all constraints in $\mathcal{C}$ will be satisfied no matter how the contingent durations turn out [13, 4]. A dynamic strategy is one whose execution decisions can react to observations of contingent executions, but without advance knowledge of future events. Many polynomial-time DC-checking algorithms have been presented [11, 1, 5], the fastest having a worst-case time-complexity of $O(mn + k^2 n + kn \log n)$, where n, m and k are the numbers of timepoints, ordinary constraints, and contingent links. Many DC-checking algorithms generate a kind of conditional constraint called a *wait* [14, 10, 5]. Although not necessary for DC-checking [1], wait constraints are needed for STNU dispatchability, as follows.

An STNU augmented with a set of waits, $\mathcal{E}_w$, is called an *Extended STNU* (ESTNU) [11]. A real-time execution algorithm for ESTNUs, called RTE*, has been defined that provides maximum flexibility while requiring minimal real-time computation [11, 7]. An ESTNU is dispatchable if every run of the RTE* algorithm is guaranteed to satisfy all of its constraints no matter how the contingent durations turn out. Equivalently, an ESTNU is dispatchable if and only if all of its STN *projections* are dispatchable (as STNs) [11]. (A projection of an ESTNU is the STN that results from fixing the durations of its contingent links.) The fastest algorithm for generating equivalent dispatchable ESTNUs is the $O(mn + kn^2 + n^2 \log n)$-time $\texttt{FD}_{\texttt{STNU}}$ algorithm [6], but it provides no guarantee about the number of edges in its output.

The MinDispESTNU problem. For any given dispatchable ESTNU $\mathcal{G}$, find an equivalent dispatchable ESTNU $\mathcal{G}'$ having a minimal number of edges. The $\texttt{minDisp}_{\texttt{ESTNU}}$ algorithm [7] solves the MinDispESTNU problem in $O(kn^3)$ time. A faster $O(n^3)$-time algorithm, called $\texttt{fastMinDisp}_{\texttt{ESTNU}}$ [8], was later found to be incomplete.

Figure 1 Stand-in edges entailed by labeled edges associated with contingent links.

This paper. Section 2 summarizes the `minDisp`$_{\text{ESTNU}}$ and `fastMinDisp`$_{\text{ESTNU}}$ algorithms. Section 3 then presents a new algorithm, `betterMinDispESTNU`, that solves the MinDispESTNU problem in $O(mn + n^2k + n^2 \log n)$ time. It employs a novel approach to generating so-called *stand-in* edges. The correctness of the algorithm is based on a new theory of the *canonical form of nested diamond structures,* which is detailed in Hunsberger and Posenato [9]. Section 4 presents an empirical evaluation that demonstrates that `betterMinDispESTNU` achieves an order-of-magnitude speedup over `minDisp`$_{\text{ESTNU}}$ in practice.

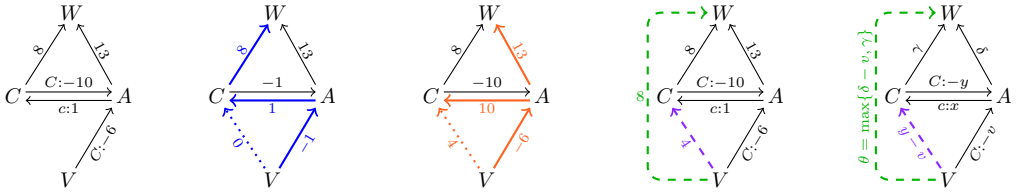
2 Overview of Existing Algorithms

The `minDisp`$_{\text{ESTNU}}$ algorithm [7] takes a dispatchable ESTNU $\mathcal{E} = (\mathcal{T}, \mathcal{E}_{\text{o}}, \mathcal{E}_{\text{lc}}, \mathcal{E}_{\text{uc}}, \mathcal{E}_{w})$ as its only input and generates as its output an equivalent dispatchable ESTNU having a minimal number of edges. It has four steps: (1) compute the set of so-called *stand-in* edges (i.e., ordinary edges that are entailed by various combinations of ESTNU edges) and insert them into the graph; (2) apply an STN-dispatchability algorithm to the resulting set of ordinary edges, thereby generating a dispatchable STN subgraph; (3) remove any remaining stand-in edges; and (4) remove any wait edges that are not needed for dispatchability. The $O(kn^3)$ worst-case time complexity of the `minDisp`$_{\text{ESTNU}}$ algorithm is dominated by Step 1. Therefore, our new, faster algorithm modifies only that step, achieving an order-of-magnitude reduction in the overall worst-case time complexity. The following paragraphs summarize Step 1 of the `minDisp`$_{\text{ESTNU}}$ algorithm, as implemented by its `genStandIns` helper algorithm.

Generating Stand-in Edges

Following Morris [11], an ESTNU is dispatchable if all of its STN projections are dispatchable (as STNs). Equivalently, in each STN projection, each pair of timepoints V and W that are connected by a path must be connected by a *shortest vee-path* (SVP) (i.e., a shortest path comprising zero or more negative edges followed by zero or more non-negative edges) [12]. A key insight behind the `minDisp`$_{\text{ESTNU}}$ algorithm is that in different projections, the shortest vee-paths from V to W may take different routes, employ different labeled edges, and have different lengths. The *longest* SVP from V to W across all projections determines an *ordinary* constraint, represented by a stand-in edge, that must be satisfied by every valid execution strategy. The `minDisp`$_{\text{ESTNU}}$ algorithm generates stand-in edges in two phases: (1) those entailed by individual labeled edges; and (2) those entailed by *VACW diamond* structures.

Stand-in edges entailed by individual labeled edges. Each LC, UC or wait edge entails a (weaker) ordinary edge. For example, consider the labeled edges associated with the contingent link $(A, 1, 10, C)$ in Figure 1. The LC edge $(A, c{:}1, 10)$ represents the possibility that the duration $C - A$ might take on its minimum value 1. Its stand-in edge $(A, 10, C)$ represents the (modeled) certainty that $C - A$ will be at most 10. Similarly, the UC edge $(C, C{:}{-}10, A)$ represents the possibility that $C - A$ might take on its maximum value 10, while its stand-in edge $(C, -1, A)$ represents the certainty that $C - A$ will be at least 1.



(a) ESTNU edges. **(b)** If $C - A = 1$. **(c)** If $C - A = 10$. **(d)** Stand-in edges. **(e)** General case.

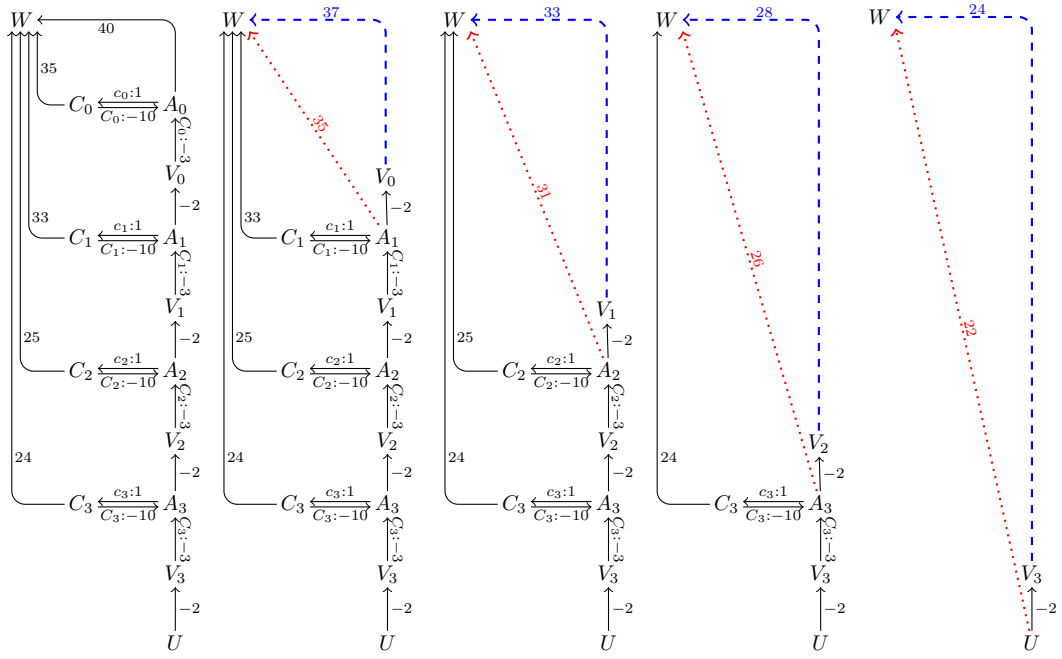
Figure 2 (Dashed) stand-in edges entailed by a *VACW* diamond structure.

Finally, the wait edge $(V, C{:}{-}6, A)$ represents the conditional constraint that, as long as C remains unexecuted, V must wait until 6 after A. Its stand-in edge $(V, -1, A)$ represents that V must unconditionally wait at least 1 after A, since C cannot execute before then.

More generally, for any contingent link (A, x, y, C), the LC edge $(A, c{:}x, C)$ entails the stand-in edge (A, y, C); the UC edge $(C, C{:}{-}y, A)$ entails the stand-in edge $(C, -x, A)$; and any wait edge $(V, C{:}{-}v, A)$ entails the stand-in edge $(V, -x, A)$, as seen in Figure 1.

Stand-in edges entailed by *VACW* diamond structures. The $\texttt{minDisp}_{\text{ESTNU}}$ algorithm uses its `genStandIns` helper algorithm to compute stand-in edges arising from diamond structures. Figure 2a shows a typical *VACW* diamond, which involves the LC and UC edges associated with a contingent link $(A, 1, 10, C)$, a wait edge $(V, C{:}{-}6, A)$, and some ordinary edges aimed at a timepoint W. Figure 2b shows that in the projection where $C - A = 1$, the shortest path from V to W has length 8, and the shortest path from V to C has length 0. Figure 2c shows that in the projection where $C - A = 10$, the shortest path from V to W has length 7, and the shortest path from V to C has length 4. Figure 2d introduces (dashed) *stand-in* edges to reflect that, across all projections, where $C - A \in [1, 10]$, the shortest path from V to W has length *at most* 8, while the shortest path from V to C has length *at most* 4. These stand-in edges represent ordinary constraints that must be satisfied by any valid dynamic execution strategy. Figure 2e shows the general case where the stand-in edge from V to W has length $\theta = \max\{\delta - v, \gamma\}$, and the stand-in edge from V to C has length $y - v$, the latter being termed an application of the *VAC rule* [9].

Stand-in edges entailed by nested diamonds. The main focus of `genStandIns` is on computing stand-in edges entailed by individual *VACW* diamond structures. But diamond structures can also be *nested.* In particular, in any *VACW* diamond, the subpath from A to W may contain a stand-in edge derived from a nested diamond. However, because the activation timepoints appearing in a nested diamond structure are subject to a strict order (as shown elsewhere [9]), diamonds can only be nested to a maximum depth of k. For this reason, the `genStandIns` algorithm does up to k iterations, each addressing one level of potential nesting. Each iteration of `genStandIns` involves two steps: (1) exploring $O(kn^2)$ individual *VACW* diamonds (k choices for the contingent link, and n choices for both V and W); and then (2) calling Johnson's algorithm [2] to update the APSP distance matrix to accommodate stand-in edges generated by the first step. Figure 3 shows how `genStandIns` deals with a sample quadruply nested diamond structure. The innermost diamond, $V_0\, A_0\, C_0\, W$, is explored during the first iteration, yielding the blue, dashed stand-in edge $(V_0, 37, W)$, shown



(a) Initial. **(b)** Iteration 1. **(c)** Iteration 2. **(d)** Iteration 3. **(e)** Iteration 4.

Figure 3 How `genStandIns` processes nested diamonds, where stand-in edges derived from individual *VACW* structures are shown in blue, and those computed by Johnson's algorithm in red.

in Figure 3b, where $\theta_0 = \max\{40 - 3, 35\} = 37$.[1] Johnson's algorithm then updates the APSP distance matrix, setting $d(A_1, W) = 35$, indicated by the red, dotted line in Figure 3b. The next iteration considers $V_1\, A_1\, C_1\, W$, which uses the new subpath from A_1 to W of length 35 to generate the blue, dashed stand-in edge $(V_1, 33, W)$, shown in Figure 3c, where $\theta_1 = \max\{35 - 3, 33\} = 33$. Johnson's algorithm then updates $d(A_2, W)$ to 31, indicated by the red, dotted line in Figure 3c. The third iteration generates the blue, dashed stand-in edge $(V_2, 28, W)$, since $\theta_2 = \max\{31 - 3, 25\} = 28$; and the red, dotted line from A_3 to W indicates the subsequent update $d(A_3, W) = 26$. Finally, as shown in Figure 3d, the last iteration generates the blue, dashed stand-in edge $(V_3, 24, W)$, since $\theta_3 = \max\{26 - 3, 24\} = 24$; while the red, dotted line from U to W indicates the update $d(U, W) = 22$.

The complexity of $\texttt{minDisp}_{\texttt{ESTNU}}$ is driven by the $O(kn^3)$-time complexity of `genStandIns`, which derives from its up to k calls of Johnson's algorithm on up to $O(n^2)$ edges.

2.1 Canonical Form of Nested Diamond Structures

The authors presented a novel, rigorous theory of the *canonical form of nested diamond structures* [9] that provides a foundation for understanding the dispatchability of ESTNUs and formally proving the correctness of the $\texttt{minDisp}_{\texttt{ESTNU}}$ algorithm. It also highlights features of such structures that suggest new approaches to solving the MinDispESTNU problem.

[1] As seen in Figure 1, each labeled edge itself entails a corresponding stand-in edge, not shown in Figure 3. Those stand-in edges ensure that there are ordinary subpaths from each A_i to W, and from each C_i to W, which implies that all of the *VACW* diamonds in Figure 3 would be processed during each iteration of `genStandIns`. However, the stand-in edges shown in Figure 3 are the strongest ones.

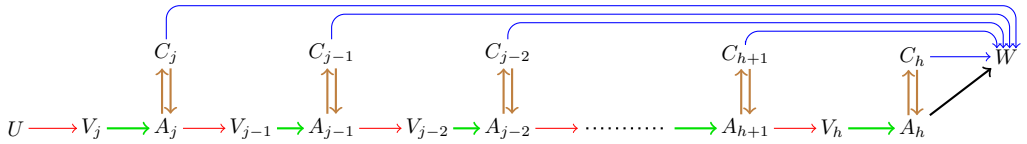


Figure 4 Canonical form of a nested diamond structure $\mathcal{S}_{uw}$ (contingent links in brown, waits in green, negative edges in red, non-negative edges in blue, and an ordinary vee-path in black).

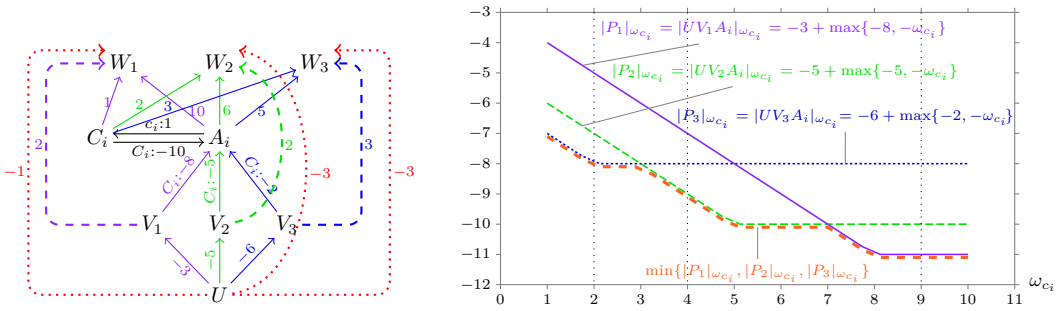


Figure 5 Three *negOrdWait* paths from U to A_i (in purple, green and blue) that determine the values of $d_*(U, W_1)$, $d_*(U, W_2)$ and $d_*(U, W_3)$, indicated by red dotted arrows. Stand-in edges are dashed. Other stand-in edges (e.g., from V_1 to W_2) are not shown.

Central to any such algorithm is computing, for each pair of timepoints U and W, the strongest ordinary constraint entailed by ESTNU paths from U to W, notated as $d_*(U, W)$. For the ESTNU in Figure 3, $d_*(U, W) = 22$ (cf. the red dotted line in Figure 3e). The theory confirms that each value $d_*(U, W)$ that derives from nested diamonds must have an associated structure, notated as $\mathcal{S}_{uw}$, whose form is illustrated in Figure 4. In particular, $\mathcal{S}_{uw}$ comprises a sequence of contingent links, shown in brown, connected by different kinds of paths. From each contingent timepoint C_i, there is a path of non-negative ordinary edges from C_i to W, shown in blue. Between consecutive pairs of activation timepoints A_f and A_g there is a *negOrdWait* path (i.e., a path comprising zero or more negative ordinary edges, shown in red, followed by a single wait edge, shown in green). There is also a *negOrdWait* path from U to the leftmost activation timepoint A_j. Finally, the path from the rightmost activation timepoint A_h to W is an ordinary path, shown in black, that is a shortest vee-path (SVP). The path from U to W that passes through all of the activation timepoints is called the *spine* of the structure. For this paper, the following properties are particularly important:

- In the situation/projection where each contingent duration along the spine satisfies $C_i - A_i = \delta_i - \gamma_i = d_*(A_i, W) - d(C_i, W)$, the length of the spine is $d_*(U, W)$.
- The *negOrdWait* paths between consecutive pairs of activation timepoints, across all canonical structures, puts the entire set of activation timepoints into a strict partial order.

2.2 Error in the `fastMinDisp`$_{\text{ESTNU}}$ Algorithm

Recent work [8] presented an algorithm, called `fastMinDisp`$_{\text{ESTNU}}$, that aimed to take advantage of certain features of nested diamonds. In particular, it exploited the fact that activation timepoints participating in nested diamonds fall into a strict partial order. That enabled processing them in a single iteration, instead of the k iterations in the `minDisp`$_{\text{ESTNU}}$ algorithm. Unfortunately, that work made an incorrect assumption. Although it is true that for any given canonical structure it suffices to include only one wait edge terminating at each activation timepoint along the spine, it is **not** the case that *all* of the canonical structures

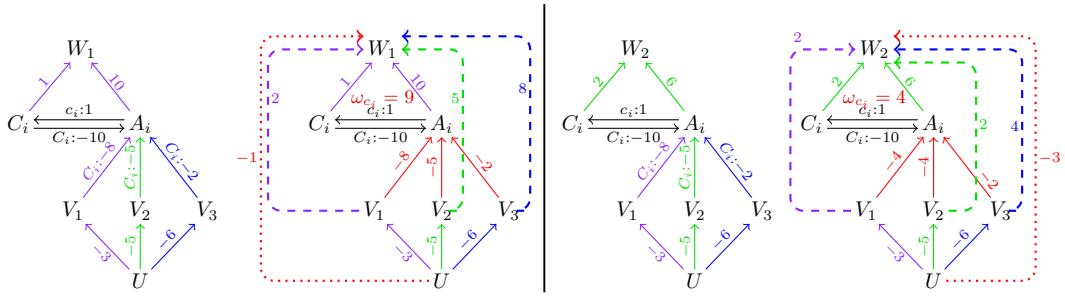


Figure 6 Computing $d_*(U, W_1)$ (left) and $d_*(U, W_2)$ (right) by back-propagation in the OW-graph.

that include some activation timepoint A_i necessarily employ the *same* wait edge terminating at A_i. Instead, as illustrated in Figure 5, different wait edges terminating at A_i may be needed in different canonical structures. In the figure, there are three overlapping canonical structures that each use the contingent link $(A_i, 1, 10, C_i)$: one from U to W_1 (in purple), one from U to W_2 (in green), and one from U to W_3 (in blue). For $d_*(U, W_1)$, the projection where $C_i - A_i = d_*(A_i, W_i) - d(C_i, W_1) = 10 - 1 = 9$ is determinative; and in that projection the shortest path from U to W_1 is through V_1 with length $d_*(U, W_1) = -1$, indicated by the red dotted arrow. The dashed, purple stand-in edge $(V_1, 2, W_1)$ has length 2, since the wait edge $(V_1, C_i{:}-8, A_i)$ has length -8 in that projection. For $d_*(U, W_2)$, the projection where $C_i - A_i = 6 - 2 = 4$ is determinative; and in that projection, the shortest path from U to W_2 is through V_2 with length $d_*(U, W_2) = -3$. The green, dashed stand-in edge $(V_2, 2, W_2)$ has length 2, since the wait edge $(V_2, C_i{:}-5, A_i)$ has length -4 in that projection. For $d_*(U, W_3)$, the projection where $C_i - A_i = 5 - 3 = 2$ is determinative; and in that projection, the shortest path from U to W_3 is through V_3 with length $d_*(U, W_3) = -3$. The blue, dashed stand-in edge $(V_3, 3, W_3)$ has length 3, since the wait edge $(V_3, C_i{:}-2, A_i)$ has length -2 in that projection. The righthand side of Figure 5 plots the lengths of the three paths from U to A_i as functions of the contingent duration $\omega_{c_i} = C_i - A_i$. It confirms that for different values of ω_{c_i}, different paths are shortest between U and A_i. As a result, each $d_*(U, W_f)$ value is based on a different path from U to A_i.

In general, for each terminus W_f, the value $d_*(U, W_f)$ is determined by the projection where $C_i - A_i = d_*(A_i, W_f) - d(C_i, W_f)$. Since these durations/projections may be different for different W_f, the wait edges terminating at A_i may provide different shortest vee-paths in different projections. Although this example shows that the `fastMinDisp`$_{\text{ESTNU}}$ algorithm does not necessarily solve the MinDispESTNU problem, it also suggests an alternative way to approach the computation of $d_*(U, W_f)$ values that results in a more efficient (and correct) algorithm for solving the MinDispESTNU problem, which is the subject of the next section.

3 A New Approach to Generating Stand-in Edges

Figure 6 illustrates our new approach to efficiently generating stand-in edges derived from nested diamond structures. It uses the following feature of the canonical form of nested diamonds: in the situation where the duration of each participating contingent link (A_i, x_i, y_i, C_i) is given by $C_i - A_i = \delta_i - \gamma_i = d_*(A_i, W) - d(C_i, W)$, the length of the path from U to W along the spine of the canonical structure equals $d_*(U, W)$. Crucially, these durations are *fixed* for a given W. Therefore, the problem of activation timepoints, A_j and A_i, that are consecutive in multiple overlapping canonical structures employing different wait edges in

different structures, can effectively be sidestepped by computing all of the $d_*(U, W)$ values for a fixed W. To do so, our new algorithm *backtracks* from W along shortest paths in the OW-graph (i.e., the graph comprising the ordinary and wait edges from the ESTNU) where wait edges, as they are encountered, are projected using the above-mentioned durations.

On the left of the figure, backtracking from W_1 encounters the activation timepoint A_i, where $d_*(A_i, W_1) = 10$ and $d(C_i, W_1) = 1$, where the determinative duration is $\omega_{c_i} = 10 - 1 = 9$. In this situation, the wait edges terminating at A_i project onto the red edges shown in the middle-left of the figure. In this projection, the path $UV_1A_iW_1$ is shortest, with a length of $-3-8+10 = -1$, indicated by the red, dotted arrow. The corresponding stand-in edge from V_1 to W_1 is shown as dashed and purple. The dashed stand-in edges emanating from V_2 (green) and V_3 (blue) are also generated, but do not contribute to $d_*(U, W_1)$.

On the righthand side of the figure, backtracking from W_2 encounters A_i and yields the duration $\omega_{c_i} = 6 - 2 = 4$. In this situation, the wait edges project to the red edges shown on the far right. Although each wait edge generates a stand-in edge, the one from V_2 to W_2 provides the shortest path (dotted, red) from U to W_2, which determines $d_*(U, W_2) = -3$.

Pseudocode for our new algorithm for generating stand-in edges entailed by nested diamond structures is given as Algorithm 1. (Appendix A provides pseudocode for all $\texttt{minDisp}_{\texttt{ESTNU}}$ procedures updated to use Algorithm 1.) Algorithm 1 works as follows.

Initialization (Lines 1–3). The `getInitStandins` algorithm (a helper for $\texttt{minDisp}_{\texttt{ESTNU}}$) is called to generate stand-in edges entailed by individual labeled edges (cf. Figure 1) or from applications of the VAC rule (cf. Figure 2e). Next, the Bellman-Ford algorithm [2] is called to compute a solution to the STN, $\mathcal{G}_{ow}$, that comprises the ordinary and wait edges from $\mathcal{G}$, ignoring any alphabetic labels. That solution, f, is then used as a potential function to re-weight the edges in $\mathcal{G}_{ow}$ to have non-negative values, thereby enabling the use of Dijkstra's algorithm [2] to guide the subsequent back-tracking from each W. Finally, Johnson's algorithm [2] is used to compute the initial distance matrix for ordinary paths.

Main foreach Loop (Lines 4–30). Each iteration of the main **foreach** loop processes a single timepoint W. It uses a modified version of Dijkstra's algorithm to back-propagate from W through the edges in the $\mathcal{G}_{ow}$ graph, aiming to update the distance function d so that by the end of the iteration, for each timepoint T, $d(T, W) = d_*(T, W)$, and all needed stand-in edges terminating at W have been generated.

Iteration initialization (Lines 5–8). First, a minimum priority queue, $\mathcal{Q}$, is initialized. For each timepoint T in the queue, its priority is the current estimate of $d_*(T, W)$, re-weighted by the potential function f. In particular, the priority of T is given by: $f(T) + \delta_{tw} - f(W)$. Initially, the queue contains only W, with a priority of 0. The n-vector, *priority* enables anytime access to the priorities of timepoints in the queue.

Next, a set *needStandIn2W* is initialized. It is used to keep track of timepoints T for which a stand-in edge from T to W will need to be generated. If the current estimate of $d_*(T, W)$ derives from a path (1) that forms the spine of a canonical diamond structure; and (2) whose first edge is a wait edge, then T is added to *needStandIn2W*, at Line 26. However, should subsequent propagation discover a shortest path from T to W for which no stand-in edge is needed, then T is removed from *needStandIn2W*, at Line 16. Since the status of a given timepoint T may change during the algorithm, stand-in edges are not actually accumulated until the end of the iteration, at Lines 28–30.

Iteration Body (Lines 9–30). The body of each iteration is a **while** loop that carries out the back-propagation from W. At Line 10, a timepoint T is extracted from the queue, along with its priority δ^*_{tw}. At Line 11, the value of $d_*(T, W)$ is extracted from δ^*_{tw} by

Algorithm 1 `betterGenStandIns`: Better Algorithm for Generating Stand-in Edges Entailed by Nested Diamonds.

Input: $\mathcal{G} = (\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{\text{lc}} \cup \mathcal{E}_{\text{uc}} \cup \mathcal{E}_{w})$, a dispatchable ESTNU graph
Output: $(\mathcal{E}_{si}, d)$, where $\mathcal{E}_{si}$ is a set of stand-in edges; and d is the updated distance matrix

1 $\mathcal{E}_{si} :=$ `getInitStandins`$(\mathcal{G})$ *//Stand-in edges entailed by individual labeled edges and VAC rule*
2 $f :=$ `bellmanFord`$(\mathcal{G}_{ow})$ *//Potential function for OW-graph, $\mathcal{G}_{ow} = (\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{si} \cup \mathcal{E}_{w})$*
3 $d :=$ `johnson`$((\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{si}))$ *//Compute the APSP distance matrix for $(\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{si})$*
4 **foreach** $W \in \mathcal{T}$ **do**
//Init min priority queue, where priority(T) = current estimate of $d_*(T, W)$ re-weighted by f
5 $\mathcal{Q} :=$ new min priority queue
6 $priority := (\infty, \ldots, \infty)$ *//For tracking priority of timepoints in the queue*
7 $\mathcal{Q}.insert(W, 0)$; $priority[W] := 0$
8 $needStandIn2W := \emptyset$ *//$T \in needStandIn2W$ means "need stand-in edge from T to W"*
9 **while** $\neg\mathcal{Q}.empty()$ **do**
10 $(T, \delta^*_{tw}) := \mathcal{Q}.extractMin()$ *//$\delta^*_{tw} = d_*(T, W)$ reweighted by potential function f*
11 $\delta_{tw} := -f(T) + \delta^*_{tw} + f(W)$ *//$\delta_{tw} = d_*(T, W)$ (un-reweighted)*
12 $d(T, W) := \delta_{tw}$ *//Update distance matrix*
13 **foreach** $(R, \delta_{rt}, T) \in (\mathcal{E}_{\text{o}} \cup \mathcal{E}_{si})$ **do** *//Back-propagate along ordinary edges*
14 $\delta^*_{rw} := (f(R) + \delta_{rt} - f(T)) + \delta^*_{tw}$ *//Possible new estimate of $d_*(R, W)$ (re-weighted)*
15 **if** $\delta^*_{rw} < priority[R]$ **then** *//New estimate of $d_*(R, W)$ shorter*
16 $needStandIn2W := needStandIn2W \setminus \{R\}$ *//No stand-in edge RW needed*
17 $\mathcal{Q}.insertOrDecreaseKey(R, \delta^*_{rw})$; $priority[R] := \delta^*_{rw}$
18 **if** $T = A$ *is an activation timepoint for a contingent link* (A, x, y, C) **then**
19 $\omega_c := d(A, W) - d(C, W)$ *//ω_c = contingent duration that determines d^* values*
20 **if** $\omega_c \in (x, y]$ **then** *//Condition for generating a non-redundant stand-in edge*
21 **foreach** $(V, C{:}{-}v, A) \in \mathcal{E}_{w}$ **do** *//Back-propagate along incoming wait edges*
22 $v_{\omega_c} := \max\{-\omega_c, -v\}$ *//Length of wait edge in projection ω_c*
23 $v^*_{\omega_c} := f(V) + v_{\omega_c} - f(A)$ *//Re-weighted length in projection ω_c*
24 $\delta^*_{vw} := v^*_{\omega_c} + priority[A]$ *//δ^*_{vw} = possible new estimate of $d_*(V, W)$*
25 **if** $\delta^*_{vw} \leq priority[V]$ **then**
26 $needStandIn2W := needStandIn2W \cup \{V\}$ *//Need stand-in edge VW*
27 $\mathcal{Q}.insertOrDecreaseKey(V, \delta^*_{vw})$; $priority[V] := \delta^*_{vw}$
28 **foreach** $T \in needStandIn2W$ **do**
29 $\delta_{tw} := -f(T) + priority[T] + f(W)$ *//Actual value of $d_*(T, W)$*
30 $\mathcal{E}_{si} := \mathcal{E}_{si} \cup \{(T, \delta_{tw}, W)\}$ *//Accumulate stand-in edge*
31 **return** $(\mathcal{E}_{si}, d)$

undoing the re-weighting using the potential function f. (The next section proves the invariant that when a timepoint T is popped from the queue, its priority equals $d_*(T, W)$, re-weighted by the potential function f.) That value is then used to update the distance function d, at Line 12.

Next, Lines 13–17 back-propagate along each incoming ordinary edge (R, δ_{rt}, T). First, at Line 14, a possible new estimate of $d_*(R, W)$ using a path from R to T to W, re-weighted using the potential function f, is computed and stored in δ^*_{rw}. (Note that $f(R) + \delta_{rt} - f(T)$ is the re-weighted length of the incoming edge from R to T.) If that estimate is less than the current priority of R (cf. Line15), then R is removed from *needStandIn2W* to reflect that this newly found shortest path from R to W does not begin with a wait edge and, hence, does not require a stand-in edge (cf. Line 16). At Line 17, R is inserted into the queue and its priority is updated.

Lines 18–27 carry out the back-propagation along any incoming wait edges, which can only happen if $W = A$ is an activation timepoint for a contingent link (A, x, y, C). Line 19 computes the value of the contingent duration $\omega_c = C - A = \delta - \gamma = d_*(A, W) - d(C, W)$ that determines whether any stand-in edges terminating at W can use this contingent link (cf. Figures 2e and 6). Note that the algorithm relies on the fact that $d(A, W) = d_*(A, W)$ at this point. Line 20 checks whether $\omega_c \in (x, y]$, since otherwise, as shown in Claim 10 of Hunsberger and Posenato [9], it is not necessary to back-propagate along any wait edge coming in to A (i.e., ordinary edges suffice). Line 22 computes the projection of the wait edge in the situation where $C - A = \omega_c$. Line 23 re-weights the projected length using the potential function. Line 24 computes the length of the path from V to T to W in the re-weighted graph. If that length less than or equal to the current key of V in the queue (Line 25), then V is added to *needStandIn2W* (at Line 26) to reflect that a stand-in edge should be generated; and the key for V is updated in the priority queue (at Line 27).

Finally, at the end of the iteration, stand-in edges for all of the flagged timepoints are generated at Lines 28–30.

Correctness of the `betterMinDispESTNU` Algorithm

The correctness of `betterMinDispESTNU` relies on the following properties of the canonical form of nested diamond structures that we have rigorously presented elsewhere [9]. (The claims mentioned below are from that work.) First, for each pair of timepoints U and W, there is a canonical form $\mathcal{S}_{uw}$ that determines the value $d_*(U, W)$. Furthermore, $d_*(U, W)$ equals the length of the *spine* of that structure in the situation where each $C_i - A_i = d_*(A_i, W) - d(C_i, W)$. (See the proof of Claim 8.) Second, using the same techniques as in the proof of Claim 7, we get that for a fixed W, there is a *single* situation ω that is simultaneously *maximal* for *all* $d_*(U, W)$ values (i.e., in the projection determined by ω, the length of the spine of each structure $\mathcal{S}_{uw}$ equals $d_*(U, W)$). For each contingent link (A, x, y, C) appearing in *any* canonical structure $\mathcal{S}_{uw}$ from *any* U to the fixed timepoint W, ω specifies the duration, $\omega_c = C - A = d_*(A, W) - d(C, W)$. Therefore, the `betterGenStandIns` algorithm, as it backtracks from W, can be understood as incrementally computing the durations, $\omega_c = C - A$, for each activation timepoint A that it encounters, based on the accumulated values, $d_*(A, W)$ and $d(C, W)$. It then computes the length of each incoming wait edge $(V, C{:}{-}v, A)$ in that projection (i.e., $\max\{-v, -\omega_c\}$), which is its length in the spine of any structure that uses it.

Worst-Case Time Complexity of the `betterMinDispESTNU` Algorithm

First, let $m = |\mathcal{E}_{\text{o}}|, k = |\mathcal{E}_{\text{lc}}| = |\mathcal{E}_{\text{uc}}|$ and $r = |\mathcal{E}_w| \leq nk$ be the numbers of ordinary, lower-case, upper-case, and wait edges, respectively, in the input ESTNU. Generating stand-in edges for individual labeled edges along with those derived from the VAC rule add $2k+2r$ more ordinary edges. Afterward, `betterMinDispESTNU` is applied to the OW-graph which has $m + 2k + 3r$ edges. For each timepoint W, `betterMinDispESTNU` uses a Dijkstra-like back-propagation that runs in $O((m + 2k + 3r + nk) + n \log n)$ time. (At most nk additional stand-in edges can be added during the course of the algorithm.) Therefore, its n iterations can be done in $O((m + 2k + 3r + nk)n + n^2 \log n)$ time, which reduces to $O(mn + n^2k + n^2 \log n)$. For dense graphs, where $m = O(n^2)$, this reduces to $O(n^3)$, but for sparse graphs, for example, where $m = O(n \log n)$ and $k = O(\log n)$, it reduces to $O(n^2 \log n)$.

4 Empirical Evaluations

We implemented the `betterMinDispESTNU` algorithm containing the procedure Algorithm 1 in Java – publicly available as part of the CSTNU Tool framework [18] – and evaluated its performance using the STNU benchmark published by Posenato [17]. This benchmark was created using the STNU random generator of the CSTNU Tool framework. The public benchmark comprises 1000 instances, all having the same topology, the *worker-lanes* topology, which simulates the worker lanes of business process modeling [16]. In this topology, the set of contingent links is divided into five lanes, with each lane representing a sequence of tasks that must be executed by an agent. The contingent links within each lane are interspersed with ordinary constraints that specify delays between the end of one task and the start of the next. Additionally, there are extra constraints between nodes in different lanes to represent temporal-coordination constraints among tasks executed by different agents.

For each possible number of nodes $n \in \{500, 1000, 1500, 2000, 2500\}$, the benchmark contains 200 DC instances and 200 non-DC instances, each having $k = n/10$ contingent links and, on average, $6.56n - 2.56k - 10$ edges (i.e., $O(n)$ edges). We considered the first 30 instances for each value of n in the benchmark.

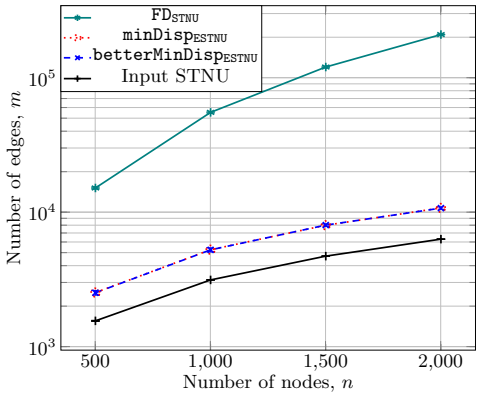
All of the experiments were executed on an OpenJDK JVM 21 configured with 16 GB of heap memory (parameters `-Xmx16G` and `-Xms16G`), on a Linux computer equipped with two AMD Opteron™ 4334 processors running at 3.1 GHz (6200 BogoMIPS) and 64 GB RAM.

Each DC STNU $\mathcal{G}$ was first pre-processed by the $\texttt{FD}_{\texttt{STNU}}$ dispatchability algorithm to generate an equivalent dispatchable ESTNU, $\mathcal{G}_{\text{fd}}$. Then, the dispatchable ESTNU, $\mathcal{G}_{\text{fd}}$, was fed as input to $\texttt{minDisp}_{\texttt{ESTNU}}$ and $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ to generate equivalent dispatchable ESTNUs having minimal numbers of edges (called μESTNUs) to: (1) confirm that the output μESTNUs were identical; and (2) compare the average execution times.

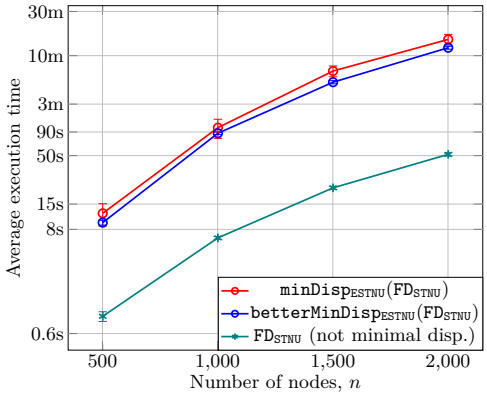
Surprisingly, during the execution of $\texttt{minDisp}_{\texttt{ESTNU}}$, we observed that *no instances* from the considered benchmarks contain any nested diamond structures. Consequently, there were no opportunities for the $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ algorithm to outperform $\texttt{minDisp}_{\texttt{ESTNU}}$.

Figure 7a shows the average numbers of edges in the input STNUs (black), the dispatchable ESTNUs generated $\texttt{FD}_{\texttt{STNU}}$ (teal), and the minimal dispatchable ESTNUs produced by $\texttt{minDisp}_{\texttt{ESTNU}}$ (dotted red) and $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ (dashed blue). (The dotted red and dashed blue lines in the figure are completely overlapping and, hence, difficult to distinguish.) The error bars denote 95% confidence intervals, which are scarcely visible due to the minimal standard deviations. The findings reveal that the average numbers of edges in the minimized networks are approximately one order of magnitude smaller than in the ESTNUs generated by $\texttt{FD}_{\texttt{STNU}}$. Since the numbers of edges in dispatchable networks directly impact the performance of real-time execution algorithms, these results demonstrate that $\texttt{minDisp}_{\texttt{ESTNU}}$ and $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ generate dispatchable networks that can be more efficiently executed. We also confirmed that they output the same minimal networks.

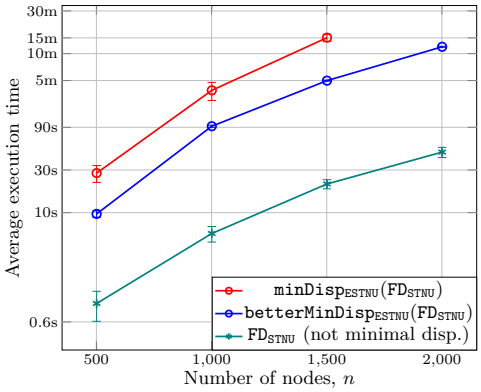
Figure 7b plots the computational cost associated with generating μESTNUs. The lower teal line shows the average execution times for $\texttt{FD}_{\texttt{STNU}}$ to generate equivalent dispatchable networks that are typically not μESTNUs. The upper two (red and blue) lines show the average execution times for generating equivalent dispatchable networks having minimal numbers of edges, obtained by applying $\texttt{minDisp}_{\texttt{ESTNU}}$ or $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ to $\mathcal{G}_{\text{fd}}$. As expected, if there are no nested diamond structures, then both algorithms will have essentially equivalent performance since they both end up doing two calls to Johnson's algorithm (or a Johnson-like algorithm).



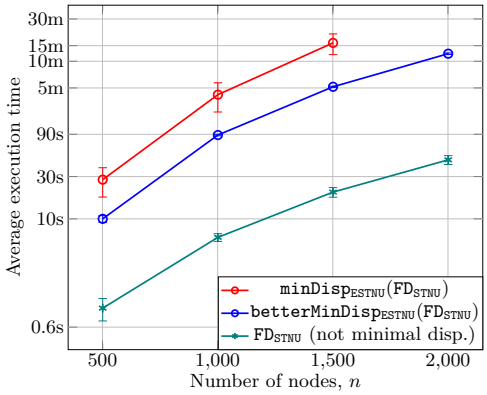
(a) Number of edges in the ESTNUs generated by $\texttt{FD}_{\texttt{STNU}}$, $\texttt{minDisp}_{\texttt{ESTNU}}$ and $\texttt{betterMinDisp}_{\texttt{ESTNU}}$.



(b) $\texttt{minDisp}_{\texttt{ESTNU}}$ and $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ performance versus network size.



(c) $\texttt{minDisp}_{\texttt{ESTNU}}$ and $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ performance versus network size for instances containing a depth-4 nested diamond structure (cf. Figure 3a).



(d) $\texttt{minDisp}_{\texttt{ESTNU}}$ and $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ performance versus network size for instances containing a depth-6 nested diamond structure.

Figure 7 Results of the empirical evaluation of the `betterMinDispESTNU` algorithm.

To assess the impact of nested diamond structures on the performance of the two algorithms, we created two new benchmarks, one comprising random STNU instances that each contain one copy of the depth-4 nested diamond structure depicted in Figure 3a, the other similar to the first, but where the diamond structure has depth 6.

The presence of the depth-4 nested diamond structure in each instance requires the `genStandIns` helper algorithm used by $\texttt{minDisp}_{\texttt{ESTNU}}$ to perform up to five iterations, each taking $O(mn + n^2 \log n)$ time, to generate the appropriate stand-in edges. In contrast, $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ replaces `genStandIns` with Algorithm 1 (`betterGenStandIns`) whose worst-case time complexity is only $O(mn+n^2k+n^2 \log n)$, regardless of how deeply nested the diamond structure may be. We therefore expected to see an especially pronounced difference in average execution times for instances having the depth-6 nested diamond structure.

The results are presented in Figures 7c and 7d. The execution time of $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ ($\texttt{FD}_{\texttt{STNU}}$) (in blue) is significantly less than that of $\texttt{minDisp}_{\texttt{ESTNU}}$ ($\texttt{FD}_{\texttt{STNU}}$) (in red) across all instances. In addition, for instances having 2000 nodes, the execution time of $\texttt{minDisp}_{\texttt{ESTNU}}$ ($\texttt{FD}_{\texttt{STNU}}$) exceeded the 30-minute timeout. Such results confirm that the $\texttt{betterMinDisp}_{\texttt{ESTNU}}$ algorithm is significantly more efficient than the $\texttt{minDisp}_{\texttt{ESTNU}}$ algorithm when the input instances contain nested diamond structures, even when the number of nested diamonds is small. Regarding the depth-6 nested diamond structure, we discovered that, on average, the

presence of random constraints among nodes in different lanes and those in the diamond structure sometimes entailed stronger constraints than the stand-in edges associated with the diamond structure and, therefore, the `genStandIns` helper for $\texttt{minDisp}_{\texttt{ESTNU}}$ performs on average five internal iterations, the same as for instances having the quadruply-nested diamond structure.

5 Conclusions

Generating an equivalent dispatchable ESTNU having a minimal number of edges is an important problem for applications involving actions with uncertain, but bounded durations. The number of edges in the dispatchable network is important because it directly impacts the real-time computations required during execution. Therefore, for time-sensitive applications it is important to generate an equivalent dispatchable ESTNU having a minimal number of edges, which we call a μESTNU. This paper modified the only existing algorithm for generating μESTNUs, making it an order-of-magnitude faster. It also showed that a second previously presented algorithm does not in fact solve the MinDispESTNU problem. The new algorithm, `betterMinDispESTNU`, reduced the worst-case time-complexity from $O(kn^3)$ to $O(mn + n^2k + n^2 \log n)$ which, for sparse networks, reduces to $O(n^2 \log n)$.

References

1. Massimo Cairo, Luke Hunsberger, and Romeo Rizzi. Faster Dynamic Controllablity Checking for Simple Temporal Networks with Uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME-2018)*, volume 120 of *LIPIcs*, pages 8:1–8:16, 2018. doi:10.4230/LIPIcs.TIME.2018.8.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 4th Edition*. MIT Press, 2022. URL: https://mitpress.mit.edu/9780262046305/introduction-to-algorithms.
3. Rina Dechter, Itay Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.
4. Luke Hunsberger. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *16th International Symposium on Temporal Representation and Reasoning (TIME-2009)*, pages 155–162, 2009. doi:10.1109/TIME.2009.25.
5. Luke Hunsberger and Roberto Posenato. Speeding up the RUL$^-$ Dynamic-Controllability-Checking Algorithm for Simple Temporal Networks with Uncertainty. In *36th AAAI Conference on Artificial Intelligence (AAAI-22)*, volume 36-9, pages 9776–9785. AAAI Pres, 2022. doi:10.1609/aaai.v36i9.21213.
6. Luke Hunsberger and Roberto Posenato. A Faster Algorithm for Converting Simple Temporal Networks with Uncertainty into Dispatchable Form. *Information and Computation*, 293(105063):1–21, 2023. doi:10.1016/j.ic.2023.105063.
7. Luke Hunsberger and Roberto Posenato. Converting Simple Temporal Networks with Uncertainty into Minimal Equivalent Dispatchable Form. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, volume 34, pages 290–300, 2024. doi:10.1609/icaps.v34i1.31487.
8. Luke Hunsberger and Roberto Posenato. Faster Algorithm for Converting an STNU into Minimal Dispatchable Form. In *31st International Symposium on Temporal Representation and Reasoning (TIME 2024)*, volume 318 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:14, 2024. doi:10.4230/LIPIcs.TIME.2024.11.

9 Luke Hunsberger and Roberto Posenato. Canonical Form of Nested Diamond Structures. Technical Report 111/2025, Dipartimento di Informatica - Università degli Studi di Verona, May 2025. URL: https://iris.univr.it/handle/11562/1163111.

10 Paul Morris. A Structural Characterization of Temporal Dynamic Controllability. In *Principles and Practice of Constraint Programming (CP-2006)*, volume 4204, pages 375–389, 2006. doi:10.1007/11889205_28.

11 Paul Morris. Dynamic controllability and dispatchability relationships. In *Int. Conf. on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-2014)*, volume 8451 of *LNCS*, pages 464–479. Springer, 2014. doi:10.1007/978-3-319-07046-9_33.

12 Paul Morris. The Mathematics of Dispatchability Revisited. In *26th International Conference on Automated Planning and Scheduling (ICAPS-2016)*, pages 244–252, 2016. doi:10.1609/icaps.v26i1.13739.

13 Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001)*, volume 1, pages 494–499, 2001. URL: https://www.ijcai.org/Proceedings/01/IJCAI-2001-e.pdf.

14 Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 1193–1198, 2005. URL: https://www.aaai.org/Papers/AAAI/2005/AAAI05-189.pdf.

15 Nicola Muscettola, Paul H. Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, KR'98, pages 444–452, 1998.

16 Object Management Group (OMG). Business process definition metamodel (bpdm), Beta 1. http://www.omg.org, 2007.

17 Roberto Posenato. STNU Benchmark version 2020, 2020. URL: https://profs.scienze.univr.it/~posenato/software/cstnu/benchmarkWrapper.

18 Roberto Posenato. CSTNU Tool: A Java library for checking temporal networks. *SoftwareX*, 17:100905, 2022. doi:10.1016/j.softx.2021.100905.

19 Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris. Fast Transformation of Temporal Plans for Efficient Execution. In *15th National Conf. on Artificial Intelligence (AAAI-1998)*, pages 254–261, 1998. URL: https://cdn.aaai.org/AAAI/1998/AAAI98-035.pdf.

A Pseudocode

Algorithm 2 `betterMinDispESTNU`: Solving the MinDispESTNU problem.

Input: $\mathcal{G} = (\mathcal{T},\ \mathcal{E}_{\rm o} \cup \mathcal{E}_{\rm lc} \cup \mathcal{E}_{\rm uc} \cup \mathcal{E}_{\rm ucg})$, dispatchable ESTNU
Output: A μESTNU for $\mathcal{G}$

//Compute the set of (ordinary) stand-in edges

1 $(\mathcal{E}_o^{\rm si}, d) :=$ `betterGenStandIns`$(\mathcal{T},\ \mathcal{E}_{\rm o} \cup \mathcal{E}_{\rm lc} \cup \mathcal{E}_{\rm uc} \cup \mathcal{E}_{\rm ucg})$

//STN dispatchability on ordinary edges, reorienting labeled edges

2 $(\mathcal{T}, \mathcal{E}_{\rm o}^*, \hat{\mathcal{E}}_{\rm l}, \hat{\mathcal{E}}_{\rm u}, \hat{\mathcal{E}}_{\rm ucg}) :=$ $\texttt{disp}_{\rm STN}(\mathcal{T}, \mathcal{E}_{\rm o} \cup \mathcal{E}_o^{\rm si}, \mathcal{E}_{\rm lc}, \mathcal{E}_{\rm uc}, \mathcal{E}_{\rm ucg})$

3 $\hat{\mathcal{E}}_{\rm o}^* := \mathcal{E}_{\rm o}^* \backslash \mathcal{E}_o^{\rm si}$ *//Remove any remaining stand-in edges from* $\mathcal{E}_{\rm o}^*$

4 $\hat{\mathcal{E}}_{\rm ucg} := \hat{\mathcal{E}}_{\rm ucg} \backslash$ `markWaits`$(\mathcal{T}_c, \hat{\mathcal{E}}_{\rm ucg}, d)$ *//Remove dominated waits*

5 **return** $\mathcal{G} = (\mathcal{T}, \hat{\mathcal{E}}_{\rm o}^* \cup \hat{\mathcal{E}}_{\rm l} \cup \hat{\mathcal{E}}_{\rm u} \cup \hat{\mathcal{E}}_{\rm ucg})$

Algorithm 3 `getInitStandins`: Generate stand-in edges entailed by individual labeled edges.

Input: $\mathcal{G} = (\mathcal{T}_x \cup \mathcal{T}_c,\ \mathcal{E}_{\rm o} \cup \mathcal{E}_{\rm lc} \cup \mathcal{E}_{\rm uc} \cup \mathcal{E}_{\rm ucg})$, a dispatchable ESTNU
Output: The set $\mathcal{E}_o^{\rm si}$ of ordinary stand-in edges for the individual labeled edges in $\mathcal{G}$
Side Effect: Modifies $\mathcal{G}$ by fixing any weak or misleading wait edges

$\mathcal{E}_o^{\rm si} := \emptyset$
foreach $(A, x, y, C) \in \mathcal{L}$ **do** *//Collect stand-in edges for LC, UC and wait edges*
  $\mathcal{E}_o^{\rm si} := \mathcal{E}_o^{\rm si} \cup \{(A, y, C), (C, -x, A)\}$ *//Collect stand-in edges for LC and UC edges*
  foreach $(V, C{:}{-}v, A) \in \mathcal{E}_{\rm ucg}$ **do**
    if $-v \geq -x$ **then** *//Replace weak wait edge by an ordinary edge*
      $\mathcal{E}_{\rm ucg} := \mathcal{E}_{\rm ucg} \backslash \{(V, C{:}{-}v, A)\};\ \mathcal{E}_{\rm o} := \mathcal{E}_{\rm o} \cup \{(V, -v, A)\}$
    else
      if $-v < -y$ **then** *//Fix misleading wait by adjusting its wait time*
        $\mathcal{E}_{\rm ucg} := \mathcal{E}_{\rm ucg} \setminus \{(V, C{:}{-}v, A)\} \cup \{V, C{:}{-}y, A)\}$
      //Add stand-in edges for wait edge and from the VAC rule
      $\mathcal{E}_o^{\rm si} := \mathcal{E}_o^{\rm si} \cup \{(V, -x, A), (V, \max\{y - v, 0\}, C)\}$
return $\mathcal{E}_o^{\rm si}$

Algorithm 4 `markWaits`: Mark wait edges for removal.

Input: $\mathcal{T}_c$, contingent TPs; $\hat{\mathcal{E}}_{\rm ucg}$, wait edges; d, distance fn.
Output: A set $\mathcal{E}_{\rm w}^m \subseteq \hat{\mathcal{E}}_{\rm ucg}$ of wait edges marked for removal

$\mathcal{E}_{\rm w}^m := \emptyset$
foreach $(V, C{:}{-}v, A) \in \hat{\mathcal{E}}_{\rm ucg}$ **do** *//Collect waits dominated by ordinary paths, UC edges, or other waits*
  if $d(V, A) \leq -v$ **or** $d(V, C) < 0$ **then**
    $\mathcal{E}_{\rm w}^m := \mathcal{E}_{\rm w}^m \cup \{(V, C{:}{-}v, A)\}$ *//Dominated by an ordinary path or the UC edge*
  else
    foreach $U \in \mathcal{T} \mid \exists (U, C{:}{-}u, A) \in \hat{\mathcal{E}}_{\rm ucg}$ **do**
      if $d(V, U) < 0$ **and** $d(V, U) - u \leq -v$ **then**
        $\mathcal{E}_{\rm w}^m := \mathcal{E}_{\rm w}^m \cup \{(V, C{:}{-}v, A)\}$ *//Dominated by another wait*
return $\mathcal{E}_{\rm w}^m$

On the Complexity of the Realisability Problem for Visit Events in Trajectory Sample Databases

Arthur Jansen[1]
Hasselt University, Databases and Theoretical Computer Science Group and
Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Bart Kuijpers
Hasselt University, Databases and Theoretical Computer Science Group and
Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Abstract

Trajectory sample databases store finite sequences of measured space-time locations of moving objects, along with a speed bound for each object. These databases can be seen as uncertain databases. We propose a language that allows the formulation of queries about the uncertainty in trajectory sample databases. As part of that language, we introduce the notion of *visit events*, which are used to describe certain constraints on the movement of an object. In our language, an atomic query asks whether a moving object can, given its limitations, realise such an event. We give complexity results for this realisability problem, in various settings.

2012 ACM Subject Classification Information systems → Spatial-temporal systems; Information systems → Query languages

Keywords and phrases Trajectory sample databases, uncertain databases, query languages, complexity

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.12

Funding *Arthur Jansen*: Bijzonder Onderzoeksfonds (BOF22OWB06) from UHasselt

1 Introduction

Due to the proliferation of location-aware devices (such as GPS receivers) in the past two decades, one of the use-cases of moving object databases [8] is the storage of time-stamped measured locations of moving objects [7]. Such a sequence of spatio-temporal measurements of a single moving object is called a *trajectory sample*. Given this type of partial information on a moving object, we do not know the precise space-time path (or trajectory) which the object has followed, but we do know that the trajectory must have passed these measured spatio-temporal locations. However, without making further assumptions, there are no theoretical limits to the movement of the object in between two measurements. An assumption that originates from the area of time geography, where the moving object's accessibility to an environment is studied, is that we know a bound on the speed of the moving object [5, 9, 13]. Therefore, it is common to associate a maximal speed to each moving object, alongside a trajectory sample. With this additional knowledge, the actual trajectory of a moving object is guaranteed to be contained in a spatio-temporal region known as a "lifeline necklace" in spatio-temporal and moving object databases [6, 10, 18], or simply as a "chain of space-time prisms" in the fields of time geography [9] and Geographical Information Systems (GIS) [15, 12, 14].

[1] Corresponding author

We refer to a moving object database with the specific purpose of storing trajectory samples and speed bounds as a *trajectory sample database*. A trajectory sample database can be seen as an uncertain (or incomplete) database [1, 11]. In general, an uncertain database represents a set of "possible worlds", where each possible world is a concrete instantiation of the data. In our setting, a possible world corresponds to an assignment of a trajectory to every object, satisfying the known limitations of that object. In this paper, we propose a *query language for trajectory sample databases* that is based on events that are possibly semantically interesting for some application and that may occur during a trajectory of a moving object. We introduce *visit events* as part of our language, which are used to describe constraints on trajectories. The most basic visit event expresses that a trajectory visits a particular region during a particular period of time. To allow for composition, the class of visit events is closed under Boolean combinations (using negation, conjunction and disjunction). For example, we could express the complex event that states that an object has visited a museum in the morning, some restaurant at lunch and has not been in a particular church in the afternoon. For a dataset of tourists visiting some city, this event may have occurred during the movement of some of the tourists. This example shows that our proposed query language is related to the field of *semantic trajectories* and *places of interest* (POIs) in such trajectories and could be used to match trajectories with event patterns [4, 17, 21]. The queries appearing in [20], where a cylinder model of uncertainty is used, are similar to what our languages can express.

The atomic query in our language asks whether a given visit event is *realisable* by the trajectory of some moving object, that is, whether there exists a trajectory that satisfies the constraints described by that event. We call the evaluation of this atomic query the *realisability problem*. To allow for composition on the query level, the query expressions are also closed under Boolean combinations. An important detail is that our query language is defined relative to two parameters: $\mathcal{R}$, denoting the class of spatial regions that can occur in visit events, and $\mathcal{T}$, denoting the periods of time that can occur in visit events. We study the complexity of the realisability problem for different choices of $\mathcal{R}$ and $\mathcal{T}$. For $\mathcal{R}$, we consider the class of singleton points (Point) and of semi-algebraic sets (SemiAlg). For $\mathcal{T}$, on the other hand, we consider the class of singleton moments (Moment) and intervals of time (Interval). Because the realisability problem already becomes NP-hard for very restrictive classes of events, and because we believe it is of interest to measure the influence that the size of the database has on the complexity of query evaluation, we distinguish between combined complexity, data complexity and query complexity [1]. A summary of our complexity results is displayed in Table 1.

Table 1 A summary of the complexity results: data, query, and combined complexity.

Class of events	Data	Query	Combined
(Point, Moment)-event in DNF	linear	polynomial	polynomial
Positive (Point, Moment)-events	linear	NP-hard	NP-hard
(SemiAlg, Moment)-events	linear	NP-hard	NP-hard
Conjunctive (Point, Interval)-events	polynomial	NP-hard	NP-hard
Positive (SemiAlg, Interval)-events	polynomial	NP-hard	NP-hard

The paper is organised as follows. In Section 2, we give the definitions that are necessary to formalize the notion of a trajectory sample database. In Section 3, we define the syntax and the semantics of our query languages. In Section 4, we study the complexity of the realisability problem in different settings. Finally, we conclude the paper with Section 5.

2 Definitions and preliminaries on trajectory sample databases

In this section, we give definitions of the concepts needed to introduce the notion of a *trajectory sample database*.

We consider the space in which objects move to be the plane $\mathbb{R}^2$, and we use the letters $p, q, \ldots$ (with or without indices) to denote locations in space. Time is modelled as the real line $\mathbb{R}$, and we use t (with or without indices) to refer to the temporal points (or moments). We also use x and y to refer to the real coordinates of spatial locations. A spatio-temporal point (p, t) is then an element of $\mathbb{R}^2 \times \mathbb{R}$ and if $p = (x, y)$, we also write (x, y, t) for the spatio-temporal point (p, t). Furthermore, we use d to denote the Euclidean distance in $\mathbb{R}^2$.

The movement of an object is captured by the notion of "trajectory" and it corresponds to a function mapping (all possible) moments in time to locations in space, as is expressed by the following definition.

▶ **Definition 1.** *A* trajectory *is a continuous function from* $\mathbb{R}$ *to* $\mathbb{R}^2$.

We use the Greek letter γ (with or without indices) to refer to trajectories. In practice trajectories are only observed or measured at discrete moments in time and we call these partial views on trajectories "trajectory samples".

▶ **Definition 2.** *A* trajectory sample *(or* sample*, for short) is a finite sequence of space-time points* $\langle (p_1, t_1), \ldots, (p_n, t_n) \rangle$ *that is ordered by the temporal component (that is,* $t_1 < \cdots < t_n$*).*

We use the letter S (with or without indices) to refer to trajectory samples. The following definition captures the notion of trajectories matching trajectory samples.

▶ **Definition 3.** *We say a trajectory* γ visits *a space-time point* (p, t) *when* $\gamma(t) = p$ *and we say that a trajectory* γ visits *a subset of space-time if* γ *visits one of its elements.*

A trajectory sample $\langle (p_1, t_1), \ldots, (p_n, t_n) \rangle$ matches *the trajectory* γ *if* γ *visits all the space-time points* (p_i, t_i)*, for* $i = 1, \ldots, n$.

Trajectory sample databases do not only store a trajectory sample for each moving object, but also a maximal speed for each moving object. Because we assume that every moving object has such speed bound, these bounds further restrict trajectories (as trajectory samples do). This is capured in the following definition.

▶ **Definition 4.** *A trajectory* γ *is called* $v_{\max}$*-bounded when for all* $t, t' \in \mathbb{R}$ *we have* $d(\gamma(t), \gamma(t')) \leq v_{\max} \cdot |t - t'|$.

We use $\Gamma_{(S, v_{\max})}$ to denote the set of all $v_{\max}$-bounded trajectories matching the sample S. Obviously, some trajectory samples have no $v_{\max}$-bounded trajectories that match them. The consistency of a sample with a speed bound $v_{\max}$ is expressed in the following definition.

▶ **Definition 5.** *A trajectory sample* $\langle (p_1, t_1), \ldots, (p_n, t_n) \rangle$ *is called* $v_{\max}$-consistent *when* $d(p_i, p_{i+1}) \leq v_{\max} \cdot (t_{i+1} - t_i)$*, for* $i = 1, \ldots, n-1$.

In the remainder of this paper, we assume, when given a trajectory sample S and a speed bound $v_{\max}$, that S is $v_{\max}$-consistent.

▶ **Definition 6.** *The* linear interpolation trajectory *of a trajectory sample* $S = \langle (p_1, t_1), \ldots, (p_n, t_n) \rangle$*, denoted by* $\mathsf{LIT}(S)$*, is defined as the trajectory* γ*, with*

$$\gamma(t) = \begin{cases} p_1 & \text{if } t \leq t_1 \\ \frac{t_{i+1} - t}{t_{i+1} - t_i} \cdot p_i + \frac{t - t_i}{t_{i+1} - t_i} \cdot p_{i+1} & \text{if } t_i < t \leq t_{i+1} \qquad (\text{for } 1 \leq i < n) \\ p_n & \text{if } t_n < t. \end{cases}$$

We note that if S is $v_{\max}$-consistent, then $\mathsf{LIT}(S)$ is a $v_{\max}$-bounded trajectory.

In trajectory sample databases, we use natural numbers as identifiers for moving objects. Therefore, such database is defined relative to a finite subset Obj of $\mathbb{N}$, called the *object identifiers*.

▶ **Definition 7.** *A* trajectory sample database D *over object identifiers* $\mathsf{Obj} \subset \mathbb{N}$ *is a function mapping each identifier* $i \in \mathsf{Obj}$ *to a pair* $D(i) = (S_D(i), v_D(i))$*, where* $S_D(i)$ *is a trajectory sample and* $v_D(i)$ *is a speed bound.*

This means that, given a database D, the set $\Gamma_{D(i)} = \Gamma_{(S_D(i), v_D(i))}$ contains all trajectories that the object with identifier i may have followed (given the sample and the speed bound).

3 Syntax, semantics and evaluation of $(\mathcal{R}, \mathcal{T})$-queries

In this section, we describe a family of query languages for trajectory sample databases. Our languages consists of two "tiers": in the inner tier, we have the events, which are used as subexpressions in the outer tier, where we have the query expressions. We define the language relative to two parameters, $\mathcal{R}$ and $\mathcal{T}$, where $\mathcal{R}$ is a collection of spatial regions (or subsets of $\mathbb{R}^2$), and $\mathcal{T}$ is a collection of temporal periods (or subsets of $\mathbb{R}$).

In the following subsections, we define the syntax of the query languages, their semantics, and we end with a remark on the evaluation of query expressions.

3.1 The syntax of $(\mathcal{R}, \mathcal{T})$-queries

We start by defining the inner tier of our language, which is a calculus of *events*. These events occur as subexpressions in the query expressions defined later on. Events express visits that may occur during the movement of an object.

▶ **Definition 8.** *We define the* $(\mathcal{R}, \mathcal{T})$-events *as follows:*

1. $\mathsf{visits}(R, T)$*, with* $R \in \mathcal{R}$ *and* $T \in \mathcal{T}$*, is an atomic* $(\mathcal{R}, \mathcal{T})$*-event;*
2. *If* e, e_1 *and* e_2 *are* $(\mathcal{R}, \mathcal{T})$*-events, then so are* $(\neg e)$*,* $(e_1 \wedge e_2)$ *and* $(e_1 \vee e_2)$*.*

In other words, $(\mathcal{R}, \mathcal{T})$-events can be considered as propositional formulas over the propositional symbols $\mathsf{visits}(R, T)$, for every $R \in \mathcal{R}$ and $T \in \mathcal{T}$. Now, we are ready to define $(\mathcal{R}, \mathcal{T})$-queries. Their expression is given in the following definition.

▶ **Definition 9.** *Let* Var *be a set of* object identifier variables *(or variables, for short). The* $(\mathcal{R}, \mathcal{T})$*-query expressions are defined as follows:*

1. *If* e *is an* $(\mathcal{R}, \mathcal{T})$*-event,* i *a natural number and* $x \in \mathsf{Var}$*, then* $\mathsf{realisable}(i, e)$ *and* $\mathsf{realisable}(x, e)$ *are atomic* $(\mathcal{R}, \mathcal{T})$*-query expressions;*
2. *If* q, q_1 *and* q_2 *are* $(\mathcal{R}, \mathcal{T})$*-query expression, then so are* $(\neg q)$*,* $(q_1 \wedge q_2)$ *and* $(q_1 \vee q_2)$*.*

3.2 The semantics of $(\mathcal{R}, \mathcal{T})$-queries

To define the semantics of query expressions, we first define what it means for an event to be *realised* by a trajectory.

▶ **Definition 10.** *Let* e *be an* $(\mathcal{R}, \mathcal{T})$*-event and let* γ *be a trajectory.*

1. *If* e *is the atomic event* $\mathsf{visits}(R, T)$*, then* e *is* realised *by* γ *if* $\gamma(t) \in R$ *for some* $t \in T$ *(that is,* γ *visits* $R \times T$*).*
2. *If* e *is of the form* $(\neg e_1)$*, then* e *is realised by* γ *when* e_1 *is not realised by* γ*.*
3. *If* e *is of the form* $(e_1 \wedge e_2)$*, then* e *is realised by* γ *when* e_1 *and* e_2 *are realised by* γ*.*
4. *If* e *is of the form* $(e_1 \vee e_2)$*, then* e *is realised by* γ *when* e_1 *or* e_2 *is realised by* γ*.*

In our definition of the semantics of $(\mathcal{R}, \mathcal{T})$-queries, we distinguish between Boolean and non-Boolean queries. The first type of queries contain no variables and they evaluate to true or false. The second type of queries contain variables and they define a relations over Obj.

▶ **Definition 11.** *Let D be a trajectory sample database over object identifiers Obj. We write $D \models q$ to express that a variable-free $(\mathcal{R}, \mathcal{T})$-query expression q evaluates to true on D. This relation is defined as follows:*

1. $D \models \mathsf{realisable}(i, e)$ *if* $i \in \mathsf{Obj}$ *and there exists a trajectory in* $\Gamma_{D(i)}$ *that realises* e.
2. $D \models \neg q$ *if* $D \models q$ *does not hold.*
3. $D \models q_1 \wedge q_2$ *if* $D \models q_1$ *and* $D \models q_2$.
4. $D \models q_1 \vee q_2$ *if* $D \models q_1$ *or* $D \models q_2$.

▶ **Definition 12.** *Let D be a trajectory sample database over object identifiers Obj. If q is an $(\mathcal{R}, \mathcal{T})$-query expression containing variables $x_1, \ldots, x_k \in \mathsf{Var}$, then the result of q evaluated in D is*

$$q(D) = \left\{ (i_1, \ldots, i_k) \in \mathsf{Obj}_D^k \mid D \models q[i_1/x_1, \ldots, i_k/x_k] \right\},$$

where $q[i_1/x_1, \ldots, i_k/x_k]$ is obtained from q by instantiating the variable x_j in q by i_j, for $j = 1, ..., k$.

3.3 Evaluation of $(\mathcal{R}, \mathcal{T})$-queries

Having defined the semantics of our $(\mathcal{R}, \mathcal{T})$-query languages, there is a "standard" way of evaluating query expressions with variables: given a query expression with k variables, we enumerate all k-tuples of object identifiers, consider all the instantiations associated with them and then evaluate the variable-free expression obtained by substituting the variable occurrences by the concrete object identifiers from this tuple.

However, it is not immediately clear from the definition of the semantics how an atomic query of the form $\mathsf{realisable}(i, e)$ can be evaluated. In what follows, we restrict our attention to this decision problem.

▶ **Definition 13.** *We define the* $(\mathcal{R}, \mathcal{T})$-realisability problem *to be the following decision problem: given $(\mathcal{R}, \mathcal{T})$-event e, sample S and speed bound v, does there exist a trajectory in $\Gamma_{(S,v)}$ that realises e?*

From now on, we use the notation $(S, v) \models e$ to express that there exists a trajectory in $\Gamma_{(S,v)}$ that realises the event e.

4 The complexity of the $(\mathcal{R}, \mathcal{T})$-realisability problem

In this section, we give a number of complexity results on the above realisability problem. We recall that the input to this problem is a trajectory sample, a speed bound and an $(\mathcal{R}, \mathcal{T})$-event. In order for the realisability problem to be a proper computational decision problem, these inputs must have finite representations. This means, for example, that we cannot take arbitrary real numbers as input and that we need to assume some finite encoding of our inputs. Here, we assume that the spatio-temporal points occurring in the trajectory sample have rational coordinates, and that the speed bound is rational. As for the $(\mathcal{R}, \mathcal{T})$-event, its representation depends on the choice for $\mathcal{R}$ and $\mathcal{T}$ and the representation of their elements. The choices for $\mathcal{R}$ that we consider are Point, the collection of all singletons in $\mathbb{Q}^2$; and $\mathsf{SemiAlg}$, the collection of all semi-algebraic sets in the plane. A semi-algebraic set

in the plane is a subset of $\mathbb{R}^2$ that can be defined using a Boolean combination of polynomial (in)equalities over two real variables (where the polynomials have integer coefficients) [3]. Elements of Point are simply represented by the coordinates of the point in question, while an element of SemiAlg is represented by some encoding of its defining formula. The choices for $\mathcal{T}$ we consider are Moment, containing all singletons in $\mathbb{Q}$, and Interval, containing all closed intervals of $\mathbb{R}$ with rational endpoints.

For notational convenience, the atomic $(\mathsf{Point}, \mathcal{T})$-event $\mathsf{visits}(\{p\}, T)$ will be written as $\mathsf{visits}(p, T)$. And similarly, we will write $\mathsf{visits}(R, t)$ for the atomic $(\mathcal{R}, \mathsf{Moment})$-event $\mathsf{visits}(R, \{t\})$.

As we show below, the realisability problem is already NP-hard for quite restricted classes of events. Similar to the evaluation of queries in relational databases, the hardness of the realisability problem is caused by the size of the event, and not by the size of the trajectory sample. Therefore, we study the complexity of the realisability problem in three different settings [1], being:

- the *data complexity*, where we measure the complexity in terms of the size of the sample, and consider the event to be fixed,
- the *query complexity*, where we measure the complexity in terms of the size of event, and consider the sample to be fixed, and
- the *combined complexity*, where we measure the complexity in terms of both the size of the sample, and the size of the event.

We also consider several restrictions to the class of input events, such as *positive* events, containing no negations, *conjunctive* events, being conjunctions of atoms (or, not containing disjunctions and negations), and events in disjunctive normal form (DNF).

In the remainder of this section, we give various results on the complexity of the $(\mathcal{R}, \mathcal{T})$-realisability problem, for different choices of $\mathcal{R}$ and $\mathcal{T}$ and in the three different settings.

For the complexity results mentioned below, we use a computational model in which operations (addition, multiplication, ...) and comparison relations ($=$, $<$, ...) on rational numbers are assumed to take unit time. That is, we measure the time complexity in terms of the number of spatio-temporal points in a trajectory sample and in terms of number of atoms (and their length) in an event-expression.

4.1 The query complexity of the $(\mathcal{R}, \mathcal{T})$-realisability problem

Our first result shows that the realisability problem is already NP-hard for a relatively restricted class of events, namely the positive (Point, Moment)-events.

▶ **Theorem 14.** *In terms of query complexity, the* (Point, Moment)*-realisability problem for positive events is NP-hard.*

Proof. To prove NP-hardness, we describe a reduction from SAT, the satisfiability problem of propositional formulas. Because the statement concerns query complexity, we reduce SAT to the (Point, Moment)-realisability problem with fixed sample and speed bound. In this case, we choose the sample $S = \langle((0,0),0),((0,0),1)\rangle$ and the speed bound $v = 1$. The input of the reduction is a propositional formula ϕ. We can assume that ϕ is in negation normal form[2], because the satisfiability problem remains NP-hard under this restriction. The output is a positive (Point, Moment)-event e_ϕ such that the formula ϕ is satisfiable iff there exists a trajectory γ in $\Gamma_{(S,v)}$ that realises the event e_ϕ.

[2] A formula is said to be in negation normal form if negation operators are only applied to atoms.

Let $P_1, \dots P_k$ be the propositional symbols occuring in ϕ. For $1 \leq i \leq k$, we define $t_i = \frac{i+1}{k+2}$, $p_i = (\frac{1}{2(k+2)}, 0)$ and $p'_i = (-\frac{1}{2(k+2)}, 0)$. Now, we take e_ϕ to be the result of substituting occurences of $\neg P_i$ by $\mathsf{visits}(p'_i, t_i)$ and non-negated occurrences of P_i by $\mathsf{visits}(p_i, t_i)$ in ϕ. Clearly, e_ϕ is a $(\mathsf{Point}, \mathsf{Moment})$-event, not containing negations.

Finally, we show that ϕ is satisfiable if and only if there exists a γ in $\Gamma_{(S,v)}$ that realises e_ϕ. The "if"-direction is straightforward. If there is some γ that realises e_ϕ, then ϕ must certainly be satisfiable. To be precise, the assignment that assigns P_i to true if and only if $\gamma(t_i) = p_i$ satisfies ϕ. For the "only if"-direction, assume that ϕ is satisfiable. Then there exists a truth assignment α that makes ϕ true. Now, we extend the sample S to a sample S' such that it contains (p_i, t_i) if P_i is assigned true by α, and otherwise contains (p'_i, t_i). It is easily verified that S' is v-consistent, which means $\mathsf{LIT}(S')$ is a v-bounded trajectory. Now, we have that $\mathsf{LIT}(S')$ realises $\mathsf{visits}(p_i, t_i)$ if P_i is true under α, and realises $\mathsf{visits}(p'_i, t_i)$ if $\neg P_i$ is true under α. It follows from the way we constructed the event, that $\mathsf{LIT}(S')$ realises e_ϕ. ◀

While the above result shows that the $(\mathsf{Point}, \mathsf{Moment})$-realisability problem is NP-hard for positive events (not containing negations), the problem for $(\mathsf{Point}, \mathsf{Interval})$-events already becomes NP-hard for conjunctions of atoms, as shown below.

▶ **Theorem 15.** *In terms of query complexity, the* $(\mathsf{Point}, \mathsf{Interval})$*-realisability problem for conjunctive events is NP-hard.*

Proof. We give a reduction from the Euclidean travelling saleman problem (E-TSP for short), shown to be NP-hard in [16] (the problem we refer to here is called the Euclidean tour-TSP there). The Euclidean travelling saleman problem asks, given a finite set of locations $P \subseteq \mathbb{Q}^2$ and a positive number $\ell \in \mathbb{Q}$, whether there is a cycle through all locations of P whose length is at most ℓ. Formally, this means there is a permutation $p_1, \dots, p_n$ of the locations in P such that $\sum_{i=1}^{n-1} d(p_i, p_{i+1}) + d(p_n, p_1) \leq \ell$. Without loss of generality, we assume that P always contains the origin $(0, 0)$.

Again, we work with a fixed sample $S = \langle((0,0), 0)\rangle$ and speed bound $v = 1$. From an instance P, ℓ of E-TSP, we give a conjunction of $(\mathsf{Point}, \mathsf{Interval})$-atoms C, such that $(S, v) \models C$ if and only if there is cycle through P of length at most ℓ. We define C as

$$\mathsf{visits}((0,0), [\ell, \ell]) \wedge \bigwedge_{p \in P \setminus \{(0,0)\}} \mathsf{visits}(p, [0, \ell]).$$

To prove that this reduction is correct, we first show that if $(S, v) \models C$, then there is cycle through P of length at most ℓ. Let γ be a v-bounded trajectory matching S and realising C. Because γ matches S, we have $\gamma(0) = (0, 0)$. And, because γ realises C, it reaches every location in P at some moment in $[0, \ell]$, and $\gamma(\ell) = (0, 0)$. This induces an order $p_1, \dots, p_n$ of the locations in P, where γ first reaches p_1, then p_2, and so on (we note that p_1 is always $(0, 0)$). Thus, if we let t_i be the first moment where $\gamma(t_i) = p_i$ for $i = 1, \dots, n$, then $0 = t_1 < \dots < t_n \leq \ell$. We claim the cycle $p_1, \dots, p_n, p_1$ has length at most ℓ. The length of this cycle is $\sum_{i=1}^{n-1} d(p_i, p_{i+1}) + d(p_n, p_1)$. For every i, γ visits (p_i, t_i), and because γ is v-bounded, we have $d(p_i, p_{i+1}) \leq v \cdot |t_i - t_{i+1}| = t_{i+1} - t_i$. Similarly, γ visits (p_n, t_n) and (p_1, ℓ), thus $d(p_n, p_1) \leq \ell - t_n$. It follows that the length of the cycle is at most $\sum_{i=1}^{n-1}(t_{i+1} - t_i) + \ell - t_n = t_n - t_1 + \ell - t_n = \ell$.

Finally, we show that if there is cycle through P of length at most ℓ, then $(S, v) \models C$. Let $p_1, \dots, p_n, p_1$ be such cycle, where we choose p_1 to be $(0, 0)$. Now, let $t_1 = 0$ and for $i = 2, \dots, n$, take $t_i = t_{i-1} + d(p_{i-1}, p_i)$. Then, $t_n = \sum_{i=1}^{n-1} d(p_i, p_{i+1})$, which, by assumption, is at most $\ell - d(p_n, p_1)$. Define the trajectory sample $S' = \langle(p_1, t_1), \dots, (p_n, t_n), (p_1, \ell)\rangle$. It is clear from the definition of t_i that the part of S' excluding (p_1, ℓ) is v-consistent. We have seen that $t_n \leq \ell - d(p_n, p_1)$, and thus $d(p_n, p_1) \leq \ell - t_n$, which implies that S' is v-consistent. From this follows that $\mathsf{LIT}(S')$ is v-bounded, and it clearly matches S and realises C. ◀

4.2 The data complexity of the $(\mathcal{R}, \mathcal{T})$-realisability problem

In this section, we show that the (SemiAlg, Moment)-realisability problem has linear-time data complexity, and the (SemiAlg, Interval)-realisability problem has polynomial-time data complexity

We start with the result on (SemiAlg, Moment)-queries, but first we introduce some notation and we give two lemmas. Every region in SemiAlg is of the form $\{(x, y) \in \mathbb{R}^2 \mid \varphi(x, y)\}$, where $\varphi = \varphi(x, y)$ is a quantifier-free formula over the vocabulary $(+, \times, <, 0, 1)$, with x and y as free variables. The set $\{(x, y) \in \mathbb{R}^2 \mid \varphi(x, y)\}$ is called the region defined by φ, and we denote it by $R(\varphi)$.

▶ **Definition 16.** *If $C = \mathsf{visits}(R_1, t_1) \wedge \cdots \wedge \mathsf{visits}(R_k, t_k)$ is a conjunction of atomic $(\mathcal{R}, \mathsf{Moment})$-events, and $I \subseteq \mathbb{R}$ is an interval, the formula C_I is the conjunction of those $\mathsf{visits}(R_i, t_i)$, with $1 \leq i \leq k$, for which $t_i \in I$.*

▶ **Lemma 17.** *Let C be a conjunction of atomic $(\mathcal{R}, \mathsf{Moment})$-events, let S be a trajectory sample $\langle (p_1, t_1), \ldots, (p_n, t_n) \rangle$ and let v be a speed bound. Then, $(S, v) \models C$ if and only if all of the following are true:*

(1) $(\langle (p_1, t_1) \rangle, v) \models C_{(-\infty, t_1]}$,

(2) *for $i = 1, \ldots, n-1$, we have $(\langle (p_i, t_i), (p_{i+1}, t_{i+1}) \rangle, v) \models C_{[t_i, t_{i+1}]}$, and*

(3) $(\langle (p_n, t_n) \rangle, v) \models C_{[t_n, +\infty)}$.

Proof. The "only if"-direction is obvious. We prove the "if"-direction. Assume (1), (2) and (3) are true. By (1), there exists a v-bounded trajectory γ_0 matching $\langle (p_1, t_1) \rangle$ that realises $C_{(-\infty, t_1]}$. By (2), for $i = 1, \ldots, n-1$, there exists a v-bounded trajectory γ_i matching $\langle (p_i, t_i), (p_{i+1}, t_{i+1}) \rangle$ that realises $C_{[t_i, t_{i+1}]}$. And by (3), there exists a v-bounded trajectory γ_n matching $\langle (p_n, t_n) \rangle$ that realises $C_{[t_n, +\infty)}$. We define the trajectory γ as follows:

$$\gamma(t) = \begin{cases} \gamma_0(t) & \text{if } t \in (-\infty, t_1], \\ \gamma_i(t) & \text{if } t \in [t_i, t_{i+1}] \text{ and} \\ \gamma_n(t) & \text{if } t \in [t_n, +\infty). \end{cases}$$

We note that the intervals $[t_{i-1}, t_i]$ and $[t_i, t_{i+1}]$ both contain t_i. However, this does not pose a problem for the definition of γ, because both γ_{i-1} and γ_i visit (p_i, t_i), which means $\gamma_{i-1}(t_i) = \gamma_i(t_i) = p_i$. It only requires a simple application of the triangle inequality (of Euclidean distance) to show that γ is a v-bounded trajectory. For $i = 1, \ldots, n$, we have $\gamma(t_i) = p_i$, thus γ matches S. The only thing left to prove is that γ realises C. Let $A = \mathsf{visits}(R, t)$ be an arbitrary conjunct of C. Depending on t, there are three cases to be considered. First, if $t \in (-\infty, t_1]$, then A is a conjunct of $C_{(-\infty, t_1]}$. This implies γ_0 realises A, so $\gamma(t) = \gamma_0(t) \in R$, which means γ realises A. The other two cases, when t is contained in $[t_i, t_{i+1}]$ or in $[t_n, +\infty)$, are similar. Because γ realises all the conjuncts of C, it also realises C itself. ◀

▶ **Lemma 18.** *If $C = \mathsf{visits}(R_1, t_1) \wedge \cdots \wedge \mathsf{visits}(R_k, t_k)$ is a conjunction of atomic $(\mathcal{R}, \mathsf{Moment})$-events, S is a trajectory sample and v a speed bound, then $(S, v) \models C$ if and only if there exist k spatial locations $p_1, \ldots, p_k$ such that*

(1) *for $i = 1, \ldots, k$ we have $p_i \in R_i$, and*

(2) *the sample containing the points $(p_1, t_1), \ldots, (p_k, t_k)$, as well as the ones in S, is v-consistent.*

We remark that we consider condition (2) from the lemma to be false when some space-time point among $(p_1, t_1), \dots (p_k, t_k)$ shares its temporal component with a point in S, while their spatial component differs.

Proof. The "only if"-direction is obvious. We prove the "if"-direction. Assume there are locations $p_1, \dots, p_k$ satisfying (1) and (2). Now take the trajectory γ to be the linear interpolation trajectory of the sample containing the points $(p_1, t_1), \dots, (p_k, t_k)$, as well as the ones in S. It is clear that γ matches S and assumption (2) implies it is a v-bounded trajectory. Finally, assumption (1) implies that γ realises C. ◀

▶ **Definition 19.** *We say two $(\mathcal{R}, \mathcal{T})$-events A and B are* equivalent *if for every sample S and speed bound v, we have $(S, v) \models A$ if and only if $(S, v) \models B$.*

The following proposition follows directly from the fact that, for $p \in \mathbb{R}^2$, we have $p \notin R(\varphi)$ if and only if $p \in R(\neg\varphi)$.

▶ **Proposition 20.** *A $(\mathsf{SemiAlg}, \mathsf{Moment})$-event of the form $\neg\mathsf{visits}(R(\varphi), t)$ is equivalent to the event $\mathsf{visits}(R(\neg\varphi), t)$.*

Given an arbitrary $(\mathsf{SemiAlg}, \mathsf{Moment})$-event, we can convert it into negation normal form and use Proposition 20 to remove all negations. The result of this process is a positive $(\mathsf{SemiAlg}, \mathsf{Moment})$-event that is equivalent to the original. Because this process can be performed in linear time (with respect to the length of the event), we can assume that any given $(\mathsf{SemiAlg}, \mathsf{Moment})$-event is positive, without loss of generality.

▶ **Theorem 21.** *In terms of data complexity, the $(\mathsf{SemiAlg}, \mathsf{Moment})$-realisability problem is decidable in linear time.*

Proof. We describe an algorithm to decide the realisability problem of a $(\mathsf{SemiAlg}, \mathsf{Moment})$-event e for input sample $S = \langle (p_1, t_1), \dots, (p_n, t_n) \rangle$, where $p_i = (x_i, y_i)$, and speed bound v. Noting the remark made above, we assume that e is positive. The first step is to convert e into its disjunctive normal form $\bar{e}$. Since we are dealing with data complexity, the possibly increased size of $\bar{e}$, compared to e, has no impact on the running time of our method. In fact, because the number of disjuncts of $\bar{e}$ is constant, it is sufficient to show that the realisability problem for a single disjunct can be decided in linear time.

Consider a disjunct C of $\bar{e}$. Because $\bar{e}$ is positive and in DNF, the event C must be a conjunction of atomic events. This means we can apply Lemma 17, and we can determine whether $(S, v) \models C$ by testing whether each of the following conditions are met:

(1) $(\langle (p_1, t_1) \rangle, v) \models C_{(-\infty, t_1]}$,
(2) for $i = 1, \dots, n-1$, we have $(\langle (p_i, t_i), (p_{i+1}, t_{i+1}) \rangle, v) \models C_{[t_i, t_{i+1}]}$, and
(3) $(\langle (p_n, t_n) \rangle, v) \models C_{[t_n, +\infty)}$.

We focus our attention to condition (2). For every value of i, we want to decide whether $C_{[t_i, t_{i+1}]}$ is realisable for sample $\langle (p_i, t_i), (p_{i+1}, t_{i+1}) \rangle$ and speed bound v. Let us write the conjunction $C_{[t_i, t_{i+1}]}$ as $\mathsf{visits}(R(\varphi_1), t'_1) \wedge \dots \wedge \mathsf{visits}(R(\varphi_k), t'_k)$, with $t'_1 \leq \dots \leq t'_k$. We remark that this implies $t_i \leq t'_1 \leq \dots \leq t'_k \leq t_{i+1}$. From Lemma 18, we know that $(\langle (p_i, t_i), (p_{i+1}, t_{i+1}) \rangle, v) \models C_{[t_i, t_{i+1}]}$ if and only if there exist locations $q_1, \dots, q_k \in \mathbb{R}^2$, with $q_i = (x'_i, y'_i)$, such that

(a) for $j = 1, \dots, k$ we have $q_j \in R(\varphi_j)$, and
(b) the sample $\langle (p_i, t_i), (q_1, t'_1), \dots, (q_k, t'_k), (p_{i+1}, t_{i+1}) \rangle$ is v-consistent.

In other words, we have $(\langle (p_i, t_i), (p_{i+1}, t_{i+1})\rangle, v) \models C_{[t_i, t_{i+1}]}$ if and only if the formula $\psi = \exists x'_1 \exists y'_1 \exists \ldots \exists x'_k \exists y'_k (\psi_a \wedge \psi_b)$ is true, where ψ_a is $\varphi_1(x'_1, y'_1) \wedge \cdots \wedge \varphi_k(x'_k, y'_k)$, expressing condition (a), and to express condition (b), we take ψ_b to be the conjunction of $k+1$ distance inequalities.

Because ψ is a first-order logic sentence over the ordered field of real numbers, its truth can be determined by a decision procedure for the theory of real closed fields (first described by Tarski [19], we refer to Basu et al. [2] for a modern exposition). We note that the size of ψ is independent of n, and thus has constant size in the data complexity setting (k is bounded by the length of C). The time needed to determine the truth of ψ is thus also constant. To test for condition (2), we have to perform the above steps for $n-1$ values of i. Conditions (1) and (3) can both be tested in constant time, in a manner similar to the above, the only difference is that the constructed sentence requires one less inequality for expressing v-consistency. We have thus shown that the realisability problem for a single disjunct of $\bar{e}$ can be answered in $O(n)$ time. This concludes the proof. ◀

Our next result concerns the data complexity of the (SemiAlg, Interval)-realisability problem for positive events.

▶ **Lemma 22.** *If e is a positive $(\mathcal{R}, \mathsf{Interval})$-event containing k distinct atoms $A_1, \ldots, A_k$, where $A_i = \mathsf{visits}(R_i, T_i)$, then $(S, v) \models e$ if and only if there exist k space-time points $(p_1, t_1), \ldots, (p_k, t_k)$ such that*

(1) *the sample containing the points $(p_1, t_1), \ldots, (p_k, t_k)$, as well as the ones in S, is v-consistent, and*

(2) *the Boolean expression obtained by replacing A_i in e by true if $(p_i, t_i) \in R_i \times T_i$, and by false otherwise, evaluates to true.*

Proof. We first prove the "if"-direction. Assume that there exist $(p_1, t_1), \ldots, (p_k, t_k)$ satisfying (1) and (2). Let S' be the sample containing the points $(p_1, t_1), \ldots, (p_k, t_k)$, as well as the ones in S. Assumption (1) says S' is v-consistent, which means $\mathsf{LIT}(S')$ is v-bounded. Because S' is an extension of S, and γ matches S', it must also match S. The trajectory $\mathsf{LIT}(S')$ realises all the atoms A_i for which $(p_i, t_i) \in R_i \times T_i$, and potentially others. Because of assumption (2) and the fact that e is positive, $\mathsf{LIT}(S')$ realises e, and thus $(S, v) \models e$.

For the "only if"-direction, we assume that $(S, v) \models e$. That means that there exists a v-bounded γ realising e and matching S. We choose $(p_1, t_1), \ldots, (p_k, t_k)$ as follows. For every atom A_i which γ realises, we know that γ visits some point in $R_i \times T_i$, and take (p_i, t_i) to be such point. For an atom A_i not realised by γ, we take (quite arbitrarily) (p_i, t_i) to be the first anchor point of S. Then, $(p_i, t_i) \notin R_i \times T_i$, since otherwise γ would realise A_i. All points of $(p_1, t_1), \ldots, (p_k, t_k)$ and S are visited by γ, so the sample containing all those points must be v-consistent, satisfying condition (1). Because γ realises A_i if and only if $(p_i, t_i) \in R_i \times T_i$ and γ realises e, condition (2) is also satisfied. ◀

The following proposition follows directly from the definition of v-consistency and the fact that the distance function d obeys the triangle inequality.

▶ **Proposition 23.** *If M is an (unordered) finite set of space-time points, then the sample containing all points in M is v-consistent if and only if for every pair of points (p_1, t_1) and (p_2, t_2) from M, we have $d(p_1, p_2) \leq v \cdot |t_1 - t_2|$.*

It will be of interest later that the condition $d(p_1, p_2) \leq v \cdot |t_1 - t_2|$ from the above proposition is equivalent to $d(p_1, p_2)^2 \leq v^2 \cdot (t_1 - t_2)^2$, which, if $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, is a polynomial inequality with variables $x_1, y_1, t_1, x_2, y_2, , t_2, v$.

▶ **Theorem 24.** *In terms of data complexity, the* (SemiAlg, Interval)*-realisability problem for positive events can be decided in polynomial time.*

Proof. We describe an algorithm to decide the realisability of a positive (SemiAlg, Interval)-event e for input sample S, of length n, and speed bound v. Lemma 22 gives a condition equivalent to $(S, v) \models e$. We can express this condition using a first-order logic sentence over the ordered field of real numbers $\psi = \exists x_1 \exists y_1 \exists t_1 \ldots \exists x_k \exists y_k \exists t_k (\psi_1 \wedge \psi_2)$, where ψ_1 expresses part (1) of Lemma 22, and ψ_2 expresses part (2). To express part (1), stating that the sample containing $(x_1, y_1, t_1), \ldots, (x_k, y_k, t_k)$ as well as the points from S is v-consistent, we can take ψ_1 to be a conjunction of $(n + k)^2$ distance inequalities, as per Proposition 23. To express the second part, we construct ψ_2 by taking e and replacing every atom $\mathsf{visits}(R(\varphi_i), [t_i^-, t_i^+])$ by the formula $\varphi_i(x_i, y_i) \wedge t_i^- \leq t_i \wedge t_i \leq t_i^+$.

We have now constructed a formula ψ which is true if and only if $(S, v) \models e$. Thus, if there is a method to determine the truth ψ, we can decide the realisability problem for positive (SemiAlg, Interval)-events. Because ψ is an existantial sentence, we can apply known decision procedures for the existential theory of the reals. Of course, the time complexity required to decide the realisability of positive (SemiAlg, Interval)-events in the described manner, depends on the time complexity of the existential theory of the reals. In [2] (see theorem 13.14), an upper bound of $s^{m+1}d^{O(m)}$ is given, where s is the number of polynomials occuring in the formula, m the number of variables, and d the maximum degree of the polynomials. Because we are considering data complexity, the number of polynomials in the φ_i's, as well as their maximum degree, is constant, and so is k. This implies that ψ contains $O(n^2)$ polynomials of constant degree, and $3k$ variables. Thus, by Theorem 13.14 from [2], the truth of ψ can be decided in $O((n^2)^{3k+1}) = O(n^{6k+2})$ time, which is polynomial in n. It is also clear that ψ can be constructed in polynomial time. ◀

4.3 A class of events for which the realisability problem has polynomial time combined complexity

Until now, we have only seen hardness results in the query (and thus, combined) complexity setting. In this section, we give an example of a class of events for which the realisability problem has polynomial-time combined complexity.

An $(\mathcal{R}, \mathcal{T})$-*literal* is either an atomic $(\mathcal{R}, \mathcal{T})$-event, or the negation of an atomic $(\mathcal{R}, \mathcal{T})$-event. Conjunctions of (Point, Moment)-literals provide a class of events for which the realisability problem has an efficient solution in terms of combined complexity. Before giving our result, we first prove a lemma.

▶ **Lemma 25.** *If S_1 and S_2 are trajectory samples, then there exists a v-bounded trajectory matching S_1 and not visiting any point in S_2 if and only if*

(1) *S_1 does not contain a point in S_2,*

(2) *S_1 is v-consistent, and*

(3) *if $\mathsf{LIT}(S_1)$ visits a point from S_2, on the line segment between points (p_i, t_i) and (p_{i+1}, t_{i+1}) from S_1, then $d(p_i, p_{i+1}) < v \cdot (t_{i+1} - t_i)$.*

Proof. It is obvious that the conditions (1), (2) and (3) are necessary for the existence of a v-bounded trajectory matching S_1 and not visiting points in S_2. To show that they are also sufficient, we assume that all three conditions are met. Consider the trajectory $\mathsf{LIT}(S_1)$. By condition (2), it is v-bounded. In case $\mathsf{LIT}(S_1)$ visits one or more points from S_2, we show that we can extend (by adding points) S_1 to a sample S^*, such that $\mathsf{LIT}(S^*)$ is v-bounded, matches S_1 and does not visit points from S_2. Because S^* is obtained by adding points to S_1,

it is obvious that $\mathsf{LIT}(S^*)$ matches S_1. Let us say that $\mathsf{LIT}(S_1)$ visits a point (q,t) from S_2 on the line segment between (p_i,t_i) and (p_{i+1},t_{i+1}). From (1) we know that $t_i < t < t_{i+1}$, and from (3) we have $d(p_i,p_{i+1}) < v \cdot (t_{i+1}-t_i)$. Together, these observations imply there must be a small disk around q, such that for every p in this disk, the extension of S_1 with the point (p,t) remains v-consistent. Informally, this means we can deviate $\mathsf{LIT}(S_1)$ slightly between t_i and t_{i+1}, while the trajectory remains v-bounded. Now, for every point (q',t') of S_2 with $t_i < t' < t_{i+1}$, there is at most one choice for p inside the disk that makes the linear interpolation trajectory of the extended sample visit (q',t'). Because there are infinitely many points in the disks, we can choose p such that, between t_i and t_{i+1}, none of the points from S_2 are visited. We can repeat this process for each line segment of $\mathsf{LIT}(S_1)$ that visits a point from S_2. The linear interpolation trajectory of the resulting sample S^* then matches S_1, is v-bounded and does not visit points from S_2, as desired. ◀

▶ **Theorem 26.** *In terms of combined complexity, the* (Point, Moment)*-realisability problem for a conjunction of k literals and a trajectory sample of length n is decidable in $O(n+k\log k)$ time.*

Proof. We assume that we are given a conjunction C of k (Point, Moment)-literals, a sample S of length n and a speed bound v as input. Let $P_1,\ldots,P_m$ be the positive atoms occuring in C and let $N_1,\ldots,N_\ell$ be the atoms occuring negated in C. First we compute the sample $S_1 = \langle (p_1,t_1),\ldots(p_{n+m},t_{n+m})\rangle$, containing both the points in S and the ones occuring in the atoms $P_1,\ldots,P_m$. Computing this sample requires ordering the points occuring in $P_1,\ldots,P_m$ by time, and merging these with those from S (which are already ordered by time). This can be done in $O(n+k\log k)$ time. We also compute a sample S_2, containing the points from $N_1,\ldots,N_\ell$, in $O(k\log k)$ time. Now we have $(S,v) \models C$ if and only if there exists a v-bounded trajectory matching S_1 and not visiting any point in S_2. This can be determined by testing for the conditions of Lemma 25 in $O(|S_1|+|S_2|)$ time, where $|S_1|+|S_2| = n+m+\ell = n+k$. Thus, the total time required by the described procedure is $O(n+k\log k)$. ◀

▶ **Corollary 27.** *In terms of combined complexity, the* (Point, Moment)*-realisability problem for an event in disjunctive normal form, of length k, and a trajectory sample of length n is decidable in $O(kn+k\log k)$ time.*

Proof. We assume that we are given a (Point, Moment)-event e in disjunctive normal form, of length k, a sample S of length n and a speed bound v as input. Then, e is of the form $C_1 \vee \cdots \vee C_\ell$, where every disjunct C_i is a conjunction of literals. Let us say that the event C_i contains k_i literals. Because e is realisable if and only if one of the C_i's is realisable, we can use Theorem 26 to determine the realisability of e, by determining it for each of the disjuncts. This takes $\sum_{i=1}^{\ell} O(n+k_i\log k_i)$ time, which is $O(kn+k\log k)$. ◀

5 Conclusion

We have proposed a family of query languages for trajectory sample databases. Query expressions in these languages contain events, which are used to describe constraints on trajectories. When performing query evaluation, an essential problem required to be solved is that of the realisability of an event. We studied the complexity of this realisability problem in terms of data, query and combined complexity. These results are summarised in Table 1. These complexity results are given in a computational model wherein (arithmetic and comparison) operations on rational numbers take unit time. It is not clear whether our

results remain true in a model in which we measure the cost of these operations in terms of the length of the bit-representation of rational numbers. Also, we did not give much attention to the evaluation of query expressions outside of the realisability problem. We assumed that query expressions are evaluated in some standard way, but more efficient strategies might exist.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

2 S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*. Algorithms and Computation in Mathematics 10. Springer, Berlin, 2003.

3 Jacek Bochnak, Michel Coste, and Marie-Françoise Roy. *Real Algebraic Geometry*, volume 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer-Verlag, 1998.

4 Vania Bogorny, Bart Kuijpers, and Luis Otávio Alvares. ST-DMQL: A semantic trajectory data mining query language. *Int. J. Geogr. Inf. Sci.*, 23(10):1245–1276, 2009. URL: `http://www.informaworld.com/smpp/content%7Edb=all%7Econtent=a915093101%7Efrm=abslink`.

5 L. Burns. *Transportation, Temporal, and Spatial Components of Accessibility*. Lexington Books, Lexington, MA, 1979.

6 Max J. Egenhofer. Approximation of geospatial lifelines. In Elisa Bertino and Leila De Floriani, editors, *SpadaGIS, Workshop on Spatial Data and Geographic Information Systems*. University of Genova, 2003.

7 Fosca Giannotti and Dino Pedreschi, editors. *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*. Springer, 2008. `doi:10.1007/978-3-540-75177-9`.

8 R. Güting and M. Schneider. *Moving Object Databases*. Morgan Kaufmann, 2005.

9 T. Hägerstrand. What about people in regional science? *Papers of the Regional Science Association*, 24:7–21, 1970.

10 Kathleen Hornsby and Max J. Egenhofer. Modeling moving objects over multiple granularities. *Ann. Math. Artif. Intell.*, 36(1-2):177–194, 2002. `doi:10.1023/A:1015812206586`.

11 Tomasz Imieliński and Witold Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, September 1984. `doi:10.1145/1634.1886`.

12 Donald Janelle and Michael Goodchild. Diurnal patterns of social group distributions in a canadian city. *Economic Geography*, 59(4):403–425, 1983. URL: `http://www.jstor.org/stable/144166`.

13 B. Lenntorp. *Paths in Space-Time Environments: A Time-Geographic Study of the Movement Possibilities of Individuals*. Number 44 in Series B. Lund Studies in Geography, 1976.

14 H.J. Miller. Modeling accessibility using space-time prism concepts within geographical information systems. *International Journal of Geographical Information Systems*, 5:287–301, 1991. `doi:10.1080/02693799108927856`.

15 H.J. Miller. A measurement theory for time geography. *Geographical Analysis*, 2005. `doi:10.1111/j.1538-4632.2005.00575.x`.

16 Christos H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977. `doi:10.1016/0304-3975(77)90012-3`.

17 Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady L. Andrienko, Natalia V. Andrienko, Vania Bogorny, Maria Luisa Damiani, Aris Gkoulalas-Divanis, José Antônio Fernandes de Macêdo, Nikos Pelekis, Yannis Theodoridis, and Zhixian Yan. Semantic trajectories modeling and analysis. *ACM Comput. Surv.*, 45(4):42:1–42:32, 2013. `doi:10.1145/2501654.2501656`.

18 Dieter Pfoser and Christian S. Jensen. Capturing the uncertainty of moving-object representations. In Ralf Hartmut Güting, Dimitris Papadias, and Frederick H. Lochovsky, editors, *Advances in Spatial Databases, 6th International Symposium, SSD'99, Proceedings*, volume 1651 of *Lecture Notes in Computer Science*, pages 111–132. Springer, 1999. `doi:10.1007/3-540-48482-5_9`.

19 Alfred Tarski and J. C. C. McKinsey. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1951.

20 G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004. doi:10.1145/1016028.1016030.

21 Zhixian Yan, Dipanjan Chakraborty, Christine Parent, Stefano Spaccapietra, and Karl Aberer. Semantic trajectories: Mobility data computation and annotation. *ACM Trans. Intell. Syst. Technol.*, 4(3):49:1–49:38, 2013. doi:10.1145/2483669.2483682.

Temporal Ensemble Logic for Integrative Representation of the Entirety of Clinical Trials

Xiaojin Li
The University of Texas Health Science Center at Houston, TX, USA

Yan Huang
The University of Texas Health Science Center at Houston, TX, USA

Rashmie Abeysinghe
The University of Texas Health Science Center at Houston, TX, USA

Zenan Sun
The University of Texas Health Science Center at Houston, TX, USA

Hongyu Chen
University of Florida, Gainesville, FL, USA

Pengze Li
Mayo Clinic in Florida, Jacksonville, FL, USA

Xing He
Indiana University Bloomington, Bloomington, IN, USA

Shiqiang Tao
The University of Texas Health Science Center at Houston, TX, USA

Cui Tao
Mayo Clinic in Florida, Jacksonville, FL, USA

Jiang Bian
Indiana University Bloomington, Bloomington, IN, USA

Licong Cui
The University of Texas Health Science Center at Houston, TX, USA

Guo-Qiang Zhang[1]
The University of Texas Health Science Center at Houston, TX, USA

Abstract

Clinical trials are typically specified with protocols that define eligibility criteria, treatment regimens, follow-up schedules, and outcome assessments. Temporality is a hallmark of all clinical trials, reflected within and across trial components, with complex dependencies unfolding across multiple time points. Despite their importance, clinical trial protocols are described in free-text format, limiting their semantic precision and the ability to support automated reasoning, leverage data across studies and sites, or simulate trial execution under varying assumptions using Real-World Data. This paper introduces a formalized representation of clinical trials using Temporal Ensemble Logic (TEL). TEL incorporates metricized modal operators, such as "always until t" ($\Box_t$) and "possibly until t" ($\Diamond_t$), where t is a time-length parameter, to offer a logical framework for capturing phenotypes in biomedicine. TEL is more expressive in syntax than classical linear temporal logic (LTL) while maintaining the simplicity of semantic structures. The attributes of TEL are exploited in this paper to formally represent not only individual clinical trial components, but also the timing and sequential dependencies of these components as a whole. Modeling strategies and demonstration case studies are provided to show that TEL can represent the entirety of clinical trials, whereby providing a formal logical framework that can be used to represent the intricate temporal dependencies in trial structure specification. Since clinical trials are a cornerstone of evidence-based medicine, serving as the scientific basis for evaluating the safety, efficacy, and comparative effectiveness of therapeutic interventions, results reported here can serve as a stepping stone that leads to scalable, consistent, and reproducible representation and simulation of clinical trials across all disease domains.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases Temporal ensemble logic, Clinical trials, Logic-based modeling

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.13

[1] Corresponding author

32nd International Symposium on Temporal Representation and Reasoning (TIME 2025).
Editors: Thierry Vidal and Przemysław Andrzej Wałęga; Article No. 13; pp. 13:1–13:16

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

Funding This work was supported in part by the National Science Foundation Award IIS2500624 and the National Institutes of Health grants R01AG084236, of the United States. The views of the paper are those of the authors and do not reflect those of the funding agencies.

1 Introduction

Clinical trials are a cornerstone of biomedical research, providing rigorous evidence for the safety, efficacy, and comparative effectiveness of therapeutic interventions [32]. These studies follow structured protocols that define eligibility criteria, intervention timing, follow-up assessments, and outcome measures. Each of these components is inherently temporal, unfolding across defined intervals and often involving complex interdependencies. As trial designs evolve to include adaptive structures, multiple treatment arms, and longitudinal assessments, there is an increasing demand for formal methods that can represent the full temporal structure of clinical trial execution [5].

One of the challenges in clinical trial informatics is the lack of a formal representation framework for trial protocol specifications. Basic protocol components such as treatment windows, observation periods, washout phases, and outcome evaluations are usually described informally or in free-text, making them difficult to reuse and interpret computationally. This limitation impairs the ability to simulate trials, align patient trajectories, validate protocol logic, or integrate information across studies [34].

Temporal logic provides a formal framework for reasoning about time, enabling the representation of temporal facts and relationships such as event ordering and interval duration [48]. Foundational work like Allen's interval-based temporal logic has shaped this field by formalizing temporal relations between intervals, laying the groundwork for advanced temporal inference [40]. The expressive power of temporal logic is particularly valuable in clinical contexts, where complex temporal dependencies must be modeled to simulate trial scenarios and interpret patient narratives [18, 48]. In clinical trials, temporal reasoning is pivotal for structuring protocol logic, modeling event sequences, and enforcing timing constraints [39, 30]. These capabilities support key tasks such as cohort modeling, protocol validation, eligibility screening, and outcome prediction across the trial lifecycle. Recent advances have also integrated temporal logic into clinical natural language processing (NLP), improving the extraction and prediction of temporally grounded events [41]. By incorporating temporal entailment between patient states, these methods enable longitudinal reasoning essential for assessing treatment efficacy and guiding patient care [1, 41].

Despite such developments and potentials, there is a lack of logical-framework with demonstrated capability and translatability in representing clinical trial protocols with temporal and semantic precision and the ability to support automated reasoning, leverage data across studies and sites, or simulate trial execution under varying assumptions using Real-World Data [22, 35, 49]. This paper introduces a framework for formalized representation of clinical trials using Temporal Ensemble Logic (TEL [48]). TEL incorporates metricized modal operators, such as "always until t" ($\Box_t$) and "possibly until t" ($\Diamond_t$), where t is a time-length parameter, to offer a logical framework for capturing phenotypes in biomedicine. TEL is more expressive in syntax than classical linear temporal logic (LTL) while maintaining the simplicity of semantic structures.

The attributes of TEL are exploited in this paper to formally represent not only individual clinical trial components, but also the timing and sequential dependencies of these components as a whole. We demonstrate that TEL can facilitate formal modeling of key elements including eligibility criteria, baseline visits, interventions, follow-up visits, and outcome assessments. It

supports both individual-level and cohort-level reasoning, making it well-suited for clinical trial simulation. Our modeling strategy uses atomic propositions formulated from a domain-specific ontology constructed from standardized biomedical vocabularies such as SNOMED CT [10], ICD-9/10-CM [16], CPT [12], RxNorm [28], and LOINC [25]. Trial participants' electronic medical records serve as models, over the domain of positive integers $\mathbb{N}^+$ which represent days (typically the finest granularity in clinical trial protocols), weeks, or months.

The main contributions of this study are as follows: 1) we introduce a modeling strategy to capture clinical trial protocol components using TEL, providing a logical specification of temporal dependencies in clinical trial protocols; 2) we develop the AD Clinical Trial Ontology (ADCTO), a lightweight, domain-specific ontology aligned with biomedical standards to codify semantic entities used in TEL for this purpose, and 3) we prototype a simulation system with interfaces that represent trial specifications as TEL formulas and executes them against Real-World Data (RWD) to support protocol validation and virtual execution using the model-checking paradigm.

The remainder of this paper is organized as follows: Section 2 introduces the formal foundation of TEL. Section 3 describes two major categories of trial simulations. Section 4 presents our modeling of clinical trials. Section 5 details the TEL-based logical formalization of clinical trial components. Section 6 reports on the implementation, ontology development, and simulation system.

2 Preliminaries

2.1 Temporal Ensemble Logic

In modeling clinical trials, we consider the time domain of positive integers $\mathbb{N}^+$, which represent days lapsed from the start of a clinical trial.

The basic construct of TEL [48] includes two types of terms: integer terms over $\mathbb{N}^+$, and logical formulas. Integer terms $s, t, u, v, \ldots$ consist of constants $a, b, c, \ldots$, variables $x, y, z, \ldots$ (from a set Var), and the addition of these terms $(s + t)$. We write Term for the set of all such integer terms. Logical formulas φ consist of primitive propositions p, indexed formulas φ_t, Boolean connectors ($\wedge$, $\neg$), time-indexed modalities $\Box_t\varphi$ and $\Diamond_t\varphi$, and first-order quantification over variables in Var.

In Backus–Naur form (BNF) notation, we have $s, t ::= a \mid x \mid (s+t)$, where a is an integer constant, $x \in \mathsf{Var}$, and $(s + t)$ represent the addition of two integer terms.

For TEL formula, we have, with t for integer term and x for variable over $\mathbb{N}^+$:

$$\varphi, \psi ::= p \mid \varphi_t \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Box_t\varphi \mid \Diamond_t\varphi \mid \forall x\varphi \mid \exists x\varphi.$$

Atomic propositions p come from a pre-defined set Prop, a set of clinical codes and ontological terms according to the conceptual correspondence provided in Table 1.

To specify the semantics of the above formulas, we define an interpretation as a function $\alpha : \mathbb{N}^+ \to 2^{\mathsf{Prop}}$, which determines, at each time-point $i \in \mathbb{N}^+$, a subset of atomic propositions that are assigned true. We can use the notation of formal languages to describe such interpretations. The alphabet is Σ $(= 2^{\mathsf{Prop}})$, to account for all possible truth-status of each proposition in Prop at a given time. Following standard formal language convention, Σ^* represents the set of all finite words over Σ, Σ^+ represents the set of all no-empty finite words over Σ, and Σ^ω represents all *ω-words* over Σ. Therefore, we can write α in the form of an ω-word $\sigma[1]\sigma[2]\cdots$, with $\sigma[i] \in \Sigma$ being the ith letter of α for all $i \geq 1$.

▶ **Definition 1.** *We define, given an ω-word $\alpha = \sigma[1]\sigma[2]\cdots$ over Σ and any $i \geq 1$,*

$(\alpha, i) \models p$ if $p \in \sigma[i]$;
$(\alpha, i) \models \varphi_t$ if $(\alpha, i+t) \models \varphi$;
$(\alpha, i) \models \neg\varphi$ if $(\alpha, i) \not\models \varphi$;
$(\alpha, i) \models \phi \wedge \psi$ if $(\alpha, i) \models \varphi$ and $(\alpha, i) \models \psi$;
$(\alpha, i) \models \phi \vee \psi$ if $(\alpha, i) \models \varphi$ or $(\alpha, i) \models \psi$;
$(\alpha, i) \models \Box_t\varphi$ if $(\alpha, j) \models \varphi$ for all j with $i \leq j < i + t$;
$(\alpha, i) \models \Diamond_t\varphi$ if $(\alpha, j) \models \varphi$ for some j with $i \leq j < i + t$;
$(\alpha, i) \models \forall x\varphi$ if for all $k \geq 1$, $(\alpha, i) \models \varphi[x := k]$.
$(\alpha, i) \models \exists x\varphi$ if there exists $k \geq 1$, $(\alpha, i) \models \varphi[x := k]$,

where $\varphi[x := k]$ is the standard syntactic convention for the formula obtained by replacing all free occurrences of variable x in φ by constant k.

2.2 Conceptual Correspondence for Trial Modeling

Clinical trials involve complex temporal dependencies between interventions, patient responses, and outcomes, requiring a formal approach to temporal reasoning. Existing logical frameworks in computer science and biomedicine either lack expressiveness for capturing trial dynamics or impose syntactic constraints that limit their applicability to real-world trial scenarios.

Table 1 Correspondence Between Clinical Trial Components and TEL Elements.

Clinical Trial Entities	*Logical Constructs*	*Logical Forms*
Controlled vocabularies, ontology terms	Atomic propositions	Prop
Temporal relationships	Modal and logical operators	$\Box_t, \Diamond_t, \wedge, \exists$
Electronic medical records	Semantic models	$\sigma[1]\sigma[2]\cdots$
Start Event, Index Event, Inclusion Criteria, Exclusion Criteria, Observation Window, Baseline Visit, Intervention Phase, Follow-Up Visits, Outcome Assessment, Base Criteria, Case Criteria, Control Criteria	TEL formulas	S, I, IC, EC, W, B, T, F, O, $Base$, $Case$, $Control$
Trial execution	Model-checking	$(\alpha, i) \models \varphi$

To make the technical motivations and design decisions more intuitive, we begin with an illustrative example. This example is intended to highlight the practical needs and conceptual foundations for combining first-order and modal logic primitives in the representation of biological phenotypes. By grounding the discussion in concrete use cases, we aim to clarify the expressive requirements that drive our modeling strategy.

▶ **Example 2.** For an Alzheimer's Disease (AD) clinical trial of demonstrating the efficacy of at least one dose of a medicine (H3 receptor antagonist) in comparison to placebo on cognitive performance in patients with mild to moderate AD [44], TEL provides a formal structure for representing its constraints and event relationships. The key components of the protocol can be expressed as follows:

$$\exists x \exists i \Big[S_x \wedge I_{x+i} \wedge (\Diamond_{4w} B)_{(x+i)} \wedge (\Box_{24w} T)_{x+i+4w} \wedge (\Diamond_{10w} F)_{x+i+28w} \wedge (\Diamond_{1w} O)_{x+i+38w} \Big]$$

where the start date x corresponds to the date of informed consent (S). The index event (I) is defined as the cognitive evaluations and biomarker analyses i days after consent. Baseline visit (B) is conducted within four weeks (4w) prior to intervention. The intervention phase (T) involves the administration of a medicine or placebo over 24 weeks (24w). Follow-up visits (F) are scheduled to monitor safety and efficacy parameters for up to 10 weeks (10w). The outcome assessment (O) occurs at week 38 (38w), evaluating changes in cognitive function from baseline.

2.3 Related Work: Temporal Logics for Clinical Trials

Many existing temporal logics have been developed to express different types of temporal relationships, each with its own strengths and trade-offs. LTL [33] captures linear sequences of states to express event orders. Based on LTL, Metric Temporal Logic (MTL) [29] adds explicit timing bounds to temporal operators. Interval Temporal Logic (ITL) [9, 14] focuses on intervals rather than points, capturing relationships. Despite the extensive development of temporal logic frameworks, their practical application in clinical trial specification remains limited. Giordano et al. modeled stroke guidelines as concurrent processes in LTL and verified them with SPIN [13], while Sanati et al. introduced a MTL-based Description Logics for formulating eligibility criteria for clinical trials [2]. Shankar et al. [39] developed a formal ontology to encode temporal constraints in clinical trials, allowing precise specification of timeline requirements (e.g., "lab test 2 must occur within 7 days after dose 1") using ITL. CNTRO [42, 43] encodes logical and temporal constraints directly within the structure of the Web Ontology Language (OWL). The Time Event Ontology (TEO) proposed by Li et al. [21] extends CNTRO by harmonizing Allen's interval algebra with a suite of basic time relations, although it lacks a language of logic for modeling trials.

Table 2 Logic-based Modeling for Clinical Trials.

Logic	Constructs	Expressiveness	Clinical Trial Application	Modeled Component
LTL	Point-based Decidable	Limited	[13]	Entire trial
MTL	Point-based Decidable (restricted cases)	Limited	[2]	Eligibility criteria
ITL	Interval-based Decidable (restricted cases)	Lack point-based references	[21, 39, 42]	Entire trial
TEL	**Point-based, limited First-order Decidability for used fragment open**	**Good fit for purpose (integrative, interval and point, first-order)**	**This study**	**Entire trial**

LTL can express sequential temporal dependencies and comes with mature automated tools, but it has limited expressiveness. It cannot capture complex interval constraints such as "observe within a specified time window". MTL can express time-bounded constraints such as visit windows or dosing intervals, but cannot directly relate distinct times or intervals. It lacks the ability to express dependencies such as "dose two administered exactly 14 days after dose one, followed by a visit 7 days later." ITL can model observation windows and treatments with interval-based semantics (e.g., "treatment continues during observation window") but it becomes overly complex and lacks cross-timeline reference capability without additional constructs. Table 2 summarizes these observations.

The use of TEL for trial modeling brings the following advantages. TEL has already been demonstrated as a natural but minimalistic logical framework with potential as an attractive approach to formalizing phenotypes in biomedicine [48]. Using the model-checking paradigm, TEL can support both point-based and interval-based temporal constructs. TEL is highly expressive in modeling temporal properties, but restricted in aspects not essential to targeted applications. The scope of first-order quantifiers for TEL is limited to unary predicates derived from an existing formula, rather than over independently given predicates. As can be seen in subsequent developments, the specific fragment of TEL used for this study does not involve the $\forall$ quantifier. Although it is beyond the scope of the current paper to address theoretical developments in decidability and algorithmic complexity results for specific fragments of TEL (given its general undecidability [48]), our experimental results show practical feasibility in reasonable settings.

3 Clinical Trial Simulation (CTS)

Clinical trials serve as the cornerstone of medical research, providing rigorous assessments of the safety and efficacy of novel interventions. However, traditional clinical trial methodologies have several challenges, including ethical considerations, patient recruitment difficulties, and potential confounding variables. Advanced simulation-based approaches have been developed to address these limitations, leveraging real-world data and computational modeling to enhance trial design and execution [4]. This paper examines two key types of clinical trial simulations [19]: Self-Controlled Case Series [31] and Eligibility Cohort Building [11].

3.1 Self-Controlled Case Series (SCCS)

The Self-Controlled Case Series (SCCS) method is a widely used design for evaluating the association between medical interventions and adverse events. By treating individuals as their own controls, SCCS effectively mitigates confounding from fixed patient-level characteristics such as genetics and lifestyle [46]. In this simulation-based framework, longitudinal health records are used to construct individual timelines of intervention exposure and subsequent adverse events. A predefined risk window is specified following the intervention, and the frequency of events during this period is compared to baseline periods within the same individual using statistical modeling.

For example, a recent study investigating thrombosis risk after COVID-19 vaccination applied SCCS by extracting temporal sequences of vaccination dates and thrombotic events from electronic health records [17]. By comparing the incidence of events before and after vaccination within each individual, the study reduced between-subject variability and provided robust evidence of causal relationships.

3.2 Eligibility Cohort Specification

Patient recruitment is a persistent bottleneck in clinical trial implementation. Eligibility Cohort Building Simulations offer a data-driven strategy to improve recruitment efficiency by leveraging EHR data to model real-world patient populations [36]. In this approach, researchers first define trial eligibility criteria such as age range, treatment history, and comorbidities, and apply them to EHR datasets to construct a virtual cohort. Computational models estimate the number of patients who meet the criteria and allow iterative refinement by adjusting thresholds or conditions to assess their impact on cohort size and diversity [20]. This simulation process enables the design of recruitment strategies that are both scientifically rigorous and operationally feasible.

For example, a Phase III clinical trial investigating a novel therapeutic for hormone receptor-positive breast cancer used this method to address recruitment delays. By simulating eligibility criteria including age between 30 and 75 years, prior chemotherapy, and absence of major cardiovascular conditions on a real-world breast cancer dataset, investigators identified constraints that limited enrollment. Adjusting these parameters, such as slightly expanding the age range, helped increase the candidate pool while preserving trial integrity [26].

4 Structure of Clinical Trial Specification

4.1 Individual Timeline-Based Structure of CTS

Start Event (Initialization). The start event denotes the earliest temporal reference point preceding trial enrollment. It may represent initial patient contact, the commencement of a data collection window, or the initiation of eligibility surveillance. Although not part of the trial intervention itself, this event serves as a critical anchor for modeling retrospective observation periods, washout phases, or baseline covariate collection from real-world data sources such as electronic health records (EHRs) or insurance claims.

Index Event (Trial Anchor Point). The index event marks the initiation of the trial timeline for each participant. Defined via clinical markers or diagnostic confirmation, this event anchors the temporal alignment of all subsequent protocol components. It facilitates consistent modeling of participant trajectories across eligibility evaluation, randomization, and treatment administration phases.

Observation Window (Pre-Index Interval). The interval between the start and index events constitutes the observation window, during which baseline covariates, prior exposures, or exclusionary conditions are assessed. This period may include look-back or washout durations essential for contextualizing eligibility and anchoring the index event.

Eligibility Criteria (Inclusion and Exclusion). Eligibility is determined based on defined inclusion and exclusion criteria. Inclusion criteria specify characteristics that must be present (e.g., age range, disease severity), while exclusion criteria identify disqualifying conditions (e.g., contraindications, comorbidities). These filters are applied either before or at the index to ensure clinical appropriateness and cohort homogeneity.

Baseline Visit (Pre-Intervention Assessments). Participants meeting eligibility requirements undergo a baseline visit for comprehensive pre-intervention evaluation. This includes the collection of demographic data, clinical measurements, and relevant biomarkers, which serve as reference points for outcome comparison and guide treatment group allocation.

Intervention Phase (Treatment Administration). The intervention phase involves the administration of assigned treatments such as experimental therapies, standard care, or placebos, initiated after the index event. This phase captures the protocol-defined exposure period and may incorporate adherence, dose variability, or pharmacokinetic modeling.

Follow-Up Visits (Post-Treatment Monitoring). Follow-up visits are conducted at scheduled intervals following the intervention phase to monitor clinical outcomes, detect adverse events, and track longitudinal changes in patient status. This component accounts for dropout risk, adherence behavior, and the timing of outcome manifestation.

Outcome Assessment (End Event). The final phase of the trial involves assessing predefined primary and secondary endpoints, including clinical efficacy, safety profiles, or composite outcomes. Outcome evaluation is conducted after the full observation period has elapsed or upon reaching specific clinical milestones, enabling rigorous analysis of treatment effects.

4.2 Cohort-Based Structure of CTS

The Individual Timeline-Based CTS captures the temporal dynamics of clinical events at the patient level, supporting analyses of time-dependent phenomena such as treatment onset and disease progression. While effective for intra-individual variability and temporal causality, it is less suited for between-group comparisons. In contrast, the Cohort-Based CTS organizes participants into predefined groups based on treatment or baseline characteristics, simulating longitudinal outcomes to assess efficacy, safety, event incidence, and dropout rates. This structure, aligned with parallel-arm trial designs, aids trial feasibility, sample size estimation, and recruitment planning. Cohorts are defined by Base Criteria (shared attributes), Case Criteria (presence of key exposures), and Control Criteria (absence of exposures), enabling robust comparative analyses and accounting for real-world complexities such as time-to-event variability, adherence, and attrition.

5 Logical Representation of CTS using Temporal Ensemble Logic

5.1 TEL-Based Formalization of Timeline-based CTS

This section formalizes a timeline-based CTS using the TEL framework. The model is anchored by a start event (S) and an index event (I), such as a diagnosis or lab finding, which serves as the temporal reference for aligning all other trial components. Between these events lies the observation window (W), used to assess baseline covariates and eligibility status. Eligibility criteria (E) may span both pre- and post-index periods. The baseline visit (B) captures pre-intervention assessments. The intervention phase (T) begins after the index event, followed by post-treatment follow-up visits (F) to monitor outcomes and safety. Outcome assessment (O) occurs within a bounded period at the end of the trial. TEL constraints are defined with modal operators to precisely capture temporal relationships.

- **Start Event**: The TEL expression for the start event occurring at time x is: $\exists x\,(S_x)$.
- **Index Event**: The TEL expression for the index event occurring at time $x+i$ is given by: $\exists x \exists i\,(S_x \wedge I_{x+i})$. The $x+i$ subscript ensures that Index event can occur only after the Start Event. This represents the key modeling strategy of using distinct terms to represent time dependencies, rather than using explicit comparison such as $x < y$ in other approaches.
- **Observation Window**: The observation window defines the time interval between the start event and the index event, within which prior clinical events must occur.The corresponding TEL expression is given by: $\exists x \exists i \Big[S_x \wedge I_{x+i} \wedge (\Diamond_i W)_x \Big]$.
- **Baseline Visit**: The TEL expression for the baseline visit B occurs within b units after the index event is given by: $\exists x \exists i \exists b \Big[S_x \wedge I_{x+i} \wedge (\Diamond_i W)_x \wedge (\Diamond_b B)_{x+i} \Big]$.
- **Intervention Phase**: The TEL expression for the intervention phase T occurring within t units post-baseline is given by:

$$\exists x \exists i \exists b \exists t \Big[S_x \wedge I_{x+i} \wedge (\Diamond_i W)_x \wedge (\Diamond_b B)_{x+i} \wedge (\Diamond_t T)_{x+i+b} \Big].$$

- **Follow-up visits**: The TEL expression for the follow-up visit F occurring within f units after the intervention phase is given by:

$$\exists x \exists i \exists b \exists t \exists f \Big[S_x \wedge I_{x+i} \wedge (\Diamond_i W)_x \wedge (\Diamond_b B)_{x+i} \wedge (\Diamond_t T)_{x+i+b} \wedge (\Diamond_f F)_{x+i+b+t} \Big].$$

- **Outcome Assessment**: The TEL expression for the outcome assessment O occurring within o units after the follow-up visit is given by:

$$\exists x \exists i \exists b \exists t \exists f \exists o \Big[S_x \wedge I_{x+i} \wedge (\Diamond_i W)_x \wedge (\Diamond_b B)_{x+i} \wedge (\Diamond_t T)_{x+i+b} \wedge (\Diamond_f F)_{x+i+b+t} \wedge (\Diamond_o O)_{x+i+b+t+f} \Big].$$

- **Eligibility Criteria**: Eligibility Criteria define the conditions under which a participant is deemed suitable for enrollment in the clinical trial.
  - **Inclusion Criteria (IC)**: which specify characteristics that must be present for a participant to qualify. The TEL expression is given by:

$$\exists x \exists i \exists c_1 \exists c_2 \Big[S_x \wedge I_{x+i} \wedge (\Diamond_{c_1} IC)_x \vee (\Diamond_{c_2} IC)_{x+i} \Big].$$

  - **Exclusion Criteria (EX)**, which specify conditions that, if present, disqualify a participant from participation. The TEL expression is given by:

$$\exists x \exists i \exists e_1 \exists e_2 \Big[S_x \wedge I_{x+i} \wedge (\Box_{e_1} \neg EX)_x \wedge (\Box_{e_2} \neg EX)_{x+i} \Big].$$

The full TEL expression representing the clinical trial structure temporally anchored on the start event and index event, and encoding the logical and temporal constraints across eligibility, baseline, treatment, follow-up, and outcome phases, is defined as follows:

$$\exists x \exists i \exists b \exists t \exists f \exists o \exists f \exists c_1 \exists c_2 \exists e_1 \exists e_2 \Big[S_x \wedge I_{x+i} \wedge (\Diamond_i W)_x \wedge (\Diamond_b B)_{x+i} \wedge (\Diamond_t T)_{x+i+b} \wedge (\Diamond_f F)_{x+i+b+t} \wedge (\Diamond_o O)_{x+i+b+t+f} \wedge \big((\Diamond_{c_1} IC)_x \vee (\Diamond_{c_2} IC)_{x+i} \big) \wedge \big((\Box_{e_1} \neg EX)_x \wedge (\Box_{e_2} \neg EX)_{x+i} \big) \Big].$$

5.2 TEL-Based Formalization of Cohort-based CTS

The Cohort-Based CTS models trial participants as belonging to logically distinct subpopulations, defined through a combination of base eligibility and group-specific cohort criteria. Each individual in the simulation is first evaluated against the Base Criteria to determine inclusion in the eligible study population. Then, individuals are assigned to mutually exclusive groups according to whether they satisfy the Case Criteria or Control Criteria.

- **Base Criteria**: All individuals must meet the base criteria within a defined observation window starting at the reference time t_0. These criteria typically include basic demographic filters, the length of the observation period, and the exclusion of confounders. The TEL expression is: $\exists t_0 \, (Base_{t_0})$.
- **Case Criteria**: Individuals who meet the case criteria are assigned to the case cohort. These criteria usually capture exposure to an intervention, diagnosis of a condition, or other qualifying events. The case criteria must be met after satisfying the base criteria. The TEL expression is: $\exists t_0 \exists t_1 \Big[Base_{t_0} \wedge (\Diamond_{t_1} \text{Case})_{t_0} \Big]$.
- **Control Criteria**: Individuals who meet the control criteria are assigned to the control cohort. These criteria typically indicate the absence of the exposure or condition that defines the case group. The assignment of control also follows the base criterion. The TEL expression is: $\exists t_0 \exists t_2 \Big[Base_{t_0} \wedge (\Diamond_{t_2} \text{Control})_{t_0} \Big]$.

5.3 Model-Checking of Clinical Trial Specifications

Formally, the model-checking [6, 45] process seeks to determine whether a patient record α satisfies a given clinical trial specification φ, as defined in TEL. That is, we aim to verify whether there exists a time point $i \geq 1$ such that $(\alpha, i) \models \varphi$, under the TEL semantics. This generalizes beyond traditional initial-time evaluation by enabling satisfaction to be checked at any point along a patient's longitudinal record, allowing for alignment of protocol logic with varying index events. The language defined by a TEL formula, denoted $\mathcal{L}(\varphi)$, consists of all ω-words (i.e., patient records) that satisfy the formula at some valid time index. This capability supports temporal abstraction and flexible anchoring of eligibility and trial components in real-world datasets. TEL-based model-checking provides a scalable and precise method for aligning structured clinical data with complex temporal protocol logic, improving the interpretability, consistency, and reproducibility of virtual clinical trials.

6 Case Study

To evaluate and demonstrate the practical utility of our TEL-based formalization framework, we applied it to the domain of AD clinical trials. AD trials present complex temporal structures and heterogeneous eligibility criteria, making them a representative use case for formal modeling. We developed two resources: (1) the AD Clinical Trial Ontology (ADCTO), a lightweight ontology that captures core concepts from AD-related trial protocols, and (2) a logic-based clinical trial simulation system that operationalizes TEL for temporally-aware trial design and execution. These tools enable precise specification, validation, and simulation of AD clinical trials grounded in real-world data.

6.1 AD Clinical Trial Ontology

We developed the AD Clinical Trial Ontology (ADCTO) to represent key elements frequently encountered in the eligibility criteria of Alzheimer's disease (AD) clinical trials, using data from ClinicalTrials.gov. The ontology encompasses a comprehensive range of categories, including Disease, Medication, Diagnostic Test, Procedure, Rating Criteria, Social Determinants of Health, and Fertility. Each concept was mapped to the Unified Medical Language System (UMLS) with assigned Concept Unique Identifiers (CUIs) and annotated with Observational Health Data Sciences and Informatics (OHDSI) Athena identifiers. We enriched ADCTO with annotations from resources such as the National Library of Medicine (NLM) Value Set Authority Center, the National Drug Code (NDC) directory, UMLS Terminology Services, Codify by the American Academy of Professional Coders (AAPC), and the Phenotype KnowledgeBase (PheKB). To construct a coherent hierarchy, we integrated established biomedical ontologies and classification systems, including the Disease Ontology for disease concepts and DrugBank and the Anatomical Therapeutic Chemical (ATC) classification system for medications, with refinements informed by domain experts to ensure semantic precision and relevance.

Compared to existing ontologies, ADCTO maintains a lightweight structure, with 274 classes, 278 logical axioms, and 284 declaration axioms. To evaluate its adequacy, we achieved a recall of 0.63 (the proportion of ontology-mapped elements relative to all extracted eligibility elements) and an average term frequency–inverse document frequency (TF-IDF) score of 3.91, indicating ADCTO's effectiveness in capturing the essential elements of AD eligibility criteria in a compact and computationally efficient structure.

6.2 AD Clinical Trial Simulation System

We developed the AD Clinical Trial Simulation System, a logic-based platform for designing and evaluating virtual clinical trials using real-world data. Built on the ADCTO, the system provides a formally defined and semantically consistent vocabulary to represent key trial components, including event types, eligibility criteria, interventions, outcomes, and their temporal dependencies. Users interact with a web-based interface that offers ontology-aligned dropdown menus, enabling the configuration of trial components without the need for manual coding. These selections are automatically compiled into formal representations, allowing for precise and interpretable modeling of trial logic. The simulation engine leverages TEL to capture and enforce time-dependent relationships among clinical events. All components are temporally anchored to a defined index event, enabling precise sequencing of trial activities. The system supports fixed and flexible timelines and allows retrospective simulations using observational data. It is designed to bridge the gap between descriptions of natural language protocols and formal computational models. The user interface reflects the core vision of the system, which is to democratize formal trial design by making logical and ontological structures accessible to clinical researchers without requiring expertise in logic or programming.

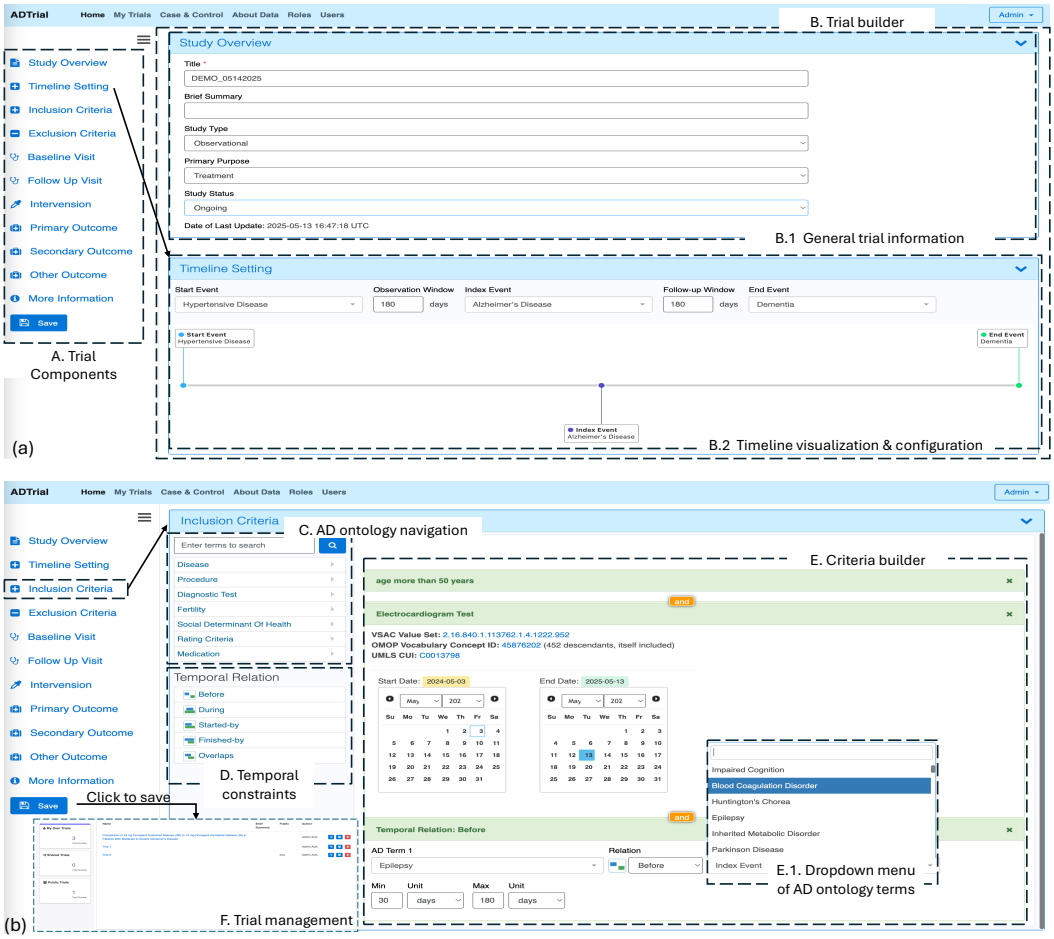


Figure 1 The interface of the AD clinical trial simulation system.

Figure 1 illustrates the user interface of our system. Figure 1.(a).A shows the components of a clinical trial and allows users to click on each element to begin configuring it. Figure 1.(a).B showcases the trial builder, with subpanels B.1 and B.2 displaying interfaces for entering trial information and configuring timeline parameters. Figure 1(b) illustrates the configuration interface for specifying Inclusion Criteria, which serves as a representative example; similar interfaces are employed across other trial components to ensure consistency and usability. Figure 1.(b).C presents the ontology navigation interface, enabling users to explore standardized ADCTO terms. Figure 1.(b).D and E highlight the specification of temporal constraints and eligibility criteria, with E.1 featuring a dropdown menu populated by ontology-aligned terms. Finally, Figure 1.(b).F depicts the trial management interface, where users can review and save configured trials. Configured trials are automatically translated into formal TEL expressions and executed against real-world EHR datasets to simulate trial behavior and validate protocol feasibility. This enables researchers to assess cohort sizes, timing constraints, and outcome trajectories in silico. The system was implemented using Ruby on Rails [15] with a Model-View-Controller (MVC) architecture and used MongoDB [3] as the backend database to support scalable, document-based storage of evolving trial representations and large-scale simulations.

The interface serves as an authoring tool and compilation engine, generating formal TEL models from high-level trial designs. Each session outputs a reusable TEL expression anchored by start and index events, enriched with modal and quantified constraints. These specifications support downstream tasks such as simulation, model-checking, and validation against EHR or registry data. The platform enforces an ontology-driven workflow that enhances semantic clarity, reduces variability, and promotes reuse across studies. Aligned with standardized vocabularies, it enables integration with external data models and supports automated reasoning. Designed for longitudinal datasets, the system facilitates rapid prototyping, cohort selection, and eligibility screening. Each simulation is saved as a machine-readable specification that adheres to FAIR (Findable, Accessible, Interoperable, Reusable) data principles, supporting reproducibility, version control, and collaboration across institutions.

7 Discussion

The simulation system implements a formal, TEL-based approach for specifying and simulating clinical trials, supporting ontology-driven configuration and automatic translation of protocol elements into logical expressions. The system represents an early-stage prototype designed to establish the feasibility of logic-based trial modeling and virtual execution. Development priorities include extending the reasoning engine, improving user interface interactivity, supporting integration with large-scale observational data platforms, and enabling export of TEL specifications in interoperable formats aligned with common data models. These enhancements will allow the platform to support complex protocol designs and to enable real-time simulation of cohort definitions and outcome trajectories. The current system implements a pipeline that maps user interface inputs directly to TEL formulas. Trial designers specify protocol elements using dropdown menus populated from ADCTO, including time constraints and event types. These inputs are compiled into formal TEL expressions through structured templates that preserve both syntactic correctness and semantic alignment. Patient medical records are represented as temporally indexed sequences of coded events, enabling the TEL engine to evaluate whether a given specification is satisfied at any admissible index point in the record.

TEL is undecidable in its general form [48]; however, the fragment used in this study avoids intractable features such as unrestricted quantifier alternation and deeply nested temporal operators. It relies on existential quantification and time-bounded modal constructs, allowing finite evaluation over RWD. We have developed and preliminarily validated algorithms for this purpose, with several manuscripts in preparation detailing their design, implementation, and performance benchmarking. These works will also introduce interval-based extensions to model durations between temporally disjoint events. Formal characterization of the fragment's decidability and complexity remains a key direction for ensuring scalability.

In its current form, TEL supports point-based reasoning with relative offsets and can explicitly model upper bounds on absolute time intervals between events. Using the modal operators $\Diamond_t$ (diamond) and $\Box_t$ (box) with time parameters, TEL can precisely define both lower and upper bounds for the occurrence of events. For example, $\Box_t\varphi$ ensures that a condition φ holds continuously for a duration of t time units, while $\Diamond_t\varphi$ guarantees that φ becomes true at some point within t units. These constructs provide direct control over temporal constraints, which is critical in clinical trial protocols where the timing of events, such as symptom onset, dosing intervals, or outcome assessments, is clinically significant. This expressiveness allows TEL to model temporally bounded windows of clinical relevance and supports fine-grained specification of interval-based requirements.

TEL has focused on clinical trial modeling, but there is a range of related efforts that address temporal representations in healthcare processes, such as clinical guidelines, care pathways, and decision-intensive workflows. Relevant work includes temporal workflow models for clinical pathways [7], decision modeling frameworks for chronic care management [8], and Business Process Model and Notation (BPMN)-based systems [24] for perioperative processes. Other studies have explored mobile delivery of guideline-based decision support [38], ontology-based search and reasoning over clinical protocols [27], and logic-based eligibility verification for trials [23]. Metric and interval temporal logic approaches have been applied to model timing constraints in clinical practice guidelines [37, 47]. Our approach complements these efforts by offering a logic tailored for patient-level temporal event alignment with computable representations integrated into simulation engines and ontology-driven user interfaces.

Finally, TEL captures temporal and structural dependencies, but clinical trials often involve normative constraints. Deontic logic provides a formal foundation for representing requirements such as obligations, prohibitions, and permissions. Integrating deontic logic into the TEL framework may enable richer modeling of regulatory rules, protocol compliance, and adaptive trial conditions. This direction will enhance the system's ability to represent real-world clinical scenarios that involve both temporal precision and rule-governed behaviors.

8 Conclusion

This study presents a TEL-based framework for modeling clinical trials through precise, formal logic representations of temporal relationships among clinical events. It addresses key limitations in current trial design, including the lack of formal temporal reasoning and unstructured protocol specifications. We developed a logic-based simulation system and the ADCT Ontology to standardize trial components and support formal reasoning. The system includes an ontology-driven user interface that enables domain experts to configure trials without coding, facilitating reproducible virtual trial design, feasibility analysis, and cohort simulation. The methodology generalizes to other domains requiring semantically rigorous and temporally aligned trial models. TEL has limited direct support from widely available verification or modeling platforms, and future work will focus on developing dedicated implementations to enhance its adoption and usability.

References

1 Klaus-Peter Adlassnig, Carlo Combi, Amar K Das, Elpida T Keravnou, and Giuseppe Pozzi. Temporal representation and reasoning in medicine: Research directions and challenges. *Artificial intelligence in medicine*, 38(2):101–113, 2006. doi:10.1016/J.ARTMED.2006.10.001.

2 Franz Baader, Stefan Borgwardt, Patrick Koopmann, Ana Ozaki, and Veronika Thost. Metric temporal description logics with interval-rigid names. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–46, 2020. doi:10.1145/3399443.

3 Shannon Bradshaw, Eoin Brazil, and Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, Inc., 2019.

4 Zhaoyi Chen, Hansi Zhang, Yi Guo, Thomas J George, Mattia Prosperi, William R Hogan, Zhe He, Elizabeth A Shenkman, Fei Wang, and Jiang Bian. Exploring the feasibility of using real-world data from a large clinical data research network to simulate clinical trials of alzheimer's disease. *NPJ digital medicine*, 4(1):84, 2021. doi:10.1038/S41746-021-00452-1.

5 Shein-Chung Chow and Mark Chang. Adaptive design methods in clinical trials–a review. *Orphanet journal of rare diseases*, 3:1–13, 2008.

6 Edmund M Clarke. Model checking. In *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*, pages 54–56. Springer, 1997.

7 Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(9):1182–1203, 2014. doi:10.1109/TSMC.2014.2300055.

8 Carlo Combi, Barbara Oliboni, Alessandro Zardini, and Francesca Zerbato. A methodological framework for the integrated design of decision-intensive care pathways—an application to the management of copd patients. *Journal of Healthcare Informatics Research*, 1(2):157–217, 2017. doi:10.1007/S41666-017-0007-4.

9 Dario Della Monica, Valentin Goranko, Angelo Montanari, Guido Sciavicco, et al. Interval temporal logics: a journey. *Bulletin of EATCS*, 3(105), 2013.

10 Kevin Donnelly et al. Snomed-ct: The advanced terminology and coding system for ehealth. *Studies in health technology and informatics*, 121:279, 2006.

11 Scott R Evans, Dianne Paraoan, Jane Perlmutter, Sudha R Raman, John J Sheehan, and Zachary P Hallinan. Real-world data for planning eligibility criteria and enhancing recruitment: recommendations from the clinical trials transformation initiative. *Therapeutic Innovation & Regulatory Science*, 55(3):545–552, 2021.

12 Tom Faciszewski, Ron Jensen, and Richard L Berg. Procedural coding of spinal surgeries (cpt-4 versus icd-9-cm) and decisions regarding standards: a multicenter study. *Spine*, 28(5):502–507, 2003.

13 Laura Giordano, Paolo Terenziani, Alessio Bottrighi, Stefania Montani, and Loredana Donzella. Model checking for clinical guidelines: an agent-based approach. In *Amia annual symposium proceedings*, volume 2006, page 289, 2006.

14 Valentin Goranko and Antje Rumberg. Temporal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2020 edition, 2020. URL: https://plato.stanford.edu/entries/logic-temporal/.

15 Michael Hartl. *Ruby on rails tutorial: learn Web development with rails*. Pearson Education, 2013.

16 JA Hirsch, G Nicola, G McGinty, RW Liu, RM Barr, MD Chittle, and L Manchikanti. Icd-10: history and context. *American Journal of Neuroradiology*, 37(4):596–599, 2016.

17 Yan Huang, Xiaojin Li, Deepa Dongarwar, Hulin Wu, and Guo-Qiang Zhang. Data mining pipeline for covid-19 vaccine safety analysis using a large electronic health record. *AMIA Summits on Translational Science Proceedings*, 2023:271, 2023.

18 JM Juarez, M Campos, A Morales, J Palma, and R Marin. Applications of temporal reasoning to intensive care units. *Journal of Healthcare Engineering*, 1(4):615–636, 2010.

19 Hui C Kimko and Stephen B Duffull. *Simulation for designing clinical trials*. Marcel Dekker Incorporated, 2002.

20 K Lee, Y Mai, Z Liu, K Raja, T Jun, M Ma, T Wang, L Ai, E Calay, W Oh, et al. Criteriamapper: establishing the automatic identification of clinical trial cohorts from electronic health records by matching normalized eligibility criteria and patient clinical characteristics. *Scientific Reports*, 14(1):25387, 2024.

21 Fang Li, Jingcheng Du, Yongqun He, Hsing-Yi Song, Mohcine Madkour, Guozheng Rao, Yang Xiang, Yi Luo, Henry W Chen, Sijia Liu, et al. Time event ontology (teo): to support semantic representation and reasoning of complex temporal relations of clinical events. *Journal of the American Medical Informatics Association*, 27(7):1046–1056, 2020. doi:10.1093/JAMIA/OCAA058.

22 Xiaojin Li, Shiqiang Tao, Samden D Lhatoo, Licong Cui, Yan Huang, Johnson P Hampson, and Guo-Qiang Zhang. A multimodal clinical data resource for personalized risk assessment of sudden unexpected death in epilepsy. *Frontiers in big Data*, 5:965715, 2022. doi:10.3389/FDATA.2022.965715.

23 Deryle W Lonsdale, Clint Tustison, Craig G Parker, and David W Embley. Assessing clinical trial eligibility with logic expression queries. *Data & Knowledge Engineering*, 66(1):3–17, 2008. doi:10.1016/J.DATAK.2007.07.005.

24 Matteo Mantovani, Andrea Caravati, Giuseppe Pozzi, Carlo Combi, and Roberto Salvia. A bpmn-based framework to manage eras-inspired pathway for patients undergoing pancreatic surgery. In *2023 IEEE 11th International Conference on Healthcare Informatics (ICHI)*, pages 21–31. IEEE, 2023. doi:10.1109/ICHI57859.2023.00015.

25 Clement J McDonald, Stanley M Huff, Jeffrey G Suico, Gilbert Hill, Dennis Leavelle, Raymond Aller, Arden Forrey, Kathy Mercer, Georges DeMoor, John Hook, et al. Loinc, a universal standard for identifying laboratory observations: a 5-year update. *Clinical chemistry*, 49(4):624–633, 2003.

26 Julia E McGuinness, Gauri Bhatkhande, Jacquelyn Amenta, Thomas Silverman, Jennie Mata, Ashlee Guzman, Ting He, Jill Dimond, Tarsha Jones, Rita Kukafka, et al. Strategies to identify and recruit women at high risk for breast cancer to a randomized controlled trial of web-based decision support tools. *Cancer Prevention Research*, 15(6):399–406, 2022.

27 Robert Moskovitch and Yuval Shahar. Vaidurya: a multiple-ontology, concept-based, context-sensitive clinical-guideline search engine. *Journal of Biomedical Informatics*, 42(1):11–21, 2009. doi:10.1016/J.JBI.2008.07.003.

28 Stuart J Nelson, Kelly Zeng, John Kilbourne, Tammy Powell, and Robin Moore. Normalized names for clinical drugs: Rxnorm at 6 years. *Journal of the American Medical Informatics Association*, 18(4):441–448, 2011. doi:10.1136/AMIAJNL-2011-000116.

29 Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 411–425. Springer, 2006. doi:10.1007/11691372_27.

30 Martin J O'Connor, Ravi D Shankar, David B Parrish, and Amar K Das. Knowledge-data integration for temporal reasoning in a clinical trial system. *International journal of medical informatics*, 78:S77–S85, 2009. doi:10.1016/J.IJMEDINF.2008.07.013.

31 Irene Petersen, Ian Douglas, and Heather Whitaker. Self controlled case series methods: an alternative to standard epidemiological study designs. *bmj*, 354, 2016.

32 Steven Piantadosi. *Clinical trials: a methodologic perspective*. John Wiley & Sons, 2024.

33 Amir Pnueli. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*, pages 46–57. ieee, 1977. doi:10.1109/SFCS.1977.32.

34 Rachel L Richesson, James E Andrews, and Kate Fultz Hollis. *Clinical research informatics*. Springer, 2012.

35 James R Rogers, Junghwan Lee, Ziheng Zhou, Ying Kuen Cheung, George Hripcsak, and Chunhua Weng. Contemporary use of real-world data for clinical trial conduct in the united states: a scoping review. *Journal of the American Medical Informatics Association*, 28(1):144–154, 2021. doi:10.1093/JAMIA/OCAA224.

36 James R Rogers, Jovana Pavisic, Casey N Ta, Cong Liu, Ali Soroush, Ying Kuen Cheung, George Hripcsak, and Chunhua Weng. Leveraging electronic health record data for clinical trial planning by assessing eligibility criteria's impact on patient count and safety. *Journal of biomedical informatics*, 127:104032, 2022. doi:10.1016/J.JBI.2022.104032.

37 Guido Sciavicco, Jose M Juarez, and Manuel Campos. Quality checking of medical guidelines using interval temporal logics: A case-study. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 158–167. Springer, 2009.

38 Erez Shalom, Ayelet Goldstein, Elior Ariel, Moshe Sheinberger, Valerie Jones, Boris Van Schooten, and Yuval Shahar. Distributed application of guideline-based decision support through mobile devices: Implementation and evaluation. *Artificial Intelligence in Medicine*, 129:102324, 2022. doi:10.1016/J.ARTMED.2022.102324.

39 Ravi D Shankar, Susana B Martins, Martin J O'Connor, David B Parrish, and Amar K Das. Representing and reasoning with temporal constraints in clinical trials using semantic technologies. In *Biomedical Engineering Systems and Technologies: International Joint Conference, BIOSTEC 2008 Funchal, Madeira, Portugal, January 28-31, 2008 Revised Selected Papers 1*, pages 520–530. Springer, 2009. doi:10.1007/978-3-540-92219-3_39.

40 Weiyi Sun, Anna Rumshisky, and Ozlem Uzuner. Temporal reasoning over clinical text: the state of the art. *Journal of the American Medical Informatics Association*, 20(5):814–819, 2013. doi:10.1136/AMIAJNL-2013-001760.

41 Tatsunori Tanaka, Fi Zheng, Kai Sato, Zhifeng Li, Yuanyun Zhang, and Shi Li. Temporal entailment pretraining for clinical language models over ehr data. *arXiv preprint arXiv:2504.18128*, 2025. doi:10.48550/arXiv.2504.18128.

42 Cui Tao, Harold R Solbrig, and Christopher G Chute. Cntro 2.0: a harmonized semantic web ontology for temporal relation inferencing in clinical narratives. *AMIA summits on translational science proceedings*, 2011:64, 2011.

43 Cui Tao, Wei-Qi Wei, Harold R Solbrig, Guergana Savova, and Christopher G Chute. Cntro: a semantic web ontology for temporal relation inferencing in clinical narratives. In *AMIA annual symposium proceedings*, volume 2010, page 787, 2010.

44 U.S. National Library of Medicine. Effect of Different Doses of SAR110894 on Cognition in Patients With Mild to Moderate Alzheimer's Disease on Donepezil. https://clinicaltrials.gov/study/NCT01266525, 2011. ClinicalTrials.gov Identifier: NCT01266525.

45 Moshe Y Vardi. Model checking for database theoreticians. In *International Conference on Database Theory*, pages 1–16. Springer, 2005. doi:10.1007/978-3-540-30570-5_1.

46 Heather J Whitaker and Yonas Ghebremichael-Weldeselassie. Self-controlled case series methodology. *Annual review of statistics and its application*, 6(1):241–261, 2019.

47 Morteza Yousef Sanati, Wendy MacCaull, and Thomas SE Maibaum. Analyzing clinical practice guidelines using a decidable metric interval-based temporal logic. In *International Symposium on Formal Methods*, pages 611–626. Springer, 2014.

48 Guo-Qiang Zhang. Temporal ensemble logic. *arXiv preprint arXiv:2408.14443*, 2024. doi:10.48550/arXiv.2408.14443.

49 Guo-Qiang Zhang, Licong Cui, Remo Mueller, Shiqiang Tao, Matthew Kim, Michael Rueschman, Sara Mariani, Daniel Mobley, and Susan Redline. The national sleep research resource: towards a sleep data commons. *Journal of the American Medical Informatics Association*, 25(10):1351–1358, 2018. doi:10.1093/JAMIA/OCY064.

QualiNet: Acquiring Bird's Eye View Qualitative Spatial Representation from 2D Images in Automated Vehicle Perception

Nassim Belmecheri
Simula Research Laboratory, Oslo, Norway

Abstract

We present QualiNet, an end-to-end deep learning framework that acquires Bird's Eye View (BEV) qualitative spatial relations directly from 2D images, eliminating the need for depth sensors. The system combines 2D object detection, masking, and classification to infer Rectangle Algebra (RA) and Qualitative Distance Calculus (QDC) relations. Evaluated on NuScenes and PandaSet datasets, QualiNet achieves 91% accuracy for RA, 80% for QDC, and 99% top-2 accuracy, demonstrating robust performance for automated vehicle perception.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Computing methodologies → Spatial and physical reasoning; Computing methodologies → Scene understanding

Keywords and phrases Qualitative Spatial Representation, Deep Learning, Computer vision, Qualitative Scene Understanding, Spatio-temporal representation and reasoning models (including moving objects tracking)

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.14

Category Short Paper

Supplementary Material *Software (Source code)*: `https://github.com/nassimbel/QualiNet.git` [3] archived at `swh:1:dir:a4900663aeb84632699b0217f7a7f98014466c00`

Funding This work is funded by the European Commission through the AI4CCAM project (Trustworthy AI for Connected, Cooperative Automated Mobility) under grant agreement No 101076911.

Acknowledgements I would like to thank my colleagues Arnaud Gotlieb, Nadjib Lazaar and Helge Spieker for the fruitful discussions and continuous support.

1 Introduction

Automated vehicle perception traditionally relies on quantitative methods that struggle with complex real-world scenarios [1]. Qualitative representations offer a promising alternative by capturing relative spatial relationships through calculi like Rectangle Algebra (RA) and Qualitative Distance Calculus (QDC) [15]. These methods simplify complex spatial information while aligning with human reasoning patterns [16, 2].

Current approaches for building qualitative representations [8, 4] typically require expensive sensors (LiDAR, depth cameras) and significant computational resources. We present QualiNet, an end-to-end deep learning framework that acquires Bird's Eye View (BEV) qualitative representations directly from 2D images using object detection and classification. Our method is validated on NuScenes [6] and PandaSet [18] datasets.

2 Related Work

Recent advances have demonstrated the effectiveness of qualitative representations across multiple domains. In action recognition, qualitative state transitions [19, 14] and relation chains [12] have proven valuable for capturing action semantics. For autonomous driving,

neuro-symbolic integration [17] and BEV qualitative representations [2] enhance both scene understanding and system explainability. Spatial knowledge acquisition methods show particular diversity, ranging from implicit templates [9] to force histograms [5] and hybrid symbolic-neural approaches [10]. While current methods typically depend on 3D sensors [13], our QualiNet system uniquely acquires BEV spatial relations directly from 2D images, eliminating the need for specialized depth sensing hardware.

Background

Perception in Automated Vehicles

Perception is crucial for automated vehicles to understand and interact with their environment. This involves processing data from various sensors like LiDAR, radar, and cameras [1].

2D perception, primarily using cameras, analyzes images to identify objects but lacks depth information, making distance and size estimation challenging [11].

3D perception overcomes this limitation by incorporating depth information from stereo cameras or LiDAR. This enables accurate spatial understanding and object localization, vital for safe and reliable autonomous navigation [7].

Qualitative Spatial Calculus

A qualitative calculus operates over domain $\mathcal{D}$ (e.g., $\mathbb{R}^2$) with binary relations $\Gamma = \{r_1, \ldots, r_m\}$ that are jointly exhaustive and pairwise disjoint. We employ:

- **QDC** [15]: Distance relations (very close, close, normal, far)
- **RA** [15]: Rectangle relations from Allen's Interval Algebra (before, meets, overlaps, etc.) on x/y axes

A scene $S = (V, O, R)$ consists of frames V, objects O, and their relations R.

Transforming Images into BEV Qualitative Constraint Networks

We represent detected objects and their spatial relations as a qualitative graph $\mathcal{G} = (\mathcal{O}, \mathcal{R})$, where $\mathcal{O}$ is the set of objects and $\mathcal{R}$ their qualitative relations from language Γ. Each object belongs to a single category (e.g., car, pedestrian).

Using the ego vehicle as reference object o_0, we construct a star graph $\mathcal{G}^* = (\mathcal{O}, \mathcal{R}^*)$ with relations between o_0 and other objects. The complete graph $\mathcal{G}$ is built through relation composition: $R_{ij} = R_{0i} \circ R_{0j} \quad \forall (o_i, o_j) \in \mathcal{O}^2, i \neq j$ using path consistency enforcement [15] until convergence.

Image to Relation Data Construction

We denote by $\mathcal{D} = \{(\mathcal{I}_i, \mathcal{R}_i^*, o_0)\}_{i=1}^N$ a dataset of tuples consisting of 2D images $\mathcal{I}_i$, qualitative relations $\mathcal{R}_i^*$ between BEV detected objects and the reference object o_0 (ego vehicle). This dataset serves as the training data for QualiNet, enabling the model to learn the mapping between visual information and qualitative spatial relations.

Consider `I2BEVQR` as a function that maps a 2D image $\mathcal{I}$ to the set of qualitative relations $\mathcal{R}^*$ between the detected objects and the reference object o_0. `I2BEVQR` takes a 2D image $\mathcal{I}$ as input and outputs the set of qualitative relations $\mathcal{R}^*$ between the detected objects and the reference object o_0. $\mathcal{R}^*$ represents the labels for $\mathcal{I}$. For each image, the function extracts the objects with their categories and bounding boxes, including the ego-vehicle. It

then constructs a star graph with the ego-vehicle as the central node and other objects as peripheral nodes. Using the QXG-builder tool [4, 2] and the specified algebra, it determines the qualitative spatial relations between the ego-vehicle and each object. These relations, along with the image and the ego-vehicle information, are added to the dataset.

Learning to Acquire Qualitative Relations

We define $\mathcal{M}$ as a function that takes a 2D image $\mathcal{I}$ and returns a binary mask M indicating the presence of detected objects: $\mathcal{M} : \mathcal{I} \rightarrow M$ where M is a binary matrix of dimensions $H \times W$ such that:

$$M(i,j) = \begin{cases} 1 & \text{if pixel } (i,j) \text{ belongs to a detected object} \\ 0 & \text{otherwise} \end{cases}$$

This masking process helps to focus the attention of the deep learning model on the relevant regions of the image, improving the accuracy and efficiency of spatial relation extraction.

The QualiNet (Algorithm 1) takes as input the training dataset, learning rate, number of epochs, and predefined architectures for the CNN and the MLPclassifier. It outputs a trained QualiNet model.

Algorithm 1 QualiNet Training Algorithm.

Input: Dataset $\mathcal{D} = \{(\mathcal{I}_i, \mathcal{R}_i^*, o_{0i})\}_{i=1}^N$, Epochs E, CNN architecture, MLPClassifier architecture
Output: Trained QualiNet model $\mathcal{F}_{\text{QualiNet}}$
Initialize CNN with a predefined architecture
Initialize Classifier with a predefined architecture
for *epoch* $\leftarrow 1$ **to** E **do**
  for $(\mathcal{I}, \mathcal{R}^*, o_0) \in \mathcal{D}$ **do**
    for $o_j \in \mathcal{O} \setminus \{o_0\}$ **do**
      $M \leftarrow \mathcal{M}(\mathcal{I}, o_j)$
      $F_j \leftarrow \text{CNN}(M)$
      $\hat{R}_{0j} \leftarrow \text{MLPClassifier}(F_j)$
      $L \leftarrow \text{Loss}(\hat{R}_{0j}, \mathcal{R}_j^*)$
      Update parameters of CNN and Classifier using gradient descent
return $\mathcal{F}_{\text{QualiNet}}$

3 Experiments

This section outlines the experimental setup and results for evaluating QualiNet's performance. Since our approach is novel, there are no direct baselines for comparison. Instead, we focus on showcasing QualiNet's capabilities and analyzing its behavior. Our evaluation aims to investigate how accurate is QualiNet's top predictions (Top 1 and Top 2 accuracy)?

We used the NuScenes [6] dataset for evaluation. NuScenes includes diverse urban scenes captured from multiple sensors. We utilized both the full dataset (NuScenes-Large) and a smaller subset (NuScenes-mini).

QualiNet is implemented in PyTorch using a ResNet-152 CNN for feature extraction and an MLP for classification. The model is trained with SGD and a learning rate scheduler. Data is split 70:30 for training and validation, and performance is evaluated over five runs using Top 1 and Top 2 accuracy.

The original dataset exhibited severe imbalance: QDC "far" (42% samples) vs "very close" (3%). After augmentation, all relations have 500±20 samples.

We assess QualiNet's performance using the following metrics:

Accuracy (Top 1): Percentage of correct top predictions. **Accuracy (Top 2):** Percentage of cases where the correct relation is among the top two predictions.

All datasets were transformed using the *I2BEV* function to generate the training and testing data. During data construction, impossible RA relations for each camera were removed to ensure the training data accurately reflects observable spatial relationships.

The code for QualiNet and the experiments presented in this paper is available at: `https://drive.google.com/drive/folders/1K9ViuyM4s_IwkcaCd3b1KYAhh0P3j1BH?usp=sharing`

3.1 Results

The results presented in this section are averaged over all the datasets test sets used in the evaluation.

Table 1 Top-1 and Top-2 Accuracy by camera: CF (Front), CFL (Front-Left), CFR (Front-Right), CB (Back), CBL (Back-Left), CBR (Back-Right).

Sensor	Relation	Top-1 Accuracy	Top-2 Accuracy
CF	RA	0.93	0.98
CFL	RA	0.92	0.96
CFR	RA	0.92	0.96
CB	RA	0.94	0.99
CBL	RA	0.88	0.93
CBR	RA	0.89	0.96
CF	QDC	0.78	0.94
CFL	QDC	0.78	0.93
CFR	QDC	0.77	0.93
CB	QDC	0.77	0.94
CBL	QDC	0.78	0.94
CBR	QDC	0.78	0.93

Table 1 presents the Top-1 and Top-2 accuracy of QualiNet for different camera sensors and relation types. As shown, the Top-1 accuracy ranges from 76% to 94%. However, the Top-2 accuracy is consistently higher, ranging from 90% to 99%. This indicates that even when QualiNet's top prediction is not the exact ground truth relation, it often includes the true relation within its top two guesses. This observation answers **RQ1** by demonstrating that QualiNet exhibits high accuracy in predicting spatial relations, particularly when considering the Top-2 accuracy, which is important for building satisfiable qualitative graphs. The model has some limitation that will be addressed in future works. The limitations include: **Small/Distant Objects:** Performance degrades for objects <50px in size (15% accuracy drop) due to limited visual information. **Detection Sensitivity:** Sensitive to object detection errors (10% error propagation to relation classification).

4 Conclusion

We presented QualiNet, a novel framework for acquiring BEV qualitative spatial relations directly from 2D images, eliminating the need for expensive depth sensors. Experimental results demonstrated high accuracy (>90% Top-2) across multiple relation types and camera views, with 92% of predicted graphs being fully consistent. Future work will extend QualiNet to dynamic scenes and enhanced occlusion handling.

References

1 C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021. doi:10.1016/J.ESWA.2020.113816.

2 N. Belmecheri, A. Gotlieb, N. Lazaar, and H. Spieker. Toward trustworthy automated driving through qualitative scene understanding and explanations. *SAE Int. J. CAV*, 8(1), 2024. doi:10.4271/12-08-01-0003.

3 Nassim Belmecheri. QualiNet. Software, swhId: swh:1:dir:a4900663aeb84632699b0217f7a7f98014466c00 (visited on 2025-09-18). URL: https://github.com/nassimbel/QualiNet.git, doi:10.4230/artifacts.24755.

4 Nassim Belmecheri, Arnaud Gotlieb, Nadjib Lazaar, and Helge Spieker. Acquiring qualitative explainable graphs for automated driving scene interpretation. *arXiv*, August 2023. arXiv:2308.12755.

5 Rajkumar Bondugula, Pascal Matsakis, and James M Keller. Force histograms and neural networks for human-based spatial relationship generalization. In *Proceedings of the International Conference on Neural Networks and Computational Intelligence*, pages 185–190, 2004.

6 Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv*, 2019. arXiv:1903.11027.

7 Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

8 AG Cohn, C Burbridge, DC Hogg, M Alomari, N Hawes, P Duckworth, P Lightbody, Y Gatsoulis, Christian Dondrup, and Marc Hanheide. Qsrlib: a software library for online acquisition of qualitative spatial relations from video. In 29^{th} *International Workshop on Qualitative Reasoning (QR'16). New York City*, 2016.

9 Guillem Collell, Luc Van Gool, and Marie-Francine Moens. Acquiring common sense spatial knowledge through implicit spatial templates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32(1), 2018.

10 Ivan Donadello, Luciano Serafini, and Artur S d'Avila Garcez. Logic tensor networks for semantic image interpretation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1596–1602, 2017. doi:10.24963/IJCAI.2017/221.

11 Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. doi:10.1109/CVPR.2012.6248074.

12 Hua Hua, Dongxu Li, Ruiqi Li, Peng Zhang, Jochen Renz, and Anthony Cohn. Towards explainable action recognition by salient qualitative spatial object relation chains. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-22)*, 2022. doi:10.1609/aaai.v36i5.20513.

13 Sang Uk Lee, Sungkweon Hong, Andreas Hofmann, and Brian Williams. Qsrnet: Estimating qualitative spatial representations from rgb-d images. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8057–8064, 2020. doi:10.1109/IROS45743.2020.9341452.

14 Dongxu Li, Enrico Scala, Patrik Haslum, and Sergiy Bogomolov. Effect-abstraction based relaxation for linear numeric planning. In *IJCAI*, pages 4787–4793, 2018. doi:10.24963/IJCAI.2018/665.

15 Jochen Renz and Bernhard Nebel. Qualitative Spatial Reasoning Using Constraint Calculi. In *Handbook of Spatial Logics*, pages 161–215. Springer, 2007. doi:10.1007/978-1-4020-5587-4_4.

16 Jakob Suchan, Mehul Bhatt, and Srikrishna Varadarajan. Commonsense visual sensemaking for autonomous driving – on generalised neurosymbolic online abduction integrating vision and semantics. *Artificial Intelligence*, 299:103522, 2021. doi:10.1016/j.artint.2021.103522.

17 Jakob Suchan, Mehul Bhatt, and Srikrishna Varadarajan. Commonsense visual sensemaking for autonomous driving – on generalised neurosymbolic online abduction integrating vision and semantics. *Artificial Intelligence*, 295:103458, 2021.

18 Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3095–3101. IEEE, 2021.

19 Tao Zhuo, Zhiyong Cheng, Peng Zhang, Yongkang Wong, and Mohan Kankanhalli. Explainable video action reasoning via prior knowledge and state transitions. In *Proceedings of the 27th acm international conference on multimedia*, pages 521–529, 2019. doi:10.1145/3343031.3351040.

The Temporal Vadalog System

Luigi Bellomarini
Bank of Italy, Rome, Italy

Livia Blasi
TU Wien, Vienna, Austria
Bank of Italy, Rome, Italy

Markus Nissl
TU Wien, Vienna, Austria

Emanuel Sallinger
TU Wien, Vienna, Austria
University of Oxford, Oxford, UK

Abstract

The recent resurgence of the Datalog language in the Knowledge Representation and Reasoning community has paved the way for a very promising proposal for temporal extension. DatalogMTL (Datalog with Metric Temporal Operators) is a language that offers a good trade-off between computational complexity and expressive power. However, existing implementations are still preliminary or prototypical. In this extended abstract, we give a brief overview of Temporal Vadalog, a system supporting reasoning over DatalogMTL programs built upon an engineered architecture and adopted in production scenarios in the financial setting.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Information systems → Database management system engines

Keywords and phrases temporal reasoning, Datalog, DatalogMTL

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.15

Category Short Paper

Related Version This is an extended abstract of a paper previously published on Theory and Practice of Logic Programming.
Full Version: `https://doi.org/10.1017/S1471068425000018` [4]

Funding This work has been supported by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201, 10.47379/VRG18013, 10.47379/NXT22018]; and by the Austrian Science Fund (FWF) 10.55776/COE12.

1 Introduction

The adoption of Datalog-based systems for *Knowledge Representation and Reasoning* (KRR) and their growing application in production settings such as the financial space [1] are going hand in hand with a wave of research into extensions to Datalog, known as the Datalog$^{\pm}$ family [10]. The aim is to strike a good balance between computational complexity and high expressivity, that is, to incorporate the features needed for real-world applications. In this context, a key requirement in KRR is native support for time through the reasoning process.

The recent introduction of the extension of Datalog with *metric temporal operators*, named *DatalogMTL* [8], along with the good computational characteristics of its fragments [16, 17, 18], brought the potential of temporal reasoning to real-world applications, from transport and robotics, from healthcare to finance. However, the integration of such capabilities into a production-ready system requires functional and architectural characteristics that

ensure rigorous and efficient implementation of the language in computation and memory footprints. To our knowledge, all currently existing DatalogMTL implementations remain at a prototypical stage or are designed primarily for exploratory research [7, 19].

Contribution. Motivated by the need to adopt temporal reasoning in high-stake financial settings for the Central Bank of Italy, in this extended abstract [4], we introduce the Temporal Vadalog System, the temporal extension to the Vadalog System, a state-of-the-art Datalog-based reasoner [2].

Overview. Due to space constraints, the remainder of the paper provides an example-based glimpse into the functional and non-functional approaches to implementing DatalogMTL in Vadalog, specifically in Section 2, focussing on joins, and in Section 3, which describes the architecture. Some preliminaries about DatalogMTL can be found in Appendix A. For a complete overview, the reader is referred to the full paper [4].

2 Time-Aware Joins and Time Series Operators by Example

To briefly illustrate the capabilities of the system, we proceed by example with a realistic – albeit simplified – case from the financial domain.

▶ **Example 1.** A Financial Intelligence Unit (FIU) is an agency responsible for collecting information about suspicious financial activity to support investigations into money laundering or terrorism financing. They aim to identify companies with abnormal spending behavior and investigate the corporate networks they belong to. We describe the scenario by a database $\mathcal{D}$ of temporally annotated facts and the following set Π of DatalogMTL rules.

$$\boxminus_{[0,3]} \mathit{suspiciousActivity}(C), \neg \diamondminus_{[0,7]} \mathit{markedAsSafe}(C) \rightarrow \mathit{directSuspicious}(C) \tag{1}$$

$$\mathit{directSuspicious}(C) \rightarrow \mathit{suspicious}(C) \tag{2}$$

$$\mathit{suspicious}(C), \mathit{linked}(C, O) \rightarrow \mathit{suspicious}(O) \tag{3}$$

We assume that the reader is familiar with the logic-based formulation of Datalog syntax, with the body of a rule (left-hand side) being the implication premise, a logical conjunction of predicate over terms (i.e., constants or variables), while the head (right-hand side) is the implication conclusion. Ignoring the temporal angle, Rule 1 describes the conditions (company C involved in *suspiciousActivity* while not being *markedAsSafe*) under which a company C is marked as *directSuspicious*, while Rules 2-3 recursively mark every *directSuspicious* company C as *suspicious* (Rule 2) and then proceed to mark every other company o that is *linked* to *suspicious* company C as *suspicious* as well (Rule 3). Coming back to Rule 1, we see how the temporal aspect is essential: a company that is *markedAsSafe* in a distant past is not thereby exempt from having its current *suspiciousActivity* scrutinized. Metric temporal operators are helpful here: assuming day granularity, when prefixed with $\boxminus_{[0,3]}$, the atom *suspiciousActivity* only holds if the atom itself has continuously held in the interval $[t-0, t-3]$ if evaluated at t – the *suspiciousActivity* continuously held for the previous 3 days – while with $\diamondminus_{[0,7]}$, *markedAsSafe* only holds if the clearance occurred sometime between $[t-0, t-7]$. Let us consider database $\mathcal{D}$:

$$\mathcal{D} =$$

$$\{\mathit{suspiciousActivity}(\mathrm{A})@[\mathtt{May}-\mathtt{01}, \mathtt{May}-\mathtt{04}], \mathit{markedAsSafe}(\mathrm{A})@[\mathtt{Apr}-\mathtt{30}, \mathtt{Apr}-\mathtt{30}],$$
$$\mathit{suspiciousActivity}(\mathrm{B})@[\mathtt{May}-\mathtt{02}, \mathtt{May}-\mathtt{05}], \mathit{markedAsSafe}(\mathrm{B})@[\mathtt{Apr}-\mathtt{21}, \mathtt{Apr}-\mathtt{21}]\}$$

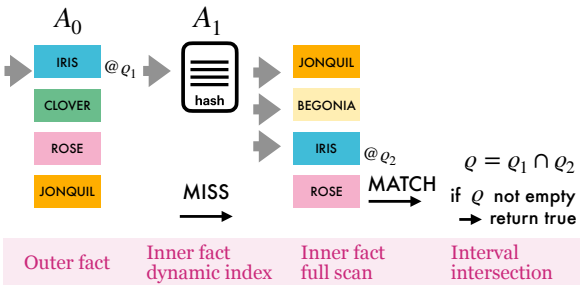


Figure 1 Illustration of an example of a temporal join algorithm execution; atoms with predicates A_0 and A_1 are joined on a *PlantNameTerm*.

Program:	**EMA**	
Dataset	Vadalog	InfluxDB
NQ1	0.0953	13.989
NQ10	4.8127	34.57
NQ50	74.1943	591.554
NQ100	306.6180	820.992

Figure 2 Experiment results, expressed in seconds, for the Exponential Moving Average in Vadalog and in the TSDB InfluxDB in the full paper [4]

By applying Rule 1, we see that only company B would be marked as *directSuspicious* as it was not *markedAsSafe* in the 7 days prior to the *suspiciousActivity*. However, to compute the conjunction between these two temporal atoms – $\boxminus_{[0,3]}$*suspiciousActivity* and $\diamondminus_{[0,7]}$*markedAsSafe* – one would need a *time-aware* join, able to join facts and their temporal intervals at the same time, while also handling stratified negation.

Temporal Join Algorithm

In Temporal Vadalog, the join algorithm is a temporal interval extension of the slot machine join of the Vadalog system [2], which is based on the index nested loop join [11]. In particular, during the first full scan of the inner relation, an in-memory index is constructed, unlike in the usual algorithms, where a precomputed index is typically present in materialized form.

The algorithm, fully reported in Algorithm 1, intuitively works as follows. Assume we have two predicates to be joined A_0 and A_1; first, we use the A_1 index to retrieve the next already-scanned fact that matches the join term(s) from A_0 (Line 8). If the index does not contain such a fact, we pursue the full scan until either a matching fact is found or all facts have been examined (Lines 10-14). If no further fact is found (Lines 15-22), we continue the scan with the next A_0 (if it exists and if A_1 is not negated). In case a fact is found, independently of whether it is from the index or the full scan, we produce the valid interval of the joined fact using the join logic (Line 23): difference for a negated literal, intersection for a positive interval, or a blend of interval operations and set operations for temporal operators like $\mathcal{S}$ (*since*) and $\mathcal{U}$ (*Until*). In the end, we check whether the resulting interval is valid, and if not, if A_1 is negated, we continue the scan with the next A_0 (if it exists) (Lines 25-29), otherwise we proceed with the loop; if the interval is non-empty and A_k is not negated, we return true as we have found a valid joined fact (Lines 31-32); otherwise, we continue retrieving the next "negated" fact. A visual representation of the execution of a temporal join is shown in Figure 1.

Time Series Operators

In Example 1, we assumed that the *suspiciousActivity* facts were provided by $\mathcal{D}$. Now we want to consider the predicate as a result of a different reasoning task.

▶ **Example 2.** Understanding whether a spending activity is suspicious implies detecting anomalous patterns, a task extremely relevant for financial authorities. A form of behavioural analysis is adopted here: if the number of flagged transactions is greater than the company average for a given period, then the spending activity is suspicious.

$$flagTransactions(C, AMT), sma(C, AVG), AMT > AVG \rightarrow suspiciousActivity(C) \quad (4)$$

Algorithm 1 Temporal Join between two predicates.

Input: predicates A_0 and A_1 to be joined, with A_0 not negated

```
1:  I_0 ← A_0.iterator()
2:  I_1 ← A_1.iterator()
3:  (a_0, ϱ_0) ← I_0.getNext()
4:  function NEXT
5:      interval ← ϱ_0
6:      while true do
7:          X ← a_0.joinTerm
8:          (a_1, ϱ_1) ← A_1.index.get(X)
9:          if a_1 is null then                          ▷ Continue full scan, if index miss
10:             while I_1.next() do
11:                 (a_1, ϱ_1) ← I_1.getNext()
12:                 A_1.index.put(a_1)                   ▷ Update the index map for A_1
13:                 if a_1.joinTerm == a_0.joinTerm then
14:                     break                            ▷ Exit the inner loop if matching fact is found
15:                 end if
16:             end while
17:         end if
18:         if a_1 is null then                          ▷ No further matching fact A_1 found
19:             if A_1 is negated then
20:                 return true
21:             end if
22:             if I_0.next() is false then
23:                 return false
24:             end if
25:             (a_0, ϱ_0) ← I_0.getNext()               ▷ Repeat loop for next a_0
26:             interval ← ϱ_0
27:             continue
28:         end if
29:         interval ← resultingIntervalFromJoinLogic(ϱ_1, interval)
30:         if interval is empty then
31:             if A_1 is negated then
32:                 if I_0.next() is false then
33:                     return false
34:                 end if
35:                 (a_0, ϱ_0) ← I_0.getNext()           ▷ Repeat loop for next a_0
36:                 interval ← ϱ_0
37:             end if
38:             continue
39:         end if
40:         if A_1 is not negated then
41:             return true
42:         end if
43:     end while
44: end function
```

The average value is given by the *sma* predicate (*simple moving average*), which encapsulates a time series operator that performs a moving average calculation to smooth the signal and filter out noise and transient variations. While time series analysis typically adopts ad-hoc software libraries, Temporal Vadalog intrinsically offers such statistics:

$$\diamondsuit_{[0,n)} timeSeries(X, Value) \rightarrow extended(X, Value) \tag{5}$$

$$timeSeries(X, Value), extended(X, Roll) \rightarrow rolling(X, Roll) \tag{6}$$

$$rolling(X, Roll), Avg = avg(Roll) \rightarrow sma(X, Avg) \tag{7}$$

We use $\diamondsuit$ to extend the validity of the window over n days (Rule 5), and we join it with the original time series to pin it to the correct starting date (Rule 6). As in SMA all data points have equal weight, Rule 7 computes the mathematical average over every window. Performance has been evaluated against a time series database (TSDB) in the full paper [4], in this case with the *exponential moving average* (EMA), on the NASDAQ Composite Index time series [14]. Results are shown in Figure 2.

3 The Temporal Vadalog Architecture

The temporal join is only one component of the system, and several others are required in order to support query answering over a set of rules like that of Examples 1-2, among which the transformations from the temporal operators and termination of recursive rules. Looking at the larger picture, the Temporal Vadalog architecture extends the *volcano iterator model* [13] of the Vadalog system [6] with *time-awareness*. A DatalogMTL program Π is transformed into an execution pipeline that reads data from sources, applies the transformations (both algebraic and time-related ones) and returns the intended output as a result. The process consists of two stages: in the first, the pipeline is built through a sequence of compilers and optimizers that gradually transforms the set of rules into a *reasoning query plan*. Taking inspiration from the *pipe and filters* architecture [9], each required transformation is represented by a filter, while dependencies between rules are represented by pipes. The second stage is at runtime, where a *pull-based* approach is used. Starting from the output filter, `next()` calls propagate through filter chains to source filters. Each filter applies the requested transformation based on the rule it represents. As long as data are available in the filter cascade, `next()` succeeds.

A number of time-relevant operations are tackled along the way: (a) the transformation of time intervals through the application of temporal operators; (b) the implementation of *merging operations* for intervals through various strategies to ensure correctness and efficiency [3]; (c) the temporal joins in the presence of *stratified negation*; (d) detection of repeating temporal patterns through the so-called *termination strategies*, i.e. techniques to guaranteee termination; (e) *temporal aggregations* [5]; and (f) the possibility to switch between temporal and to non-temporal reasoning, to activate non-temporal features, essential in some reasoning settings, such as *existential quantification*.

Summary. In this work, we showed the Temporal Vadalog system by first describing temporal joins and operators applied to an example, and concluded by briefly discussing the "big picture" of its architecture. The interested reader is referred to the full paper.

References

1 Teodoro Baldazzi, Luigi Bellomarini, and Emanuel Sallinger. Reasoning over financial scenarios with the Vadalog system. In *EDBT*, pages 782–791. OpenProceedings.org, 2023. doi:10.48786/edbt.2023.66.

2 Luigi Bellomarini, Davide Benedetto, Georg Gottlob, and Emanuel Sallinger. Vadalog: A modern architecture for automated reasoning with large knowledge graphs. *IS*, 105:101528, 2022. doi:10.1016/j.is.2020.101528.

3 Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger. The Temporal Vadalog system. In *RuleML+RR*, volume 13752, pages 130–145. Springer, 2022. doi:10.1007/978-3-031-21541-4_9.

4 Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger. The Temporal Vadalog system: Temporal Datalog-based reasoning. *Theory and Practice of Logic Programming*, pages 1–29, 2025. doi:10.1017/S1471068425000018.

5 Luigi Bellomarini, Markus Nissl, and Emanuel Sallinger. Monotonic Aggregation for Temporal Datalog. In *Proceedings of the 15th International Rule Challenge*, volume 2956, 2021. URL: https://ceur-ws.org/Vol-2956/paper30.pdf.

6 Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vadalog System: Datalog-based reasoning for knowledge graphs. *PVLDB*, 11(9):975–987, 2018. doi:10.14778/3213880.3213888.

7 Sebastian Brandt, Elem Güzel Kalayci, Roman Kontchakov, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Ontology-based data access with a Horn fragment of metric temporal logic. In *AAAI*, pages 1070–76. AAAI Press, 2017. doi:10.1609/aaai.v31i1.10696.

8 Sebastian Brandt, Elem Güzel Kalayci, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Querying log data with metric temporal logic. *J. Artif. Intell. Res.*, 62:829–877, 2018. doi:10.1613/jair.1.11229.

9 Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-oriented software architecture, 4th Edition*. Wiley, 2007. URL: https://www.worldcat.org/oclc/314792015.

10 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Semant.*, 14:57–83, 2012. doi:10.1016/j.websem.2012.03.001.

11 Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

12 Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991. doi:10.1145/116825.116838.

13 Goetz Graefe and William J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *ICDE*, pages 209–218, 1993. doi:10.1109/ICDE.1993.344061.

14 NASDAQ OMX Group. NASDAQ composite index. http://tinyurl.com/frednq, 2023. FRED, Federal Reserve Bank of St. Louis, Accessed: 2023-03-22.

15 David J. Tena Cucala, Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, and Egor V. Kostylev. Stratified negation in Datalog with metric temporal operators. In *AAAI*, pages 6488–6495, 2021. doi:10.1609/aaai.v35i7.16804.

16 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL: Computational complexity and expressive power. In *IJCAI*, pages 1886–1892, 2019. doi:10.24963/ijcai.2019/261.

17 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL over the integer timeline. In *KR*, pages 768–77, 2020. doi:10.24963/kr.2020/79.

18 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. Tractable fragments of Datalog with metric temporal operators. In *IJCAI*, pages 1919–1925, 2020. doi:10.24963/ijcai.2020/266.

19 Dingmin Wang, Pan Hu, Przemyslaw Andrzej Walega, and Bernardo Cuenca Grau. MeTeoR: Practical reasoning in Datalog with metric temporal operators. In *AAAI*, pages 5906–5913, 2022. doi:10.1609/aaai.v36i5.20535.

The appendix includes excerpts from the full version of the paper [4] to cover the more technical parts of this extended abstract. In particular, Appendix A provides the background of DatalogMTL.

A DatalogMTL

DatalogMTL is Datalog extended with operators from the metric temporal logic. We provide a summary of DatalogMTL with stratified negation under continuous semantics. DatalogMTL is defined over the rational timeline, i.e., an ordered set of rational numbers $\mathbb{Q}$. An interval $\varrho = \langle \varrho^-, \varrho^+ \rangle$ is a non-empty subset of $\mathbb{Q}$ such that for each $t \in \mathbb{Q}$ where $\varrho^- < t < \varrho^+$, $t \in \varrho$, and the endpoints $\varrho^-, \varrho^+ \in \mathbb{Q} \cup \{-\infty, \infty\}$. The brackets denote whether the interval is closed ("[]"), half-open ("[)","(]") or open ("()"), whereas angle brackets ("$\langle\rangle$") are used when unspecified. An interval is *punctual* if it is of the form $[t, t]$, *positive* if $\varrho^- \geq 0$, and *bounded* if $\varrho^-, \varrho^+ \in \mathbb{Q}$.

DatalogMTL extends the syntax of Datalog with negation with temporal operators [15]. For the following definitions, we consider a function-free first-order signature. An *atom* is of the form $P(\boldsymbol{\tau})$, where P is a n-ary predicate and $\boldsymbol{\tau}$ is a n-ary tuple of terms, where a term is either a constant or a variable. An atom is *ground* if it contains no variables. A *fact* is an expression $P(\boldsymbol{\tau})@\varrho$, where ϱ is an interval and $P(\boldsymbol{\tau})$ a ground atom and a *database* is a set of facts. A *literal* is an expression given by the following grammar, where ϱ is a positive interval: $A ::= \top \mid \bot \mid P(\boldsymbol{\tau}) \mid \boxminus_\varrho A \mid \boxplus_\varrho A \mid \diamondminus_\varrho A \mid \diamondplus_\varrho A \mid A \,\mathcal{S}_\varrho\, A \mid A \,\mathcal{U}_\varrho\, A$. A *rule* is an expression given by the following grammar, where $i, j \geq 0$, each A_k $(k \geq 0)$ is a literal and B is an atom: $A_1 \wedge \cdots \wedge A_i \wedge \text{not } A_{i+1} \wedge \cdots \wedge \text{not } A_{i+j} \rightarrow B$. The conjunction of literals A_k is the rule body, where $A_1 \wedge \cdots \wedge A_i$ denote positive literals and $A_{i+1} \wedge \cdots \wedge A_{i+j}$ denote negated (i.e., prefixed with *not*) literals. The atom B is the rule head. A rule is *safe* if each variable occurs in at least one positive body literal, *positive* if it has no negated body literals (i.e., $j = 0$), and *ground* if it contains no variables. A program Π is a set of safe rules and is *stratifiable* if there exists a stratification of a program Π. A stratification of Π is defined as a function σ that maps each predicate P in Π to a positive integer (*stratum*) s.t. for each rule, where P^h denotes a predicate of the head, and P^+ (resp. P^-) a positive (negative) body predicate, $\sigma(P^h) \geq \sigma(P^+)$ and $\sigma(P^h) > \sigma(P^-)$. The semantics of DatalogMTL is given by an interpretation $\mathfrak{M}$ that specifies for each time point $t \in \mathbb{Q}$ and each ground atom $P(\boldsymbol{\tau})$, whether $P(\boldsymbol{\tau})$ is satisfied at t, in which case we write $\mathfrak{M}, t \models P(\boldsymbol{\tau})$. This satisfiability notion extends to ground literals as follows:

$$
\begin{aligned}
&\mathfrak{M}, t \models \top && \text{for each } t \\
&\mathfrak{M}, t \models \bot && \text{for no } t \\
&\mathfrak{M}, t \models \boxminus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for all } s \text{ with } t - s \in \varrho \\
&\mathfrak{M}, t \models \boxplus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for all } s \text{ with } s - t \in \varrho \\
&\mathfrak{M}, t \models A \,\mathcal{S}_\varrho\, A' && \text{iff } \mathfrak{M}, s \models A' \text{ for some } s \text{ with } t - s \in \varrho \wedge\ \mathfrak{M}, r \models A \text{ for all } r \in (s, t) \\
&\mathfrak{M}, t \models A \,\mathcal{U}_\varrho\, A' && \text{iff } \mathfrak{M}, s \models A' \text{ for some } s \text{ with } s - t \in \varrho \wedge\ \mathfrak{M}, r \models A \text{ for all } r \in (t, s) \\
&\mathfrak{M}, t \models \diamondminus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for some } s \text{ with } t - s \in \varrho \\
&\mathfrak{M}, t \models \diamondplus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for some } s \text{ with } s - t \in \varrho
\end{aligned}
$$

An interpretation $\mathfrak{M}$ satisfies not A ($\mathfrak{M}, t \models \text{not } A$), if $\mathfrak{M}, t \not\models A$, a fact $P(\boldsymbol{\tau})@\varrho$, if $\mathfrak{M}, t \models P(\boldsymbol{\tau})$ for all $t \in \varrho$, and a set of facts $\mathcal{D}$ if it is a model of each fact in $\mathcal{D}$. Furthermore, $\mathfrak{M}$ satisfies a ground rule r if $\mathfrak{M}, t \models A_k$ for $0 \leq k \leq i$ and $\mathfrak{M}, t \models \text{not } A_k$ for $i+1 \leq k \leq i+j$ for every t; for every t, if the literals in the body are satisfied, so is the head $\mathfrak{M}, t \models B$; $\mathfrak{M}$

satisfies a rule when it satisfies every possible grounding of the rule. Moreover, $\mathfrak{M}$ is a *model* of a program if it satisfies every rule in the program and the program has a stratification, i.e., it is *stratifiable*. Given a stratifiable program Π and a set of facts $\mathcal{D}$, we call $\mathfrak{C}_{\Pi,\mathcal{D}}$ the *canonical model* of Π and $\mathcal{D}$ [8], and define it as the minimum model of Π and $\mathcal{D}$, i.e., $\mathfrak{C}_{\Pi,\mathcal{D}}$ is the minimum model for all the facts of $\mathcal{D}$ and the rules of Π. In this context, "minimum" means that the set of positive literals in $\mathfrak{M}$ is minimized or, equivalently, that the positive literals of this model are contained in every other model. Since Π is stratifiable, this minimum model exists and is unique [12]. According to Tena Cucala's notation [15], we say that a stratifiable program Π and a set of facts $\mathcal{D}$ entail a fact $P(\boldsymbol{\tau})@\varrho$, written as $(\Pi, \mathcal{D}) \models P(\boldsymbol{\tau})@\varrho$, if $\mathfrak{C}_{\Pi,\mathcal{D}} \models P(\boldsymbol{\tau})@\varrho$. In the remainder of the paper, we will assume the stratification of programs (or set of rules) as implicit.

In this context, the *query answering* or *reasoning* task is defined as follows: given the pair $Q = (\Pi, Ans)$, where Π is a set of rules, Ans is an n-ary predicate, and the query Q is evaluated over $\mathcal{D}$, then $Q(\mathcal{D})$ is defined as $Q(\mathcal{D}) = \{(\bar{t}, \varrho) \in dom(\mathcal{D})^n \times time(\mathcal{D}) \mid (\Pi, \mathcal{D}) \models Ans(\bar{t})@\varrho\}$, where $\bar{t}$ is a tuple of terms, the domain of $\mathcal{D}$, denoted $dom(\mathcal{D})$, is the set of all constants that appear in the facts of $\mathcal{D}$, and the set of all the time intervals in $\mathcal{D}$ is denoted as $time(\mathcal{D})$. As we shall see in practical cases, the Ans predicate of Π will be sometimes called "query predicate" and provided to the reasoning system with specific conventions, which we omit for space reasons, but will render in textual explanations.

Solutions to the Generalised Alibi Query in Moving Object Databases

Arthur Jansen[1]
Hasselt University, Databases and Theoretical Computer Science Group and
Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Bart Kuijpers
Hasselt University, Databases and Theoretical Computer Science Group and
Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Abstract

Space-time prisms provide a framework to model the uncertainty on the space-time points that a moving object may have visited between measured space-time locations, provided that a bound on the speed of the moving object is given. In this model, the *alibi query* asks whether two moving objects, given by their respective measured space-time locations and speed bound, may have met. An analytical solution to this problem was first given by Othman [5]. In this paper, we address the *generalised alibi query* that asks the same question for an arbitrary number $\mathsf{n} \geq 2$ of moving objects. We provide several solutions (mainly via the spatial and temporal projection) to this query with varying time complexities. These algorithmic solutions rely on techniques from convex and semi-algebraic geometry. We also address variants of the generalised alibi query where the question is asked for a given spatial location or a given moment in time.

2012 ACM Subject Classification Information systems → Spatial-temporal systems; Information systems → Query languages

Keywords and phrases Convex geometry, Semi-algebraic geometry, Space-time prism, Geographic information systems, Quantifier elimination

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.16

Category Short Paper

Related Version The paper is an extended abstract of [4].
Full Version: `https://doi.org/10.1016/j.comgeo.2024.102159`

Funding *Arthur Jansen*: Bijzonder Onderzoeksfonds (BOF22OWB06) from UHasselt

1 Introduction

In Moving Object Databases (MODs), various data models and query languages have been proposed to deal with moving objects whose position is recorded by location-aware devices (such as GPS), at not always regular moments in time [2]. The movement data of an object is therefore discrete in nature and can be seen as a sequence $\langle (x_1, y_1, t_1), \ldots, (x_n, y_n, t_n) \rangle$ of measured space-time locations, which we call a *trajectory sample*. Between measured space-time points, the trajectory of a moving object is unspecified and unknown and several models have been proposed to deal with this uncertainty. Based on the assumption that moving objects have some physically determined or law imposed speed bounds, the *space-time prism* model delimits the region in space-time which a moving object may have visited between two sampled points. This model, originating from the field of "time geography" in the 1970s, has

[1] Corresponding author

found its way into MOD research. The uncertainty on the movement of an object associated with a trajectory sample, is then modeled by a chain of space-time prisms. An illustration of two chains is shown in Figure 1.

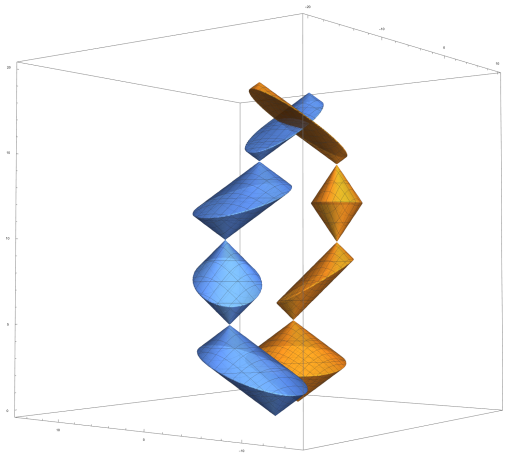


Figure 1 An illustration of two intersecting chains of space-time prisms (one chain in yellow and the other in blue).

One query of particular interest in this context is the *alibi query*, which asks whether two moving objects may have met, given their trajectory samples and speed bounds. The difficulty of answering this query can be reduced to deciding whether two space-time prisms intersect. Using a geometric argument, a solution to this problem was given by Othman et al. [5]. We address the *generalised alibi query*, which asks the same question, but for any (finite) number of moving objects. Because a space-time prism can be described by a system of polynomial inequalities, the generalised alibi query can be expressed as an existential first-order logic formula over the ordered field of real numbers. While there are algorithms for deciding the truth of such sentences [1], existing implementations cannot solve the query in practice (that is, within an acceptable amount of time). We provide several solutions (mainly via the spatial and temporal projection) to this query with varying time complexities. These algorithmic solutions rely on techniques from convex and semi-algebraic geometry, because space-time prisms are both convex and semi-algebraic sets. We also address variants of the generalised alibi query where the question is asked for a given spatial location or a given moment in time. Additionally, some of our methods are capable of producing a *sample point*, which is a point in the intersection of the n space-time prisms, if it exists. Finally, we give a quantifier-free description of the spatial projection of the intersection of n space-time prisms, which is exactly the region in space where the objects may have met (between two measured points), and allows answering the spatial variant of the generalised alibi query in linear time.

Our main contributions are summarised in Tables 1 and 2, which give an overview of the proposed methods or problems, their time complexity and their ability to produce sample points. For clarity, the complexity results in this table refer to deciding the emptiness of the intersection of n prisms. When the n moving objects are given by chains of prisms then the time complexity results in the table need to be multiplied by $L - \mathsf{n} + 2$, where L is the total number of prisms in the n chains.

We give a brief summary of the workings of each of the proposed methods in the next sections.

Table 1 Time complexity (in terms of the number of prisms n) and sample points of the proposed methods for the generalised alibi query.

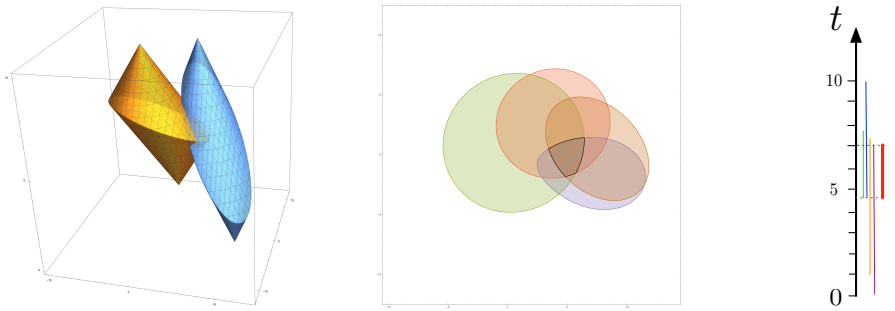
method	time complexity	sample points
via spatial projection	$O(\mathsf{n}^5)$	yes
via spatial projection with Helly	$O(\mathsf{n}^4)$	no
via temporal projection	$O(\mathsf{n}^3)$	yes

Table 2 Time complexity (in terms of the number of prisms n) and sample points of the variants for the generalised alibi query.

variants of generalised alibi query	time complexity	sample points
at a fixed location	$O(\mathsf{n})$	yes
at a fixed moment	$O(\mathsf{n}^3)$	yes

2 Deciding the generalised alibi query via the spatial projection

The essence of this method is to test whether the spatial projection of the intersection of the n space-time prisms is empty. Because this projection is precisely the region in space where the n objects could theoretically have met, we call it the *meeting region*. We show, using Fourier-Motzkin elimination, that the meeting region can be characterised as the intersection of regions enclosed by curves called "Cartesian ovals", which are generalisations of ellipses. An illustration of the meeting period as the intersection of such regions is shown in part (b) of Figure 2. This characterisation has two uses. On one hand, it provides a linear-time solution to the generalised alibi query at a fixed location, which includes the production of sample points. On the other hand, we can use it to test the emptiness of the meeting region, providing an answer to the generalised alibi query. To do this, we make use of the algebraic nature of Cartesian ovals, which allows us to compute a (finite) set of "candidate points", with the property that the meeting region is not empty if and only if it contains one of those candidate points. This method requires $O(\mathsf{n}^5)$ time and also produces sample points.



(a) (b) (c)

Figure 2 An illustration of (a) two intersecting prisms; (b) regions enclosed by Cartesian ovals whose intersection (in black) is the meeting region of the two prisms; and (c) the meeting period (in red) of the two prisms, shown as the intersection of four projections of intersections of three cones.

Due to Helly's theorem [3], which gives an equivalent condition for the non-emptiness of the intersection of convex sets, we can also solve the generalised alibi query by applying the above method to every combination of 4 of the n space-time prisms. The result of this is a method that only requires $O(\mathsf{n}^4)$ time. However, the disadvantage of that method is that it cannot produce sample points.

3 Deciding the generalised alibi query via the temporal projection

The method described here works by testing the emptiness of the temporal projection of the intersection of the space-time prisms. This temporal projection is called the *meeting period*, as it is precisely the period of time during which the n moving objects could have met. Because space-time prisms are closed and convex sets, the meeting period is a closed interval. In fact, not only can we test its emptiness, we can explicitly compute the meeting period, that is, compute its minimum and maximum. To do this, we give a new characterisation of the meeting period, based on Helly's theorem. This characterisation tells us that the computation of the meeting period can be reduced to the computation of the temporal projection of the intersection of three cones. An illustration of this characterisation of the meeting period is shown in part (c) of Figure 2. We have to compute such projection for every combination of three out of a set of $2\mathsf{n}$ cones. Because computing the temporal projection of three cones obviously takes constant time, the meeting period can be computed in $O(\mathsf{n}^3)$ time.

To compute the temporal projection of the intersection of three cones, we show that the minimum and maximum of such interval are contained in a finite set of "candidate moments". Then, we only have to test which of the candidate moments are contained in the temporal projection, which is straightforward.

Finally we use a general property about the intersection of convex sets to show that, given a moment in the meeting period, we can find a sample point in the intersection of the space-time prisms, also in $O(\mathsf{n}^3)$ time.

References

1 S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry.* Algorithms and Computation in Mathematics 10. Springer, Berlin, 2003.

2 R. Güting and M. Schneider. *Moving Object Databases.* Morgan Kaufmann, 2005.

3 E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jber. Deutsch. Math. Vereinig.*, 32:175–176, 1923.

4 A. Jansen and B. Kuijpers. Geometric and algorithmic solutions to the generalised alibi query. *Comput. Geom.*, 127:102159, 2025. doi:10.1016/J.COMGEO.2024.102159.

5 Bart Kuijpers, Rafael Grimson, and Walied Othman. An analytic solution to the alibi query in the space-time prisms model for moving object data. *International Journal of Geographical Information Science*, 25(2):293–322, February 2011. doi:10.1080/13658810902967397.

Visit Probability in Space-Time Prisms for Moving Object Data

Arthur Jansen[1]
Hasselt University, Databases and Theoretical Computer Science Group and
Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Bart Kuijpers
Hasselt University, Databases and Theoretical Computer Science Group and
Data Science Institute (DSI), Agoralaan, Building D, 3590 Diepenbeek, Belgium

Abstract

Space-time prisms have been extensively studied as a model to describe the uncertainty of the spatio-temporal location of a moving object in between measured space-time locations. In many applications, the desire has been expressed to provide an internal structure to these prisms, that includes what has been called "visit probability". Although several proposals have been studied in the past decades, a precise definition of this concept has been missing. The contribution of this paper is to provide such a specification by means of a formal framework for *visit probability*. Once this concept is established, we are able to derive on which parts of a prism, visit probability can be seen to give rise to a probability space.

2012 ACM Subject Classification Information systems → Spatial-temporal systems; Mathematics of computing → Probability and statistics

Keywords and phrases Spatio-temporal databases, moving object databases, space-time prisms, probability spaces

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.17

Category Short Paper

Funding *Arthur Jansen*: Bijzonder Onderzoeksfonds (BOF22OWB06) from UHasselt

1 Introduction

In a wide range of applications that deal with moving objects (such as people, animals or vehicles), time-stamped location data are collected using location-aware devices (such as GPS) and these data are stored and managed in moving object databases (MODs) [3]. The actual space-time trajectories of the moving object may be reconstructed or estimated from these measured space-time locations (called anchor points) using, for example, linear interpolation [11]. *Space-time prisms*, originating from the field of time geography [1, 4, 7], are used in Geographical Information Systems (GIS) [9] and MODs [2, 5] to model the movement uncertainty of a moving object between anchor points, based on a known speed bound on the object's movement. For spatio-temporal anchor points (p^-, t^-) and (p^+, t^+), with $t^- < t^+$, and a speed bound v_{max}, the prism with these and anchors and speed bound is denoted $\mathcal{P}(p^-, t^-, p^+, t^+, v_{max})$. Figure 1 depicts space-time prisms for movement in a two- and a one-dimensional space, respectively. As shown in the figure, prisms can be seen there to be the intersection of a future and past cone. The spatial projection of the prism, also called the *potential path area*, is an envelope of the spatial whereabouts of the moving object between the measured spatial locations.

[1] Corresponding author

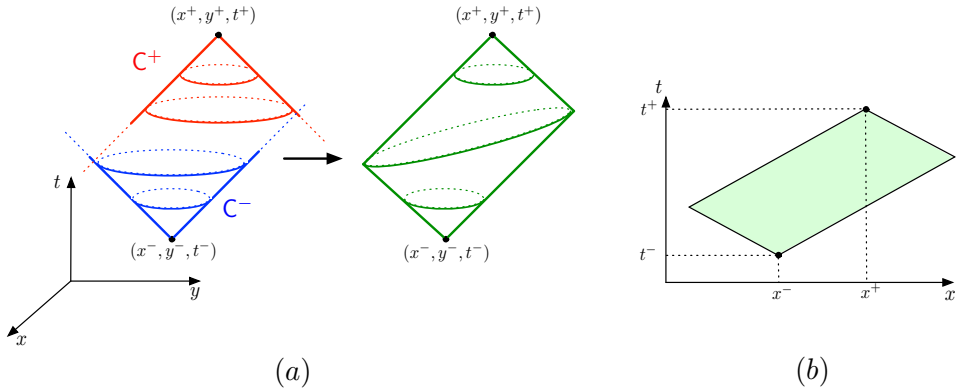


Figure 1 Part (a) of the figure shows the space-time prism (in green) for movement in the plane as the intersection of the past cone C^+ (in red) and the future cone C^- (in blue). Part (b) of the figure shows a prism for movement in a one-dimensional space.

In its basic form, a space-time prism lacks any internal structure and can be seen as a homogeneous geometric object, meaning that no two space-time points can be distinguished as more or less likely to have been visited. Conceptually, an infinite number of velocity-bound trajectories can be imagined within a prism and each point inside a prism is visited by infinitely many of them (except for some boundary points). This means that there is no a-priori reason to distinguish between space-time points inside the prism. Still, in many applications, such as animal or human movement [8], it is plausible that certain points in a prism, such as those on a linear interpolation path, should be considered more likely than other space-time points that are more towards the boundary of the prism (since they require a considerable detour). The notion of probability distributions in space-time prisms has become known as *visit probability* and has been studied extensively in the past decades (see e.g., [10, 12]). Several proposals have been made to assign probability values to space-time points or regions within a prism, thus providing the prism with an internal structure that expresses the unequal movement opportunities within the prism. Many of these proposals take an, often ad-hoc, approach towards the problem of establishing a visit probability for a particular application and each approach necessarily depends on a series of assumptions, which often remain obscure to a certain extend and the resulting visit probability therefore directly reflects these assumptions (in the best case, in a transparent way).

In this paper, we develop a general framework (or theory) of visit probability in the context of space-time prisms. In this approach, we start from the clear understanding that any definition of visit probability assumes a probability distribution on the possible velocity-bound trajectories within a prism. Once a probability space is defined on the set of trajectories (including a σ-algebra and a probability function), we can clearly determine for which parts of a space-time prims a visit probability can be derived. We also specify which conditions the σ-algebra on the set of trajectories must satisfy in order to be able to speak about a visit probability of certain classes of subsets of interest in the prism. Next, we address the question that asks on which subsets of a prism on which the derived visit probability is really a probability. We give a characterization of exactly those subsets of a prism for which this is the case. These sets are what we call "singleton-separators" and they are a wider class of subsets than just time slices of prisms. In the above mentioned literature, it is often the case that some notion of visit probability that has been defined is only considered on time slices of a prism. Our result shows that in fact it can be considered a probability on a wider class of subsets.

2 Towards a definition of visit probability in space-time prisms

In this section, we investigate how we can define the notion of "visit probability" on space-time prisms, thus providing some internal structure to an otherwise homogeneous prisms. Let $\mathcal{P} = \mathcal{P}(p^-, t^-, p^+, t^+, v_{max})$ be a space-time prism with anchors $(p^-, t^-), (p^+, t^+)$ and speed bound $v_{max} \geq 0$. A v_{max}-trajectory in the prism $\mathcal{P}$ is a (differentiable) mapping γ from the time interval $[t^-, t^+]$ to space, that connects the anchor points and whose velocity vector is bounded in size by v_{max} at all time. The set of v_{max}-trajectories in the prism $\mathcal{P}$ is denoted by $\Gamma_{\mathcal{P}}$. Space-time prisms are homogeneous in the sense that a-priori no higher likelihood can be assigned to a point (p, t) in a space-time prism $\mathcal{P} = \mathcal{P}(p^-, t^-, p^+, t^+, v_{max})$ as compared to another point (p', t') in $\mathcal{P}$. In general, the sets of v_{max}-trajectories passing through these points have the same cardinality (only points on the "rim" of a prism are exceptions [6]). Only by assigning a probability (or probability distribution) to the set of v_{max}-trajectories $\Gamma_{\mathcal{P}}$, we are be able to indicate certain points or parts of a prism as more likely than others. To obtain such a probability space on $\Gamma_{\mathcal{P}}$, we need to specify a σ-algebra $\mathcal{G}$ of subsets of $\Gamma_{\mathcal{P}}$ and to define a probability function P on this σ-algebra.

Once such a probability space $(\Gamma_{\mathcal{P}}, \mathcal{G}, P)$ has been specified, we can start working towards specifying a "visit probability": for a subset A of $\mathcal{P}$, for which $\Gamma_{\mathcal{P}}(A)$, which is the subset of v_{max}-trajectories that intersect A, belongs to $\mathcal{G}$, we define the *visit probability of A (relative to the given probability P on $\Gamma_{\mathcal{P}}$)* as $P(\Gamma_{\mathcal{P}}(A))$ and we denote it by $\mathsf{vp}_P(A)$.

However, the fact that $\Gamma_{\mathcal{P}}(A)$ belongs to the σ-algebra $\mathcal{G}$ is not guaranteed. Indeed, a σ-algebra $\mathcal{G}$ on $\mathcal{P}$ can range from very "poor" (or $\mathcal{G} = \{\emptyset, \Gamma_{\mathcal{P}}\}$) to very "rich" (or $\mathcal{G} = 2^{\mathcal{P}}$). In the former case, we can derive the visit probability of very few subsets of the prism, whereas in the latter case, we can derive it for all subsets. When we are interested in knowing the visit probability of a certain class $\mathcal{F}$ of subsets of the prism $\mathcal{P}$, we can only obtain this visit probability when for $F \in \mathcal{F}$, we have that $\Gamma_{\mathcal{P}}(F)$ belongs to the σ-algebra $\mathcal{G}$.

We turn to a visit probability for parts of the potential path area (PPA) of a prism. The following definition of visit probability of a part A of the the PPA reflects the idea that a moving object has visited A if at some point in time it has visited A. For a subset A of the PPA, we denote the cylindrical subset $(A \times \mathbb{R}) \cap \mathcal{P}$ of $\mathcal{P}$ by $\mathsf{Cyl}_{\mathcal{P}}(A)$ and when $\Gamma_{\mathcal{P}}(\mathsf{Cyl}_{\mathcal{P}}(A)) \in \mathcal{G}$, we can define the probability of visiting the part A of the the PPA as $\mathsf{vp}_P(\mathsf{Cyl}_{\mathcal{P}}(A))$.

3 When is visit probability a probability?

In this section, we determine in which circumstances the definition of visit probability on a prism, as given above, gives rise to a probability space within the prism. When we have a prism $\mathcal{P} = \mathcal{P}(p^-, t^-, p^+, t^+, v_{max})$ and consider the time slices $\mathcal{P}_{t_1}$ and $\mathcal{P}_{t_2}$ at two different moments $t^- \leq t_1 < t_2 \leq t^+$, then we clearly have $\mathsf{vp}_P(\mathcal{P}_{t_1}) = 1$ and $\mathsf{vp}_P(\mathcal{P}_{t_2}) = 1$, since every v_{max}-trajectory must intersect a time slice (that is $\Gamma_{\mathcal{P}}(\mathcal{P}_{t_1}) = \Gamma_{\mathcal{P}}(\mathcal{P}_{t_2}) = \Gamma_{\mathcal{P}}$). Since t_1 and t_2 are different, we have that the time slices $\mathcal{P}_{t_1}$ and $\mathcal{P}_{t_2}$ are disjoint subsets of $\mathcal{P}$. For disjoint unions, we would expect $\mathsf{vp}_P(\mathcal{P}_{t_1} \cup \mathcal{P}_{t_2})$ to be $\mathsf{vp}_P(\mathcal{P}_{t_1}) + \mathsf{vp}_P(\mathcal{P}_{t_2})$, but since this is $1 + 1 = 2$, that cannot be the case. This simple example shows that vp_P is not a probability on the complete prism $\mathcal{P}$. This raises the question: are there subsets of a prism on which the visit probability of Section 2 is a probability? The main contribution of this paper is a characterisation exactly those subsets of a prism for which this is the case. These subsets are "singleton-separators" in a prims, which are illustrated in Figure 2 for one-dimensional movement. A subset S of $\mathcal{P}$ is called a *singleton-separator*, if for any v_{max}-trajectory $\hat{\gamma} \in \Gamma_{\mathcal{P}}$, we have that the cardinality of $\hat{\gamma} \cap S$ equals one.

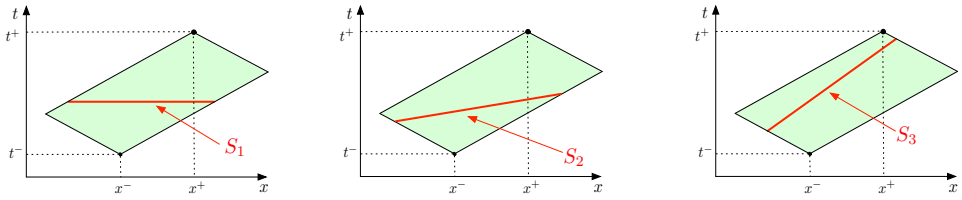


Figure 2 The subsets S_1 and S_2 are examples of singleton-separators, S_3 is not.

For a subset S of $\mathcal{P}$, we define $M_{\mathcal{G}}(S) := \{A \subseteq S \mid \Gamma_{\mathcal{P}}(A) \in \mathcal{G}\}$ and we have as a first result that S is a singleton-separator if and only if for every σ-algebra $\mathcal{G}$ on $\Gamma_{\mathcal{P}}$ we have that $M_{\mathcal{G}}(S)$ is a σ-algebra on S.

This means that the only subsets of a prism for which we can hope to obtain a probability space are the singleton-separators. The following theorem states how this probability space looks like.

▶ **Theorem 1.** *Let $\mathcal{P} = \mathcal{P}(p^-, t^-, p^+, t^+, v_{max})$ be a space-time prism and let S be a singleton-separator in $\mathcal{P}$. Then a probability space $(\Gamma_{\mathcal{P}}, \mathcal{G}, P)$ on the set of all v_{max}-trajectories induces a probability space $(S, M_{\mathcal{G}}(S), \mathsf{vp}_P)$ on S, when $\mathsf{vp}_P(A)$ is defined as $P(\Gamma_{\mathcal{P}}(A))$ for $A \in M_{\mathcal{G}}(S)$.*

References

1 L. Burns. *Transportation, Temporal, and Spatial Components of Accessibility*. Lexington Books, Lexington, MA, 1979.

2 Max J. Egenhofer. Approximation of geospatial lifelines. In Elisa Bertino and Leila De Floriani, editors, *SpadaGIS, Workshop on Spatial Data and Geographic Information Systems*. University of Genova, 2003.

3 R. Güting and M. Schneider. *Moving Object Databases*. Morgan Kaufmann, 2005.

4 T. Hägerstrand. What about people in regional science? *Papers of the Regional Science Association*, 24:7–21, 1970.

5 Kathleen Hornsby and Max J. Egenhofer. Modeling moving objects over multiple granularities. *Ann. Math. Artif. Intell.*, 36(1-2):177–194, 2002. doi:10.1023/A:1015812206586.

6 Bart Kuijpers and Walied Othman. Trajectory databases: data models, uncertainty and complete query languages. *J. Comput. Syst. Sci.*, 76(7):538–560, 2010. doi:10.1016/j.jcss.2009.10.002.

7 B. Lenntorp. *Paths in Space-Time Environments: A Time-Geographic Study of the Movement Possibilities of Individuals*. Number 44 in Series B. Lund Studies in Geography, 1976.

8 Jed A. Long. Modeling movement probabilities within heterogeneous spatial fields. *Journal of Spatial Information Science*, 16:85–116, 2018. doi:10.5311/JOSIS.2018.16.372.

9 H.J. Miller. A measurement theory for time geography. *Geographical Analysis*, 2005. doi:10.1111/j.1538-4632.2005.00575.x.

10 Ying Song and Harvey J. Miller. Simulating visit probability distributions within planar space-time prisms. *International Journal of Geographical Information Science*, 28(1):104–125, January 2014. doi:10.1080/13658816.2013.830308.

11 G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004. doi:10.1145/1016028.1016030.

12 Stephan Winter and Zhang Cai Yin. Directed movements in probabilistic time geography. *International Journal of Geographical Information Science*, 2010.

Prompting LLMs for the Run-Time Event Calculus

Andreas Kouvaras
University of Piraeus, Greece

Periklis Mantenoglou
Örebro University, Sweden

Alexander Artikis
University of Piraeus, Greece
NCSR "Demokritos", Athens, Greece

Abstract

Composite activity recognition systems analyse streams of low-level, symbolic events to identify instances of complex activities based on their formal definitions. Crafting these definitions is a challenging task, as it often requires specifying intricate spatio-temporal constraints, and acquiring labeled data for automated learning is difficult. To address this challenge, we introduce a method that leverages pre-trained Large Language Models (LLMs) to generate composite activity definitions, in the language of the Run-Time Event Calculus, from natural language descriptions.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning

Keywords and phrases Event Calculus, temporal pattern matching, composite event recognition

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.18

Category Short Paper

Supplementary Material *Software*: `https://github.com/aartikis/rtec`

Acknowledgements This work was supported partly by the EU-funded project ENEXA (101070305), and partly by the Wallenberg AI, Autonomous Systems and Software Program funded by the Knut and Alice Wallenberg Foundation.

1 Introduction

Composite event recognition (CER) systems process streams of symbolic, time-stamped events, in order to detect instances of composite activities of interest via temporal pattern matching over these input event streams [17, 7, 8, 1]. In Maritime Situational Awareness (MSA) e.g. the task is to monitor vessel position streams for detecting composite activities such as loitering and violations of the regulations concerning sailing in protected areas [3]. Composite activities are defined in a rigorous manner, by means of a formal pattern specification language. Consider, e.g., CER for city transport management (CTM), where we need to detect composite activities relating to the quality of public transportation based on symbolic event streams that stem from sensor data and contextual information, like an abrupt acceleration or a significant increase in passenger density within a bus. Towards safe public transportation, we would like to detect unsafe driving behaviours. To do this, we may employ a pattern stating that a bus is driven unsafely if it is making sharp turns, or it is accelerating or decelerating abruptly. A sharp turn may be defined as another composite activity, composed of consecutive "change in direction" events, while deriving abrupt acceleration and deceleration events may require background knowledge concerning the type of bus being used, complicating the formal definition of unsafe driving. As a result, constructing a pattern for a composite activity may become quite involved.

Hand-crafting composite activity definitions requires knowledge of formal languages – domain experts, such as transport engineers, cannot be expected to have such knowledge. Moreover, automatically constructing composite activity patterns via specialised machine learning techniques often requires large training datasets with labels for composite activity instances [6, 9, 15]. Unfortunately, the use of such techniques is commonly prohibited due to the scarcity of labels for the inherently infrequent composite activities.

To address these issues, we propose a method that constructs composite activity patterns from natural language descriptions using pre-trained Large Language Models (LLMs). We prompt LLMs to transform composite activity definitions expressed in natural language into logic programming definitions in the language of the "Run-Time Event Calculus" (RTEC) [14, 11, 12]. RTEC is an implementation of the Event Calculus, i.e., a formalism for representing events and reasoning about their effects over time [10, 5, 4]. RTEC is optimised by means of windowing and caching algorithms, demonstrating scalability in challenging domains, such as MSA and CTM, outperforming state-of-the-art CER systems [14, 13]. A key feature of our prompting method is that it is not custom to a single domain, but may be re-used, in a zero-shot manner, for the generation of composite activity definitions in any CER domain.

2 Background: RTEC

RTEC is a formal, logic programming framework that extends the Event Calculus [10] with optimisation techniques for CER [2, 14, 12]. The language of RTEC includes sorts for representing time, events and fluents, i.e., properties whose values may change over time. RTEC employs a linear time-line with non-negative integer time-points. A fluent-value pair (FVP) $F{=}V$ denotes that fluent F has value V. $\mathsf{happensAt}(E, T)$ signifies that event E occurs at time-point T. $\mathsf{initiatedAt}(F = V, T)$ (resp. $\mathsf{terminatedAt}(F = V, T)$) expresses that a time period during which a fluent F has the value V continuously is initiated (terminated) at T. $\mathsf{holdsAt}(F = V, T)$ states that F has value V at T, while $\mathsf{holdsFor}(F = V, I)$ expresses that $F{=}V$ holds continuously in the maximal intervals of list I.

A formalisation of composite activity definitions in RTEC is called *event description*. An event description may contain rules defining two types of FVPs: "simple" and "statically determined". A simple FVP is defined using a set of initiatedAt and terminatedAt rules, and is subject to the commonsense law of inertia, i.e., an FVP $F{=}V$ holds at a time-point T, if $F{=}V$ has been "initiated" by an event at a time-point earlier than T, and not "terminated" by another event in the meantime.

▶ **Example 1** (Within area)**.** In the maritime domain, vessel activity may be disallowed in certain areas, e.g., fisheries restricted areas. Thus, it is desirable to compute the maximal intervals during which a vessel is in such an area. See the definition of a simple FVP below:

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{withinArea}(\mathit{Vessel}, \mathit{AreaType}) = \mathsf{true}, T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{entersArea}(\mathit{Vessel}, \mathit{AreaID}), T), \quad \mathit{areaType}(\mathit{AreaID}, \mathit{AreaType}).\end{aligned} \tag{1}$$

$$\begin{aligned}&\mathsf{terminatedAt}(\mathit{withinArea}(\mathit{Vessel}, \mathit{AreaType}) = \mathsf{true}, T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{leavesArea}(\mathit{Vessel}, \mathit{AreaID}), T), \quad \mathit{areaType}(\mathit{AreaID}, \mathit{AreaType}).\end{aligned} \tag{2}$$

$$\begin{aligned}&\mathsf{terminatedAt}(\mathit{withinArea}(\mathit{Vessel}, \mathit{AreaType}) = \mathsf{true}, T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{gapStart}(\mathit{Vessel}), T).\end{aligned} \tag{3}$$

$\mathit{withinArea}(\mathit{Vessel}, \mathit{AreaType})$ is a Boolean fluent denoting that a Vessel is in an area of type $\mathit{AreaType}$, while $\mathit{entersArea}(\mathit{Vessel}, \mathit{AreaID})$, $\mathit{leavesArea}(\mathit{Vessel}, \mathit{AreaID})$ and $\mathit{gapStart}(\mathit{Vessel})$ are input events, derived by the online processing of vessel position signals,

and their spatial relations with areas of interest [16]. $areaType(AreaID, AreaType)$ is an atemporal predicate storing background knowledge regarding the types of areas in a dataset. Rules (1) and (2) state that $withinArea(Vessel, AreaType)$ is initiated (resp. terminated) as soon as a *Vessel* enters (leaves) an area *AreaID*, whose type is *AreaType*. Rule (3) expresses that $withinArea(Vesel, AreaType)$ is terminated when there is a communication gap, i.e., when the *Vessel* stops transmitting its position, and we become uncertain of its whereabouts. ⌟

A statically determined FVP $F = V$ is defined via a rule with head $\mathsf{holdsFor}(F = V, I)$. This rule computes the maximal intervals during which $F = V$ holds continuously by applying a set interval manipulation operations, i.e., union_all, intersect_all and relative_complement_all, on the maximal intervals of other FVPs.

▶ **Example 2** (Anchored and moored vessels)**.** Consider the following definition of a statically determined FVP:

$$\begin{aligned}&\mathsf{holdsFor}(anchoredOrMoored(Vessel) = \mathsf{true}, I) \leftarrow \\ &\quad \mathsf{holdsFor}(stopped(Vessel) = farFromPorts, I_{sf}), \\ &\quad \mathsf{holdsFor}(withinArea(Vessel, anchorage) = \mathsf{true}, I_a), \quad \mathsf{intersect\_all}([I_{sf}, I_a], I_{sfa}), \\ &\quad \mathsf{holdsFor}(stopped(Vl) = nearPorts, I_{sn}), \quad \mathsf{union\_all}([I_{sfa}, I_{sn}], I).\end{aligned} \tag{4}$$

$anchoredOrMoored(Vessel)$ is a Boolean statically determined fluent, defined in terms of three other FVPs: $stopped(Vessel) = farFromPorts$, $stopped(Vessel) = nearPorts$ and $withinArea(Vessel, anchorage) = \mathsf{true}$. The multi-valued fluent $stopped(Vessel)$ expresses the periods during which a *Vessel* is idle near some port or far from all ports. Rule (4) derives the intervals during which a *Vessel* is both stopped far from all ports and within an anchorage area, by applying the intersect_all operation on the lists of maximal intervals I_{sf} and I_a. The output of this operation is list I_{sfa}. Subsequently, list I is derived by applying union_all on lists I_{sfa} and I_{sn}. This way, list I contains the maximal intervals during which a *Vessel* has stopped near some port or within an anchorage area. ⌟

A statically determined FVP holds as long as a Boolean combination of other FVPs is satisfied. Thus, statically determined FVPs are tailored for modeling composite activities that may be defined by applying conjunction, disjunction and negation operators on other activities – see "anchored or moored". "Inertial" composite activities, i.e., activities that persist through time and may arise (or conclude) based on the satisfaction of a set of instantaneous conditions, are expressed using initiatedAt and terminatedAt rules. Typically, a statically determined FVP representation leads to more efficient reasoning, but not all simple FVPs are translatable to statically determined ones. A formal analysis, including an account of the syntax, semantics and reasoning algorithms of RTEC may be found in [14, 12].

3 Prompt Pipeline

Hand-crafting composite activity definitions requires knowledge of the language of RTEC – maritime experts e.g. cannot be expected to have such knowledge. To address this issue, we present a prompting approach that leverages the power of LLMs for constructing composite activity definitions. Figure 1 presents the pipeline for translating natural language descriptions of composite activities into RTEC rules. First, we introduce to the LLM the core predicates of RTEC (**Prompt R**), and provide the syntax for the definitions of simple and statically determined FVPs (**Prompt S**). Then, we proceed with each application domain – for each such domain we present the items of the input stream, i.e. the events and input FVPs (**Prompt E** and **Prompt F**), and the background knowledge predicates (**Prompt B**) – recall e.g. *areaType* in rule-set (1)–(3). Subsequently, **Prompt G** asks the LLM to translate

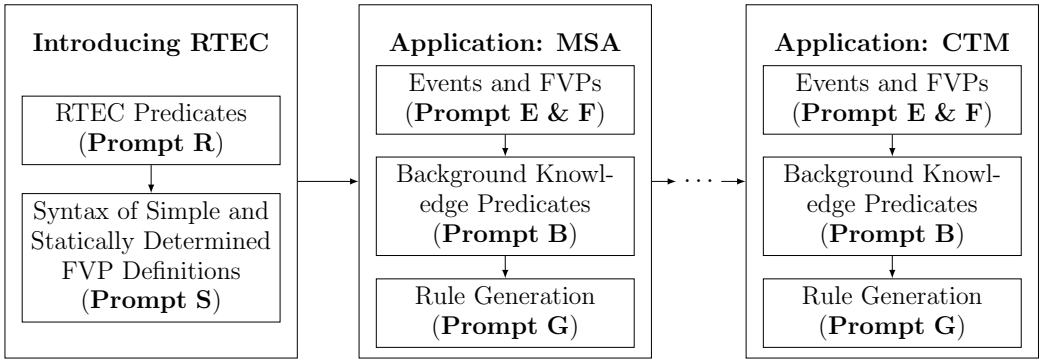


Figure 1 LLM prompting for composite activity definition generation.

a natural language description of each composite activity of the application domain under consideration into a set of RTEC rules. We may customise and repeat **Prompts E, F, B** and **G** for each subsequent application domain. **Prompts R** and **S** are not repeated.

Prompt R introduces the predicates of RTEC. Subsequently, we use **Prompt S** to demonstrate to the LLM how to express a composite activity as a simple or a statically determined FVP. We start with a description of the syntax of the rules expressing simple FVPs. Then, we employ chain-of-thought prompting, according to which we provide natural language descriptions of two example composite activities, and show how these activities may be expressed as simple FVPs. The prompt fragment below illustrates this process; a fragment illustrating statically determined FVPs, is presented in the Appendix. The chosen examples concern MSA. Lines 14, 17 and 20 of the prompt below should be replaced with resp. rules (1), (2), and (3).

Listing 1 Fragment of **Prompt S.**

```
1 There are two ways in which a composite activity may be defined in the language
  of RTEC. In the first case, a composite activity definition may be specified by
  means of rules with ‘initiatedAt(F=V, T)’ or ‘terminatedAt(F=V, T)’ in their head.
   This is called a simple fluent definition.
2
3 The first body literal of an ‘initiatedAt(F=V,T)’ rule is a positive ‘happensAt’
  predicate; this predicate is followed by a possibly empty set of positive or
  negative ‘happensAt’ and ‘holdsAt’ predicates. Negative predicates are prefixed
  with ‘not’ which expresses negation-by-failure. In some cases, the body of an  ‘
  initiatedAt(F=V,T)’ rule may include predicates expressing background knowledge.
4
5 ‘terminatedAt(F=V,T)’ rules are specified in a similar way.
6
7 Below you may find two examples of composite activity definitions, from the
  maritime domain, expressed as simple fluents.
8
9 Example 1: Given a composite maritime activity description, provide the rules in
  the language of RTEC.
10 Composite Maritime Activity Description: ‘withinArea’. This activity starts when
  a vessel enters an area of interest. The activity ends when the vessel leaves the
   area that it had entered, or when the vessel stops transmitting its position,
  since we can no longer assume that the vessel remains in the same area in the
  case of transmission gaps.
11
```

```
Answer:
The activity ‘withinArea’ is expressed as a Boolean simple fluent with two
arguments, i.e., ‘Vessel’ and ‘AreaType’. This activity starts when a vessel
enters an area of interest. We use an ‘initiatedAt’ rule to express this
initiation condition. The body literals of this rules are an event labelled ‘
entersArea’ with two arguments, ’Vessel’ and ‘Area’, and a background knowledge
predicate named ‘areaType’ with two arguments, ‘Area’ and ‘AreaType’. This rule
in the language of RTEC is the following:
<Rule (1)>

The activity ‘withinArea’ ends when a vessel leaves the area that it had entered.
 We use a ‘terminatedAt’ rule to describe this termination condition. This rule
includes an event named ‘leavesArea’ with two arguments, i.e. ‘Vessel’ and ’Area’,
 and the background knowledge predicate ‘areaType’. This rule in the language of
RTEC language is:
<Rule (2)>

In addition to the aforementioned conditions, the activity ‘withinArea’ ends when
 the vessel stops transmitting its position, i.e. when a communication gap starts.
 We use a ‘terminatedAt’ rule to express this termination condition. In this rule,
 the second argument of the ‘withinArea’ fluent is a ‘free’ Prolog variable, i.e.
 a variable starting with ‘_’. The body of this rule includes a single event
named ‘gap_start’ with one argument, i.e. ‘Vessel’. This rule in the language of
RTEC is:
<Rule (3)>

Example 2: <Description 2>
```

According to **Prompt S**, there are two ways in which a natural language description of a composite activity should be expressed. First, the textual description may indicate the conditions in which the composite activity is said to start taking place, as well as the conditions in which the activity is said to stop taking place. This may be achieved with the use of key phrases such as “the activity starts” and “the activity ends” (see **Prompt S** above). Second, we may identify the conditions that must be satisfied so that the composite activity in question holds at any given time. To achieve this, we may enter in the textual description of the activity the key phrase “as long as” (see **Prompt S – Statically determined FVPs** in the Appendix). The first type of textual description implies a simple FVP representation, while the second type of description implies a statically determined FVP formulation. In any case, the compiler of RTEC will choose the most efficient representation [12].

4 Further Work

We aim to evaluate our approach using leading LLMs in diverse domains, such as CER for Maritime Situational Awareness (MSA) and City Transport Management (CTM). Moreover, we aim to fine-tune manageable versions of LLMs for avoiding any errors that may occur in rule generation.

References

1 Elias Alevizos, Alexander Artikis, and Georgios Paliouras. Complex event recognition with symbolic register transducers. *Proc. VLDB Endow.*, 17(11):3165–3177, 2024. doi:10.14778/3681954.3681991.

2 Alexander Artikis, Marek J. Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.*, 27(4):895–908, 2015. doi:10.1109/TKDE.2014.2356476.

3 Alexander Artikis and Dimitris Zissis, editors. *Guide to Maritime Informatics*. Springer, 2021. doi:10.1007/978-3-030-61852-0.

4 Iliano Cervesato and Angelo Montanari. A calculus of macro-events: Progress report. In *TIME*, pages 47–58, 2000. doi:10.1109/TIME.2000.856584.

5 Luca Chittaro and Angelo Montanari. Efficient temporal reasoning in the cached event calculus. *Comput. Intell.*, 12:359–382, 1996. doi:10.1111/J.1467-8640.1996.TB00267.X.

6 Lars George, Bruno Cadonna, and Matthias Weidlich. Il-miner: Instance-level discovery of complex event patterns. *Proc. VLDB Endow.*, 10(1):25–36, 2016. doi:10.14778/3015270.3015273.

7 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020. doi:10.1007/S00778-019-00557-W.

8 Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. A formal framework for complex event recognition. *ACM Trans. Database Syst.*, 46(4):16:1–16:49, 2021. doi:10.1145/3485463.

9 Nikos Katzouris, Georgios Paliouras, and Alexander Artikis. Online learning probabilistic event calculus theories in answer set programming. *Theory Pract. Log. Program.*, 23(2):362–386, 2023. doi:10.1017/S1471068421000107.

10 R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Computing*, 4(1):67–96, 1986. doi:10.1007/BF03037383.

11 Periklis Mantenoglou and Alexander Artikis. Extending the range of temporal specifications of the run-time event calculus. In *TIME*, volume 318, pages 6:1–6:14, 2024. doi:10.4230/LIPICS.TIME.2024.6.

12 Periklis Mantenoglou and Alexander Artikis. Temporal specification optimisation for the event calculus. In *AAAI-25*, pages 15075–15082, 2025. doi:10.1609/AAAI.V39I14.33653.

13 Periklis Mantenoglou, Dimitrios Kelesis, and Alexander Artikis. Complex event recognition with allen relations. In *KR*, pages 502–511, 2023. doi:10.24963/KR.2023/49.

14 Periklis Mantenoglou, Manolis Pitsikalis, and Alexander Artikis. Stream reasoning with cycles. In *KR*, pages 544–553, 2022.

15 Evangelos Michelioudakis, Alexander Artikis, and Georgios Paliouras. Online semi-supervised learning of composite event rules by combining structure and mass-based predicate similarity. *Mach. Learn.*, 113(3):1445–1481, 2024. doi:10.1007/S10994-023-06447-1.

16 Georgios M. Santipantakis, Akrivi Vlachou, Christos Doulkeridis, Alexander Artikis, Ioannis Kontopoulos, and George A. Vouros. A stream reasoning system for maritime monitoring. In *TIME*, volume 120, pages 20:1–20:17, 2018. doi:10.4230/LIPICS.TIME.2018.20.

17 Walker M. White, Mirek Riedewald, Johannes Gehrke, and Alan J. Demers. What is "next" in event processing? In *PODS*, pages 263–272, 2007. doi:10.1145/1265530.1265567.

A Prompt S

Below we present a fragment of **Prompt S** that introduces the syntax and an example of statically determined FVPs.

Listing 2 Prompt S – Statically determined FVPs.

```
The second way in which a composite activity may be defined in the language of
RTEC concerns statically determined fluents. In this case, a composite activity
definition may be specified by means of a rule with 'holdsFor(F=V, I)' in its
head. The body of such a rule may include 'holdsFor' conditions for fluents other
 than F, as well as some of the interval manipulation constructs of RTEC, i.e. '
union_all', 'intersect_all', and 'relative_complement_all'. In some cases, a '
holdsFor(F=V, I)' rule may include predicates expressing background knowledge. A
rule with 'holdsFor(F=V, I)' in the head is called a statically determined fluent
 definition. Below you may find two examples of composite maritime activities
expressed as statically determined fluents.
```

```
2
3 Example 1: Given a composite maritime activity description, provide the rules in
  the language of RTEC.
4 Composite Maritime Activity Description: ‘underWay’. This activity lasts as long
  as a vessel is not stopped.
5
6 Answer: The activity ‘underWay’ is expressed as a statically determined fluent.
  Rules with ‘holdsFor’ in the head specify the conditions in which a fluent holds.
   We use a ‘holdsFor’ rule to describe that the ‘underWay’ activity lasts as long
  as a vessel is not stopped. The output is Boolean fluent named ‘underWay’ with
  one argument, i.e. ‘Vessel’. We specify ‘underWay’ with the use of the fluent ‘
  movingSpeed’. We express ‘underWay’ as the disjunction of the three values of ‘
  movingSpeed’, i.e. ‘below’, ‘normal’ and ‘above’. Disjunction in ‘holdsFor’ rules
   is expressed by means of ‘union_all’. This rule is expressed in the language of
  RTEC as follows:
7
8 holdsFor(underWay(Vessel)=true, I) :-
9         holdsFor(movingSpeed(Vessel)=below, I1),
10        holdsFor(movingSpeed(Vessel)=normal, I2),
11        holdsFor(movingSpeed(Vessel)=above, I3),
12        union_all([I1,I2,I3], I).
13
14 Example 2: <Description 2>
```

Temporal Association Rules from Motifs

Mauro Milella
Department of Mathematics and Computer Science, University of Ferrara, Italy

Giovanni Pagliarini
Department of Mathematics and Computer Science, University of Ferrara, Italy

Guido Sciavicco
Department of Mathematics and Computer Science, University of Ferrara, Italy

Ionel Eduard Stan
Department of Informatics, Systems, and Communications, University of Milano-Bicocca, Italy

Abstract

A motif is defined as a frequently occurring pattern within a (multivariate) time series. In recent years, various techniques have been developed to mine time series data. However, only a few studies have explored the idea of using motif discovery in temporal association rule mining. Interval-based temporal association rules have been recently defined and studied, along with the temporal version of the known frequent patterns, and therefore, association rule extraction algorithms (such as APRIORI and FP-Growth). In this work, we define a vocabulary of propositional letters wrapping motifs, and show how to extract temporal association rules starting from such a vocabulary. We apply our methodology to time series datasets in the fields of hand signs execution and gait recognition, and we discuss how they capture curious insights within data, keeping a high level of interpretability.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Theory and algorithms for application domains

Keywords and phrases Motifs, Interval Temporal Logic, Association Rules

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.19

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/aclai-lab/Sole.jl` [11]
archived at `swh:1:dir:dd723aee72578208606649ff12168e891cdae221`
Software (Source Code): `https://github.com/aclai-lab/ModalAssociationRules.jl` [10]
archived at `swh:1:dir:697da0b30a22cd23450ab445a887ebf1a602db8f`

Funding We acknowledge the support of the FIRD project Methodological Developments in Modal Symbolic Geometric Learning, funded by the University of Ferrara. Moreover, this research has also been funded by the Italian Ministry of University and Research through PNRR – M4C2 – Investimento 1.3 (Decreto Direttoriale MUR n. 341 del 15/03/2022), Partenariato Esteso PE00000013 – "FAIR – Future Artificial Intelligence Research" – Spoke 8 "Pervasive AI", funded by the European Union under the NextGeneration EU programme.

1 Introduction

In machine learning, we distinguish between *functional* and *symbolic* learning. The former encompasses strategies for representing the theory underlying a certain phenomenon as *functions*, while the latter derives *logical descriptions* of that phenomenon. Traditional symbolic methods rely on propositional logic and assume static data, in which each instance is described by n attributes. Temporal data, however, cannot be successfully dealt within the same schema in a native way, and it is commonly pre-processed (e.g., via averaging attributes along all dimensions) to appear static, enabling standard symbolic techniques.

Modal, and in particular temporal symbolic learning [8, 12] takes a different point of view: modal symbolic methods are characterized by being based on modal logic (e.g., temporal logic) so that non-static data can be dealt with natively, and that the extracted knowledge takes the form of interpretable modal logic formulas. Association rule extraction techniques can benefit from a similar approach [9, 14], with the introduction of the modal adaptation of the known frequent set extraction algorithms, namely APRIORI [1] and FP-Growth [5]. In particular, the temporal case is considered a representative example to illustrate the qualities and characteristics of modal association rules. Nevertheless, the original approach presents some important limitations, originating from the basic definition of temporal alphabet, which may cause difficult-to-interpret association rules. One way to overcome the limits in alphabet definition towards temporal association rule extraction is to consider time series *motifs*, that is, patterns that are considered interesting because they frequently occur.

In this paper, we consider the problem of temporal rule extraction from time series using a motif-based alphabet. We adapt the original definition of *local support*, introduced in [9, 14], to accommodate motifs suitably, and apply our methodology to two temporal datasets, showing how the obtained rules have an immediate natural language translation.

2 Background

Motif discovery for time series was introduced in 2003 [4], generating quite a body of research. Virtually every time series data mining technique has been applied to the motif discovery problem, including indexing, data discretization, triangular-inequality pruning, hashing, early abandoning.

▶ **Definition 1.** *A* time series *is a sequence* $T : t_1, \ldots, t_N$ *of* N *real-valued observations. A set of* $\mathcal{T}$ *time series* $T_1, \ldots, T_n$ *is called* multivariate *time series. A region* $T_{ij} : t_i, \ldots, t_j$ *of* $j - i + 1$ *consecutive observations in a time series is called a* subsequence*. Given two subsequences* T_i *and* T_k*, their* distance *is the Euclidean distance between their Z-normalized forms.*

Given a subsequence, we can compute its distance to all subsequences in the same time series.

▶ **Definition 2.** *Given an* N*-observations time series* T *and a subsequence* T_{ij}*, the* distance profile D_i *is the vector of all distances between* T_{ij} *and* $T_{k(k+j-i+1)}$*, for every* $1 \leq k \leq N$*.*

Distance profiles are collected in a distance matrix, from which a matrix profile is built.

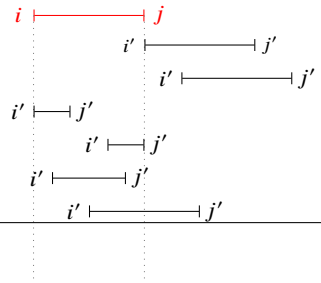
▶ **Definition 3.** *Given a time series* T *and a length* l*, the* full distance matrix $M_{T,l}$ *is the squared matrix of dimension* $N - l + 1$ *whose* i*-th row is the distance profile* D_i*. Given an integer* s*, the* matrix profile P *is the vector that at position* i *contains the minimum value in* D_i*, ignoring all the values in* $D_{i(i-s)}$ *and* $D_{i(i+s)}$*, as these distances are considered trivialities.*

For each position i in a matrix profile P, if the value P_i is lower than a threshold α, we say that $T_{i(i+l)}$ is an instance of a *motif*.

Note that both the literature and the maintained packages on motif discovery are relatively extensive; in this work, we enrich the existing software ecosystem with the framework *Sole.jl* [1], an open source solution for symbolic learning with non-tabular data, leveraging the *MatrixProfile.jl* [2] library, which implements the STAMP algorithm [15] for computing the matrix profile of a time series, allowing for motifs discovery.

[1] See `https://github.com/aclai-lab/Sole.jl` and `https://github.com/aclai-lab/ModalAssociationRules.jl`.
[2] `https://github.com/baggepinnen/MatrixProfile.jl`

Table 1 Allen's relation and their notation within HS; equality and inverse relations are omitted.

HS modality	Definition w.r.t. the interval structure			Example
$\langle A \rangle$ (after)	$[i,j]R_A[i',j']$	$\Leftrightarrow$	$j = i'$	
$\langle L \rangle$ (later)	$[i,j]R_L[i',j']$	$\Leftrightarrow$	$j < i'$	
$\langle B \rangle$ (begins)	$[i,j]R_B[i',j']$	$\Leftrightarrow$	$i = i' \wedge j' < j$	
$\langle E \rangle$ (ends)	$[i,j]R_E[i',j']$	$\Leftrightarrow$	$j = j' \wedge i < i'$	
$\langle D \rangle$ (during)	$[i,j]R_D[i',j']$	$\Leftrightarrow$	$i < i' \wedge j' < j$	
$\langle O \rangle$ (overlaps)	$[i,j]R_O[i',j']$	$\Leftrightarrow$	$i < i' < j < j'$	

In this paper, we are interested in learning association rules from data. Fixed an alphabet of propositional literals $\mathcal{P} = \{p_1, \ldots, p_k\}$, a *propositional rule* is an object of the type $\rho : X \Rightarrow Y$, where $X \subset \mathcal{P}$ is called *antecedent*, $Y \subset \mathcal{P}$ is called *consequent*, and $X \cap Y = \emptyset$; following the classical jargon, we refer to each literal as *item* and we call a set of items as *itemset*. In this scenario, $X \cup Y$ is considered interesting when it is *frequent*, that is, if it occurs more often than a predetermined threshold referred to as *minimum support*, and a rule $X \Rightarrow Y$ is extracted if the ratio between the support of X and that of $X \cup Y$ is higher than another predetermined threshold known as *minimum confidence*.

While classical association rules are designed for propositional patterns to emerge, temporal association rules are designed to generalize this idea to patterns with a temporal component. The natural choice to describe temporalized co-occurrence of events or patterns with a duration is *Halpern and Shoham modal logic of time intervals* (HS), defined over Allen's relations, as show in Tab. 1.

Interval temporal logic gives us a way to naturally describe temporal association rules, as time series can be naturally seen as interval models. Let $\mathfrak{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ be a set of (multivariate) time series, or *temporal dataset*, and fix a propositional alphabet $\mathcal{P}$; let us also assume that each time series in $\mathfrak{T}$ is based on the same temporal domain D. Each single multivariate time series $\mathcal{T}$ is a collection of n time series $T_1, \ldots, T_n$; elements of $\mathcal{P}$ are naturally associated to a specific time series. In this way, if $\mathcal{T}$ is a multivariate time series (i.e., an interval model), $[i,j]$ is an interval, and φ an interval formula, the notion of $\mathcal{T}, [i,j] \Vdash \varphi$ can be interpreted as the notion of $\mathcal{T}$ *satisfies* φ *at* $[i,j]$.

A *temporal itemset* is a set of temporal items, that is, items enriched with a temporal relation, and temporal rules are such that antecedents and consequents are temporal itemsets. The notion of support is generalized to the case of a temporal dataset $\mathfrak{T}$ by distinguishing a *local support*, computing the relative frequency of a temporal itemset occurring in some instance $\mathcal{T} \in \mathfrak{T}$, and a *global support*, counting the number of instances such that their local support is higher than a minimum local support threshold. An itemset X is said to be *frequent* if its global support is greater than a minimum global support threshold. In this scenario, ModalAPRIORI [14] and ModalFP-Growth [9] can be used to extract temporal patterns as a particular case of modal patterns.

3 Extracting Temporal Association Rule from Motifs

Let us consider a set of *feature extraction functions* $\mathcal{F} = \{F_1, \ldots, F_k\}$, where each function F is defined as $F : \mathbb{R}^d \to \mathbb{R}$ for some natural value $d \leq N$. Theoretically speaking, given a multivariate time series $\mathcal{T}$, it is possible to define an alphabet of items based on $\mathcal{F}$, that is, $\mathcal{P} = \{\alpha \leq F(T) \leq \beta \mid F \in \mathcal{F}, T \in \mathcal{T}, \alpha \in \mathbb{R} \cup \{-\infty\}, \beta \in \mathbb{R} \cup \{+\infty\}\}$, which allows for mining temporal association rules, as items can be immediately interpreted over intervals, obtaining a scalar value which can be compared with lower and upper bounds α, β.

Unfortunately, it can be shown that this approach may introduce strong bias during (local) support computation, leading to promising association rules which, however, encode trivialities. An intuition about this is that the scalar value obtained by applying a feature extraction function to an interval $[i, j]$, could be redundant with many other identical values obtained by applying the same function on sub-intervals or super intervals of $[i, j]$ (e.g., considering *max*, *min*, *average* functions, but even more refined functions such as *catch22* [7]).

To avoid flattening intervals with feature extraction functions, we modify the definition of the alphabet. We consider a set $\mathcal{A}_{\mathfrak{T}}$ of the *most representative motifs* for the temporal dataset $\mathfrak{T}$, that is, the distinct motifs approximating the largest fraction of data in $\mathfrak{T}$ [6]. Fixed a distance function δ and a constant α, we define a motif-based temporal alphabet as:

$$\mathcal{P} = \{\delta(T, \mu) \leq \alpha \mid T \in \mathfrak{T}, \mu \in \mathcal{A}_{\mathfrak{T}}, \alpha \in \mathbb{R}\}.$$

This is a first step towards dealing with intervals natively, as a given non-temporal item $p \in \mathcal{P}$ is true on T_{ij} if and only if the behaviour of T within the segment is close enough to the motif μ encapsulated by p. However, the definition above is insufficient to guarantee that the computation of local support is unbiased. For example, we can consider a temporal item enriched with Allen's $\langle B \rangle$ relation: if it is true on T_{ij}, then it automatically inflates the support computation, as it is trivially true on all the interval $T_{i(j+1)} \dots T_{iN}$.

To ensure a fair support computation, we must "anchor" temporal items to a specific temporal frame, that is, the temporal relations must be applied to the subset of intervals having at least one non-temporal item true on them, instead of possibly any interval. This consideration is also crucial when translating an itemset to natural language, as it naturally filters out two tricky scenarios: firstly, the case in which a conjunction of only temporal items is considered, as it is no clear the set of intervals they refer to, and, secondly, it ensures that two non-temporal items wrapping motifs of different lengths are not mixed, as the number of intervals the two items can be true at the same time is 0.

▶ **Definition 4.** *An itemset X is said to be* anchored *iff contains at least one non-temporal item, and all non-temporal items $\Omega \subseteq X$, called* anchor *of X, are based on motifs of the same length. Let $l(\Omega)$ denote the length of the interval on which an anchor may hold.*

In this way, the length of intervals that may potentially satisfy the entire itemset is fixed, allowing us to define the frequency of that itemset in a truly representative way.

▶ **Definition 5.** *Let $\mathfrak{T}$ be a temporal dataset, $\Lambda_{\mathcal{P}}$ be the set of temporal items built on the motif-based alphabet $\mathcal{P}$, let $X \subseteq \Lambda_{\mathcal{P}}$ be an anchored itemset, and let $\Omega \subseteq X$ be its anchor. The* motif-based local and global supports *of X on some instance $\mathcal{T} \in \mathfrak{T}$ are defined respectively as:*

$$mblfs_{\mathcal{T}}(X) = \frac{|\{[i, j] \in I(D) \mid T, [i, j] \Vdash X\}|}{|\{[i, j] \in I(D) \mid j - i + 1 = l(\Omega)\}|}, \quad mbgs_{\mathfrak{T}}^{s_l}(X) = \frac{|\{\mathcal{T} \in \mathfrak{T} \mid mblfs_{\mathcal{T}}(X) \geq s_l\}|}{|\mathfrak{T}|}.$$

Using the motif-based support, we proceed to comment our experiments.

4 Experiments

We consider two well-known public datasets concerning human gestures and gait recognition, namely NATOPS [13], from which we consider an arm gesture called "I have command", and HuGaDB [3], from which we consider the "Walk" movement, aiming to describe insightful common patterns and to express them in natural language.

In both cases, we mine frequent itemsets by leveraging ModalAPRIORI. First of all, we extract the top 5 motifs with length 10 and the top 3 with length 20, in order to capture qualitatively appreciable patterns; the two class of motifs encodes a behaviour expressed respectively in nearly half a second and one second: shorter subsequences would bring little informativeness, while longer one would be too coarse. We establish two thresholds for minimum local and global support, respectively s_l and s_g, to 0.1, which is relatively low: this is not a problem, as the vast majority of the association rules generated after the itemset extraction phase (14233 for "I have command", 253 for "Walking") are filtered out by leveraging confidence and *lift* [2] meaningfulness measures. In particular, the higher the lift, the more the antecedent and the consequent of a rule are positively correlated.

The motif-based alphabet is generated considering the Z-normalized Euclidean distance δ. We choose the threshold α associated to a specific motif μ to be the tenth percentile of the values in the distance profile of μ; in this way, we ensure each single non-temporal item to be frequent. We enrich the propositional alphabet with the set of temporal items $\Lambda_{\mathcal{P}}$ obtained by considering every Allen's relation.

Even if we did not graphically shown an example of "I have command" gesture, we try to describe a chunk of it with the following rule we extracted, which exhibits a perfect confidence (of 1.0) and an high lift (of 6.7). The rule is expressed in a compact format, fixing a coordinate, leveraging superscripts to provide an intuition about the movement captured by the motif underlying each item, and subscripts to indicate the length of the motif and which body part it refers to (e.g., *re* for right elbow, *rh* for right hand).

$$y^{up}_{20,\,re} \wedge z^{front}_{20,\,rh} \Rightarrow \langle O \rangle x^{left\&invert}_{10,\,rh} \wedge \langle B \rangle y^{rest\&up}_{10,\,re}$$

The rule above can be read as *whenever the right hand of the operator is completely stretching in front of him/her and their elbow goes all the way up on the y-axis, the same elbow started the movement range in a rest position and, near the end of the movement range, the operator's right hand is moving to the left, but will soon change direction.*

The following pair of rules describes a non-trivial behaviour typical of the walking gait; note that *rf* (*lf*) stands for right (left) foot, while *rt* (*lt*) stands for right (left) thigh.

$$x^{front}_{10,\,rf} \Rightarrow \langle A \rangle x^{up}_{10,\,lt}, \quad x^{back}_{10,\,lf} \Rightarrow \langle A \rangle x^{up}_{10,\,rt}$$

The rules states that *when the right (left) foot accelerates forward (backwards) for approximately half of a second, then the left (right) thigh accelerates upward immediately after.*

5 Conclusions

In this paper we introduced a motif-based approach to temporal alphabet definition and rule extraction. By leveraging motifs-based frequently recurring patterns in time series, we obtained a more interpretable and structurally robust framework for mining temporal association rules, solving the biases introduced by naive alphabet definitions and enhancing semantic clarity. We formalized the concept of anchored itemsets and introduced a novel definition of motif-based local and global support, ensuring that patterns are both meaningful and computationally tractable. Experimental validation on temporal datasets demonstrates the expressiveness and interpretability of the extracted rules, showing promise for applications in explainable temporal data analysis.

References

1. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of 20th International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.
2. Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 255–264, 1997. doi:10.1145/253260.253325.
3. Roman Chereshnev and Attila Kertész-Farkas. Hugadb: Human gait database for activity recognition from wearable inertial sensor networks. In *Analysis of Images, Social Networks and Texts - 6th International Conference, AIST 2017, Moscow, Russia, July 27-29, 2017, Revised Selected Papers*, pages 131–141. Springer, 2017. doi:10.1007/978-3-319-73013-4_12.
4. B.Y. Chiu, E.J. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 493–498. ACM, 2003. doi:10.1145/956750.956808.
5. J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.
6. S. Imani, F. Madrid, W. Ding, S. Crouter, and E. Keogh. Matrix profile xiii: Time series snippets: A new primitive for time series data mining. In *Proc. of the IEEE international conference on big knowledge (ICBK)*, pages 382–389. IEEE, 2018.
7. Carl H Lubba, Sarab S Sethi, Philip Knaute, Simon R Schultz, Ben D Fulcher, and Nick S Jones. catch22: Canonical time-series characteristics: Selected through highly comparative time-series analysis. *Data mining and knowledge discovery*, 33(6):1821–1852, 2019. doi:10.1007/S10618-019-00647-X.
8. Federico Manzella, Giovanni Pagliarini, Guido Sciavicco, and Ionel Eduard Stan. Interval temporal random forests with an application to COVID-19 diagnosis. In Carlo Combi, Johann Eder, and Mark Reynolds, editors, *28th International Symposium on Temporal Representation and Reasoning, TIME 2021, September 27-29, 2021, Klagenfurt, Austria*, volume 206 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.TIME.2021.7.
9. M. Milella, G. Pagliarini, G. Sciavicco, and I.E Stan. Modalfp-growth: Efficient extraction of modal association rules from non-tabular data. In *Proc. of the 25th Italian Conference on Theoretical Computer Science (ICTCS)*, volume 3811 of *CEUR*, pages 241–254. CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3811/paper150.pdf.
10. Mauro Milella, Giovanni Pagliarini, Guido Sciavicco, and Ionel Eduard Stan. ModalAssociationRules.jl. Software, version 0.1.0., swhId: swh:1:dir:697da0b30a22cd23450ab445a887ebf1a602db8f (visited on 2025-09-18). URL: https://github.com/aclai-lab/ModalAssociationRules.jl, doi:10.4230/artifacts.24783.
11. Mauro Milella, Giovanni Pagliarini, Guido Sciavicco, and Ionel Eduard Stan. Sole.jl. Software, version 0.6.2., swhId: swh:1:dir:dd723aee72578208606649ff12168e891cdae221 (visited on 2025-09-18). URL: https://github.com/aclai-lab/Sole.jl, doi:10.4230/artifacts.24782.
12. Guido Sciavicco and Ionel Eduard Stan. Knowledge extraction with interval temporal logic decision trees. In Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald, editors, *27th International Symposium on Temporal Representation and Reasoning, TIME 2020, September 23-25, 2020, Bozen-Bolzano, Italy*, volume 178 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.TIME.2020.9.
13. Y. Song, D. Demirdjian, and R. Davis. Tracking body and hands for gesture recognition: NATOPS aircraft handling signals database. In *Proc. of the 9th IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, pages 500–506, 2011.

14 I.E Stan, G. Sciavicco, E. Muñoz-Velasco, G. Pagliarini, M. Milella, and A. Paradiso. On modal logic association rule mining. In *Proc. of the 23rd Italian Conference on Theoretical Computer Science (ICTCS)*, volume 3284 of *CEUR*, pages 53–65. CEUR-WS.org, 2022. URL: `https://ceur-ws.org/Vol-3284/492.pdf`.

15 Y. Zhu, Z. Zimmerman, N.S. Senobari, C-C.M. Yeh, G.J. Funning, A. Mueen, P. Brisk, and E.J. Keogh. Matrix profile II: exploiting a novel algorithm and GPUs to break the one hundred million barrier for time series motifs and joins. In *Proc. of the 16th International Conference on Data Mining (ICDM)*, pages 739–748. IEEE, 2016.

Temporal Considerations in DJ Mix Information Retrieval and Generation

Alexander Williams[1]
Centre for Digital Music, Queen Mary University of London, United Kingdom

Gregor Meehan[1]
Centre for Digital Music, Queen Mary University of London, United Kingdom

Stefan Lattner
Sony CSL, Paris, France

Johan Pauwels
Centre for Digital Music, Queen Mary University of London, United Kingdom

Mathieu Barthet
Aix-Marseille Univ CNRS PRISM, France
Centre for Digital Music, Queen Mary University of London, United Kingdom

Abstract

Music is the art of arranging sounds in time so as to produce a continuous, unified, and evocative composition. Electronic dance music (EDM) is a collection of musical sub-genres produced using computers and electronic instruments and often presented through the medium of DJing, where tracks are curated and mixed sequentially into a continuous stream of music to offer unique listening and dancing experiences over time periods ranging from several minutes to several hours. A DJ's actions and decisions occur at several levels of temporal granularity, from real-time audio manipulation (e.g. of tempo) for smooth inter-track transitions to long-term planning of track selection and sequencing for mix content and flow. While human DJs can instinctively operate across these different temporal resolutions, replicating this capability in an end-to-end automated DJing system presents significant challenges. In this paper, we analyse existing works in DJ mix information retrieval and generation from this temporal perspective. We first explain the close link between DJing and the temporal notion of musical rhythm, then describe a framework for categorising DJing actions by temporal granularity. Using this framework, we summarise and contrast potential approaches for automating and augmenting sequential DJ decision making, and discuss the unique characteristics of DJ mix track selection as a sequential recommendation task. In doing so, we hope to facilitate the implementation of more robust and complete automated DJing systems in future research.

2012 ACM Subject Classification Applied computing → Sound and music computing; Computing methodologies → Control methods; Computing methodologies → Planning under uncertainty

Keywords and phrases Music Information Retrieval, Computational Creativity, Recommender Systems, Electronic Dance Music, DJ

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.20

Category Short Paper

Funding Alexander Williams and Gregor Meehan are research students at the UKRI Centre for Doctoral Training in Artificial Intelligence and Music, supported by UK Research and Innovation [grant number EP/S022694/1] and Queen Mary University of London. Alexander Williams is also supported by Sony CSL.

[1] Alexander Williams and Gregor Meehan contributed equally to this work.

1 Introduction

Electronic Dance Music (EDM) has seen a huge growth in popularity in recent years, from a relatively underground movement to a mainstream industry worth billions [30]. EDM is often consumed through the medium of DJ performances, leading to a corresponding increase in interest in DJing among EDM fans. Computer programs that could perform or assist with the creative task of DJing have the potential to democratise access to high quality continuous music mixes [42], and therefore have seen a corresponding increase in research attention [45]. At the highest abstraction, we can specify two key tasks for an end-to-end computational DJ system: long-term track selection and sequencing, and real-time audio manipulation for inter-track DJ mixing. However, existing computational DJ systems are typically not end-to-end: to simplify their implementation, they are instead deconstructed into a hierarchy of components, some of which involve manual curation or simple rule-based heuristics. In this paper, we aim to highlight the challenges of implementing an intelligent and fully end-to-end DJing system by viewing its actions from a temporal perspective. We first explain the link between DJing and the temporal notion of musical rhythm, then describe how a DJ's actions can be divided across different levels of temporal granularity. Through this framework, we analyse existing works in automated DJ mixing and how its unique challenges prevent straightforward application of related techniques in sequential music recommendation. Finally, we discuss challenges relating to the adoption of computational DJ systems and outline future research directions.

2 Rhythm and DJing

Music is typically organised into beats and repetitive rhythmic patterns that collectively form the foundation of musical rhythm [32]. Here rhythm refers to the medium-scale temporal organisation of sound, characterised by the arrangement of events, beats, and accents over time. It is composed of several key components: tempo, i.e. the speed of the pulse; timing, i.e. the placement of events relative to pulses; metre, i.e. the structural relationship among pulses; and grouping, i.e. the phrasing of musical ideas independent of metre.

Rhythmic properties are particularly important in EDM, as many EDM sub-genres are defined by their timbral palette and associated rhythm patterns. Further, structural changes and progression in EDM tracks / DJ mixes are usually indicated by an evolution of timbre and rhythm rather than melody and harmony [34, 22, 46, 24, 44]. EDM tracks, and subsequently DJ mixes, are typically built around repeating loops of melodies, vocals, drums, and sound effects (FX) that change and are layered over time – for example, through an element being added or removed from the mix or being affected by continuous temporal processes such as FX and synthesizer parameter automation [20, 37] – to produce rhythmic and timbral variation and progression in the composition. Modelling these rhythmic properties is therefore an important task in automated DJing systems, as we discuss further in Section 3 below.

3 Temporal Abstraction in DJ Tasks

We next examine in more detail the necessary capabilities of an end-to-end automated DJing system. The two key tasks in DJing are transitioning from track to track (i.e. DJ mixing) and selecting and ordering the tracks to be played. In the remainder of this section, we discuss existing approaches to each of these tasks, dividing DJing actions into three temporal levels as shown in Fig. 1. From most to least granular, these are:

1. **Transition-level**: the immediate actions needed to transition from the current song to a given next song;
2. **Track-level**: the choice of what the next song should be; and
3. **Mix-level**: the overall flow and content of the entire mix.

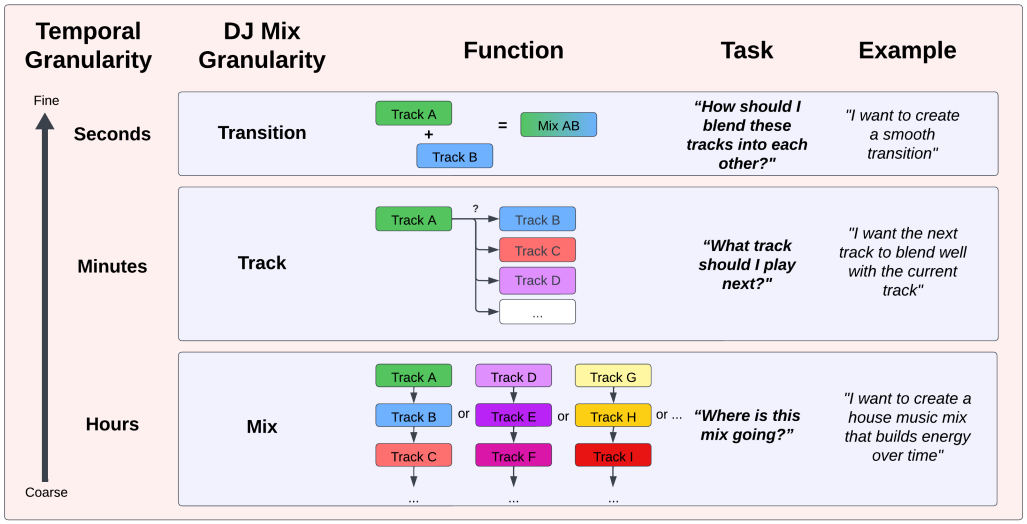


Figure 1 The temporal abstraction hierarchy for key sub-tasks in DJ mix construction.

3.1 Automatic DJ Mixing

DJ mixing can be formulated as a real-time sequential decision-making problem, where the objective is to apply a sequence of audio signal transformations to two (or more) concurrently playing tracks to optimise the perceptual quality of the resulting audio mixture for the listener. These decisions take place entirely at the transition-level (1), using features relating to the rhythm and other characteristics of the current and incoming tracks.

DJ transitions are considered an art-form and their perceived quality is highly subjective and dependent on the audio tracks being mixed and the cultural context in which the transition takes place. Nevertheless, common techniques exist and many works have proposed automatic DJ mixing systems, attempting to implement probabilistic rules informed by signal processing to produce systems exhibiting what each author believes constitutes valid DJ mixing. For example, (beat-matched) crossfading over fixed-length temporal horizons is commonly applied [7, 33, 40, 42, 10, 23, 3, 27, 26, 2, 38]. Other approaches include 3-band equaliser mixing [21, 38], avoiding vocal clashes [42, 38], double drop / rolling / relaxed transition [42], looping out [33], looping to produce a drum roll-like effect [3], adding samples [27] or an MC-like voiceover [33], slam / cut / switching [40, 2, 3], echo out [40, 2], and power down [40]. While such rules can produce reasonable transitions, the predictability arising from using the same limited transition techniques mean that computer-generated transitions are more likely to be identified and negatively viewed by listeners in repeated exposures [18].

A notable exception to rule-based systems is a deep neural network to generate audio-dependent control trajectories for DJ transitions [7]. Although described as having developed a unique style, it also employed unfamiliar mixing techniques and subsequently its performance was only considered on par with rule-based methods. This highlights the challenge of

predicting effective temporal control sequences, which must balance a degree of unpredictability with some adherence to established mixing norms and contextual appropriateness to deliver robust mixes that are satisfactory at a psychoacoustic level [41]. So far, no automatic DJ transition generation system has matched the performance of human DJs.

3.2 DJ Track Selection and Sequencing

DJ track selection is influenced by a DJ's own curatorial interests and the context of the mix (e.g. a pop music night at a club versus an EDM festival). Sequencing depends on both transition compatibility and the DJ's longer-term goals: for example, after playing several songs with high tempo, a DJ may choose to play lower energy tracks to give the audience a break. To successfully select and sequence tracks, generative DJ mix systems must therefore act across all three levels of temporal granularity. However, operating intelligently across all three temporal levels simultaneously is a fundamental difficulty in automated DJ mixing, and, to the best of our knowledge, no existing works attempt to do so. Most existing systems operate primarily at levels (1) and (2) in their sequencing algorithms: to ensure that the automated transitions discussed in Section 3.1 can be applied effectively, they select an appropriate next track by "mixability", i.e. by compatibility in tempo, rhythm, or harmony. This compatibility can be either fine-grained at the transition points [19, 42, 38] (at transition-level), based on coarser similarity of overall acoustic features [40] (at track-level), or a combination of both [42]. Other works [11, 2, 33] place more emphasis on flow at the mix-level but forgo mix-level content curation and/or transition-level compatibility, while DJ-MC [25] focuses only on the higher two levels.

The above temporal framework also helps explain why existing sequential music recommendation algorithms cannot be straightforwardly applied in the DJ mixing context, even though DJ track selection is closely related to tasks such as playlist generation [13, 35, 1] and next song recommendation [16, 43, 36]. In particular, transition-level factors are unique to DJ mixing, and are not modelled by generic sequential music recommenders. However, these transition-level constraints exert a considerable influence on the other levels and therefore on the overall system: track-level choices require consideration of transition-level compatibility, which in turn affects mix-level track selection. There are also practical difficulties arising from the need to model all three levels of granularity, from the coarse sequence-level view of the selected tracks down to musical content at the beat or frame level. Furthermore, although some existing sequential music recommender systems or DJing systems do implement curatorial filtering (e.g. to specific genres or "vibes", as in [33]), they typically do not operate with long-term goals in mind in the same way a DJ would. It is not clear whether such behaviour could be explicitly encouraged during model training, and any such effort would likely require a large dataset of DJ mixes with associated audio and track labels. However, no such corpus is currently publicly available.

4 Adoption and Ethical Challenges

Many DJs are concerned about how automation may affect their livelihoods [39], as computational DJing is already reshaping both live performance [5] and radio or home listening contexts [6, 2, 4]. For instance, iHeartMedia – a global network of local radio stations – laid off up to 850 DJs and producers in favour of software to schedule music, mix songs, and mimic the voice of radio hosts [9, 17, 14]. Yet, DJ practices have continually evolved alongside technological innovation [29]. Automatic beat tracking and alignment, key / tempo detection and transposition, and various visualisation techniques [46, 15, 8] to aid musical understanding

and inform DJ decision making at multiple temporal levels are now ubiquitous in digital DJ workflows. Rather than replacing DJs, computational tools, like those discussed in this paper, can complement and expand contemporary practice. While quality-cost considerations may drive some autonomous applications [28], socio-cultural factors – particularly the importance of the dynamic between DJs and live audiences [31] – ensure continued demand for human DJs and intelligent mixing and recommendation systems can be used to support their creative expression. The key challenge comes in developing application use-cases that respect DJs' artistic autonomy while leveraging the potential of these technologies to reduce various task complexities, enable enhanced performance, and augment creative decision making [12].

5 Conclusion and Future Directions

This paper outlines how a multi-resolution temporal framework is critical for both feature extraction and the development of hierarchical, modular systems capable of intra/inter-track mixing, track-level selection, and mix-level sequencing in an EDM DJing context. We also distinguish between DJ track selection and general sequential music recommendation based on this framework, and discuss potential consequences of this technology being democratised. In previous work, we identified that audio features that capture and emphasise timbral and rhythmic features are valuable for the analysis of DJ mixes at arbitrarily fine temporal intervals [46], particularly in computing DJ transition properties such as length and relative smoothness. In the future, we hope to apply such representations at various temporal resolutions for DJ mix information retrieval and to integrate approaches at multiple temporal levels to assist in various artist-centric, co-creative scenarios such as informative visualisation, multi-track mixing, and real-time assistance in DJ mix construction.

References

1 Walid Bendada, Guillaume Salha-Galvan, Thomas Bouabça, and Tristan Cazenave. A scalable framework for automatic playlist continuation on music streaming services. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 464–474, 2023. doi:10.1145/3539618.3591628.

2 Rachel M Bittner, Minwei Gu, Gandalf Hernandez, Eric J Humphrey, Tristan Jehan, P Hunter McCurry, and Nicola Montecchio. Automatic Playlist Sequencing and Transitions. In *Proceedings of the 18th ISMIR Conference*, Suzhou, China, 2017.

3 Jaume Parera Bonmati. DJ Codo Nudo: a novel method for seamless transition between songs for electronic music. Master's thesis, Universitat Pompeu Fabra, Barcelona, 2016. URL: http://mtg.upf.edu/system/files/publications/Jaume-Parera-Master-thesis-2016.pdf.

4 Georgina Born, Jeremy Morris, Fernando Diaz, and Ashton Anderson. Artificial Intelligence, Music Recommendation, and the Curation of Culture: A White Paper, June 2021.

5 Becky Buckle. East London's first "AI-rave" receives mixed reviews. *Mixmag*, March 2023. URL: https://mixmag.net/read/east-london-nightclub-hosts-algorhythm-rave-with-ai-generated-music-tech.

6 Nyshka Chandran. RA Pro Newsletter: AI's role in independent radio, May 2025. URL: https://ra.co/news/82747.

7 Bo-Yu Chen, Wei-Han Hsu, Wei-Hsiang Liao, Marco A. Martínez Ramírez, Yuki Mitsufuji, and Yi-Hsuan Yang. Automatic DJ Transitions with Differentiable Audio Effects and Generative Adversarial Networks. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 466–470, May 2022. ISSN: 2379-190X. doi:10.1109/ICASSP43922.2022.9746663.

8 Ya-Xi Chen and René Klüber. ThumbnailDJ: Visual Thumbnails of Music Content. In *11th International Society for Music Information Retrieval Conference*, 2010.

9 Martin Clancy. *Reflections on the Financial and Ethical Implications of Music Generated by Artificial Intelligence*. PhD Thesis, Trinity College, Dublin, Ireland, 2021.

10 Dave Cliff. Hang the DJ: Automatic sequencing and seamless mixing of dance-music tracks. Technical report, Hewlett-Packard Laboratories, 2000.

11 Dave Cliff. hpDJ: An Automated DJ with Floorshow Feedback. In Kenton O'Hara and Barry Brown, editors, *Consuming Music Together: Social and Collaborative Aspects of Music Consumption Technologies*, Computer Supported Cooperative Work, pages 241–264. Springer Netherlands, Dordrecht, 2006. doi:10.1007/1-4020-4097-0_12.

12 Emmanuel Deruty, Maarten Grachten, Stefan Lattner, Javier Nistal, and Cyran Aouameur. On the Development and Practice of AI Technology for Contemporary Popular Music Production. *Transactions of the International Society for Music Information Retrieval*, 5(1):35–49, February 2022. Ubiquity Press. doi:10.5334/tismir.100.

13 Anna Gatzioura, João Vinagre, Alípio Mário Jorge, and Miquel Sanchez-Marre. A hybrid recommender system for improving automatic playlist continuation. *IEEE Transactions on Knowledge and Data Engineering*, 33(5):1819–1830, 2019.

14 Nell Geraets. Thy has been on the radio for six months. Turns out she isn't real. *The Sydney Morning Herald*, April 2025. Section: TV & radio. URL: https://www.smh.com.au/culture/tv-and-radio/thy-has-been-on-the-radio-for-six-months-turns-out-she-isn-t-real-20250424-p5ltxi.html.

15 Masatoshi Hamanaka. Real-time Future-rhythm Visualizer for DJ Performance. In *Proceedings of the Sound and Music Computing Conference*, Porto, Portugal, July 2024.

16 Casper Hansen, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi, and Mounia Lalmas. Contextual and sequential user embeddings for large-scale music recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 53–62, 2020. doi:10.1145/3383313.3412248.

17 Drew Harwell. iHeartMedia laid off hundreds of radio DJs. Executives blame AI. DJs blame the executives. *The Washington Post*, January 2020. URL: https://www.washingtonpost.com/technology/2020/01/31/iheartmedia-radio-artificial-intelligence-djs/.

18 Jordan V. Hawkins. *Automating Music Production with Music Information Retrieval*. Undergraduate Thesis, The Ohio State University, Ohio, USA, March 2013. URL: https://kb.osu.edu/handle/1811/54438.

19 Tatsunori Hirai, Hironori Doi, and Shigeo Morishima. MusicMixer: Automatic DJ System Considering Beat and Latent Topic Similarity. In *Proceedings, Part I, of the 22nd International Conference on MultiMedia Modeling - Volume 9516*, MMM 2016, pages 698–709, Berlin, Heidelberg, January 2016. Springer-Verlag. doi:10.1007/978-3-319-27671-7_59.

20 Aline Honingh, Maria Panteli, Thomas Brockmeier, David Iñaki López Mejía, and Makiko Sadakata. Perception of Timbre and Rhythm Similarity in Electronic Dance Music. *Journal of New Music Research*, 44(4):373–390, October 2015. doi:10.1080/09298215.2015.1107102.

21 Cheng-Zhi Anna Huang, Hendrik Vincent Koops, and Ed Newton-Rex. AI Song Contest: Human-AI Co-Creation in Songwriting. In *21st International Society for Music Information Retrieval Conference,*, Montreal, Canada, 2020.

22 Thor Kell and George Tzanetakis. Empirical Analysis of Track Selection And Ordering In Electronic Dance Music Using Audio Feature Extraction. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, Curitiba, Brazil, 2013.

23 Adrian Kim, Soram Park, Jangyeon Park, Jung-Woo Ha, Taegyun Kwon, and Juhan Nam. Automatic DJ Mix Generation Using Highlight Detection. In *Extended abstracts for the Late-Breaking Demo Session of the 18th International Society for Music Information Retrieval Conference*, Suzhou, China, October 2017.

24 Taejun Kim, Minsuk Choi, Evan Sacks, Yi-Hsuan Yang, and Juhan Nam. A Computational Analysis of Real-World DJ Mixes Using Mix-to-Track Subsequence Alignment. In *Proc. of the 21st Int. Society for Music Information Retrieval Conf.*, Montreal, Canada, 2020.

25 Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, Istanbul, Turkey, May 2015. URL: https://www.ifaamas.org/Proceedings/aamas2015/aamas/p591.pdf.

26 Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. The Right Music at the Right Time: Adaptive Personalized Playlists Based on Sequence Modeling. *MIS Quarterly*, 43(3):765–786, January 2019. doi:10.25300/MISQ/2019/14750.

27 Yin-Tzu Lin, Chuan-Lung Lee, Jyh-Shing Roger Jang, and Ja-Ling Wu. Bridging Music Using Sound-Effect Insertion. *IEEE MultiMedia*, 22(4):30–38, October 2015. Conference Name: IEEE MultiMedia. doi:10.1109/MMUL.2015.71.

28 Nigel P. Melville, Lionel Robert, and Xiao Xiao. Putting humans back in the loop: An affordance conceptualization of the 4th industrial revolution. *Information Systems Journal*, 33(4):733–757, 2023. doi:10.1111/isj.12422.

29 Ed Montano. 'How do you know he's not playing Pac-Man while he's supposed to be DJing?': technology, formats and the digital future of DJ culture. *Popular Music*, 29(3):397–416, October 2010. Publisher: Cambridge University Press. doi:10.1017/S0261143010000449.

30 Mark Mulligan. The ims business report 2023, 2023. URL: https://www.internationalmusicsummit.com/business-report.

31 Keith Munro, Ian Ruthven, and Perla Innocenti. Can you feel it? The information behaviour of creative DJs. *Journal of Documentation*, 79(4), January 2022. doi:10.1108/JD-05-2022-0106.

32 Meinard Müller. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21945-5.

33 Metehan Pala. RoboDJ: automatic dj mix generation using Spotify Web API and machine learning methods. Master's thesis, Politecnico Milano, Milan, Italy, 2021. URL: https://www.politesi.polimi.it/handle/10589/179255.

34 Maria Panteli, Bruno Rocha, Niels Bogaards, and Aline Honingh. A model for rhythm and timbre similarity in electronic dance music. *Musicae Scientiae*, 21(3):338–361, September 2017. Publisher: SAGE Publications Ltd. doi:10.1177/1029864916655596.

35 Keigo Sakurai, Ren Togo, Takahiro Ogawa, and Miki Haseyama. Controllable Music Playlist Generation Based on Knowledge Graph and Reinforcement Learning. *Sensors (Basel, Switzerland)*, 22(10):3722, May 2022. doi:10.3390/s22103722.

36 Pavan Seshadri, Shahrzad Shashaani, and Peter Knees. Enhancing sequential music recommendation with negative feedback-informed contrastive learning. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 1028–1032, 2024. doi:10.1145/3640457.3688188.

37 Jeremy W. Smith. The Functions of Continuous Processes in Contemporary Electronic Dance Music. *Music Theory Online*, 27(2), June 2021. URL: https://mtosmt.org/issues/mto.21.27.2/mto.21.27.2.smith.html.

38 Robert Sowula and Peter Knees. Mosaikbox: Improving Fully Automatic DJ Mixing Through Rule-Based Stem Modification and Precise Beat-Grid Estimation. In *Proceedings of the 25th International Society for Music Information Retrieval Conference*, pages 850–857, San Francisco, California, USA and Online, November 2024. ISMIR. doi:10.5281/zenodo.14877463.

39 Anna Spagnolli, Diletta Mora, Matteo Fanchin, Valeria Orso, and Luciano Gamberini. Automation and Creativity: A Case Study of DJs' and VJs' Ambivalent Positions on Automated Visual Software. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–11, Honolulu HI USA, April 2020. ACM. doi:10.1145/3313831.3376463.

40 Florian Thalmann, Lucas Thompson, and Mark Sandler. A User-Adaptive Automated DJ Web App with Object-Based Audio and Crowd-Sourced Decision Trees. In *Web Audio Conference*, Berlin, Germany, September 2018.

41 Amelia Skye Turrell. *The Emotional & Neurological Effects of Expectancy and Tension Within Electronic Dance Music*. PhD Thesis, University of Kent, March 2022. doi:10.22024/UniKent/01.02.93493.

42 Len Vande Veire and Tijl De Bie. From raw audio to a seamless mix: creating an automated DJ system for Drum and Bass. *EURASIP Journal on Audio, Speech, and Music Processing*, 2018(1):13, September 2018. doi:10.1186/s13636-018-0134-8.

43 Dongjing Wang, Xin Zhang, Yuyu Yin, Dongjin Yu, Guandong Xu, and Shuiguang Deng. Multi-view enhanced graph attention network for session-based music recommendation. *ACM Transactions on Information Systems*, 42(1):1–30, 2023. doi:10.1145/3592853.

44 Brian C. Wesolowski and Alex Hofmann. There's More to Groove than Bass in Electronic Dance Music: Why Some People Won't Dance to Techno. *PLoS ONE*, 11(10):e0163938, October 2016. doi:10.1371/journal.pone.0163938.

45 Alexander Williams, Stefan Lattner, and Mathieu Barthet. A Review of Music Information Retrieval and Computational Creativity Tools for Electronic Dance Music and DJing. (forthcoming).

46 Alexander Williams, Haokun Tian, Stefan Lattner, Mathieu Barthet, and Charalampos Saitis. Deep Learning-based Audio Representations for the Analysis and Visualisation of Electronic Dance Music DJ Mixes. In *AES International Symposium on AI and the Musician*, Boston, MA, USA, June 2024. Audio Engineering Society.

A Translation of Probabilistic Event Calculus into Markov Decision Processes

Lyris Xu
Dept. of Information Studies, University College London, UK

Fabio Aurelio D'Asaro
Dip. di Studi Umanistici, Università del Salento, Lecce, Italy

Luke Dickens
Dept. of Information Studies, University College London, UK

Abstract

Probabilistic Event Calculus (PEC) is a logical framework for reasoning about actions and their effects in uncertain environments, which enables the representation of probabilistic narratives and computation of temporal projections. The PEC formalism offers significant advantages in interpretability and expressiveness for narrative reasoning. However, it lacks mechanisms for goal-directed reasoning. Our work bridges this gap by developing a formal translation of PEC domains into Markov Decision Processes (MDPs), introducing the concept of "action-taking situations" to preserve PEC's flexible action semantics. The resulting PEC-MDP formalism enables the extensive collection of algorithms and theoretical tools developed for MDPs to be applied to PEC's interpretable narrative domains. We demonstrate how the translation supports both temporal reasoning tasks and objective-driven planning, with methods for mapping learned policies back into human-readable PEC representations, maintaining interpretability while extending PEC's capabilities.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning; Computing methodologies → Planning and scheduling; Computing methodologies → Temporal reasoning; Computing methodologies → Markov decision processes

Keywords and phrases Probabilistic Event Calculus, Markov Decision Processes, Temporal Projection, Narrative Reasoning

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.21

Category Short Paper

Related Version *Full Version*: `https://arxiv.org/abs/2507.12989` [15]

Supplementary Material *Software*: `https://github.com/LyrisX02/PEC-MDP`
archived at `swh:1:dir:c557c388a86ca997275d269b6f53c804051a8727`

Funding *Fabio Aurelio D'Asaro*: Partially supported by the project FAIR – Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

1 Introduction

Reasoning about actions and their effects in dynamic, uncertain environments is a fundamental challenge in Artificial Intelligence (AI). The importance of narrative reasoning – the ability to represent and reason about sequences of events and their causal relationships over time – has been recognised since the early days of AI, leading to various temporal reasoning formalisms [10, 1, 5, 12]. Building on these early foundations, the Probabilistic Event Calculus (PEC) [3] has emerged as a powerful framework for representing and reasoning about uncertain scenarios. PEC extends the Event Calculus (EC) [5, 11], to provide an "action-language" style framework for modelling actions, their effects, and the evolution of world states over time under uncertainty. PEC-style frameworks offer highly interpretable

and flexible representations of complex narratives, with demonstrated applications in domains such as medicine, environmental monitoring, and commonsense reasoning [2, 3]. Markov Decision Processes (MDPs), meanwhile, have established themselves as a powerful framework for modelling time-evolving systems controlled by an agent. MDPs and their variants are widely used across AI for decision-making under uncertainty and serve as the foundation for many statistical and reinforcement learning algorithms. While MDPs excel at control optimisation problems, PEC and its variants offer superior narrative interpretability but lack mechanisms for learning goal-directed behaviour.

A translation between these frameworks presents a compelling opportunity to bridge this gap, combining PEC's human-readable representation with MDP's computational efficiency and reinforcement learning capabilities. Such a bridge would allow for both efficient PEC implementation and the application of statistical learning techniques to narrative reasoning tasks. Towards this goal, we have developed a novel translation of PEC into an MDP framework, termed the PEC-MDP. This translation preserves the core assumptions and semantics of PEC while enabling the application of a wide range of MDP-based algorithms to PEC's human-readable domains.

The key contributions of our work include:

- A formal translation of PEC domains to PEC-MDPs, with a Python implementation made available through a shared repository.
- An approach for performing temporal projection via the PEC-MDP formalism.
- A novel approach to planning under uncertainty in PEC domains.

2 Background

Probablistic Event Calculus. A PEC domain $\mathcal{D}$ comprises a finite, non-empty set of fluents $\mathcal{F}$ and values $\mathcal{V}$, a finite set of actions $\mathcal{U}$, a set of fluent states $\tilde{\mathcal{S}}$, and a non-empty set of time instants $\mathcal{I}$. Fluents, as in classical narrative reasoning formalisms [12, 9], refer to properties of the world which may be affected by actions taken. A fluent state $\tilde{S} \in \tilde{\mathcal{S}}$ is an assignment of values to all fluents of the domain, while a partial fluent state $\tilde{X} \in \tilde{\mathcal{X}}$ is a subset of value-assigned fluents, i.e., $\tilde{X} \subseteq \tilde{S}$ for some $\tilde{S} \in \tilde{\mathcal{S}}$. PEC uses a set of action-language style propositions to specify probabilistic narrative information. A domain consists of:

1. A finite set of *v-propositions* which detail the values that a fluent may take.
2. Exactly one *i-proposition* which specifies the probabilities of initial fluent states that hold at the minimum time instant.
3. A finite set of *c-propositions* which model the causal effects of actions, each specifying a set of preconditions for effects to take hold (where at least one action is true), a partial fluent $\tilde{X}$ for each set of effects, and their corresponding probabilities.
4. A set of *p-propositions* modelling the occurrence of actions, each specifying an action, a time instant, the probability for the action to be taken, and optional fluent preconditions.

PEC supports a possible-worlds semantics, where a world is an evolution of an environment over time. Using this semantics, *temporal projection* computes the probability of fluent states or partial fluent states at future time points given an initial state distribution and a narrative of action occurrences. See [3] for more.

Markov Decision Processes. An MDP is defined as the tuple $(\mathcal{S}, \mathcal{A}, p_0, T, R)$, where $\mathcal{S}$ and $\mathcal{A}$ are finite sets of *states* and *actions* respectively (distinct from PEC states and actions). The initial state distribution is given by p_0, where the probability of starting at time $t = 0$ in state s is given by $p_0(s)$. Transition dynamics are given by transition function

$T(s, a, s') = \Pr(s_{t+1} = s' | a_t = a, s_t = s)$, while the reward function $R(s, a, s') = r$ maps each state transitions to a numerical rewards. Behaviours in an MDP are encoded in a policy μ. Stationary policies map states to actions independent of time, either deterministically $\mu(s) = a$ or stochastically $\mu(s, a) = \Pr(a_t = a | s_t = s)$. Non-stationary policies [6] allow time-dependent mappings, μ_t, where if $t \neq t'$, then $\mu_t(s, a)$ can be different from $\mu_{t'}(s, a)$.

3 PEC-MDP Formalism

A translation of PEC domains into a MDP-derivative requires reconciliation of several key differences between the two frameworks: i. PEC dynamics operate on fluents (properties of the environment), while the MDP operates directly on environment states without internal structure; ii. PEC does not model rewards; iii. PEC assumes a progression of time independent to actions and environmental changes, while the MDP assumes a direct correspondence of each discrete time step to each episode of agent-state interaction; iv. PEC allows for simultaenous action-taking while the MDP does not; v. Action-taking in PEC conditions obligatorily on time-instants and optionally on specific partial fluent conditions, while the MDP's standard stationary policy which conditions only on the state an agent resides in.

Given that PEC's action-taking is conditioned on time, a non-stationary policy [6] is adopted to model time-dependent policies while maintaining stationary transition dynamics. The reward component of the MDP framework is omitted in the initial translation of PEC domains as PEC domains are without inherent reward signals. The PEC-MDP formalism is thus a reward-free MDP with a non-stationary policy.

To allow for efficient matrix operations, we translate PEC's natural language components into 0-based numerical encodings while maintaining bidirectional mappings to preserve interpretability. This encoding assigns each element an index based on arbitrary orderings over PEC fluents $\mathcal{F}$, values $\mathcal{V}$, and actions $\mathcal{U}$. The PEC-MDP state space is constructed through a two-step process: first, PEC fluent states are mapped to vector representations of fluent value indices (i.e. $(x_0, x_1, x_2, ...)$ where x_j denotes that the index of the value taken by the fluent of index j); next, these vectors are mapped to integers for a more compact representation, acting as the base unit of a PEC-MDP state. The vector representation allows access to specific fluents, enabling two crucial functions: i. the mapping of a partial fluent state $\tilde{X}$ to a set of PEC-MDP states in which fluents in $\tilde{X}$ are entailed; ii. the update of a PEC-MDP state with the effect of a partial fluent state $\tilde{X}$ by modifying fluent elements affected by $\tilde{X}$ but retaining those which are not.

Next, to accommodate PEC's more flexible action-taking mechanisms into the MDP's rigid framework, we introduce action-taking situations composing *single*, *composite*, and *null* actions to simulate time-points at which agents perform one action, multiple actions simultaneously, or do not take any actions, respectively. The set of action-taking situations is determined by referring to p-propositions, to find all possible action combinations that may be taken at one time, including the null action situation where no action is taken.

Time instants are normalised to begin at 0 while preserving temporal ordering. The initial distribution is retrieved from PEC's i-propositions, as a probability distribution over PEC-MDP states mapped from fluent states. Transition probabilities are derived from c-propositions, where the effects of an action-taking situation are aggregated from its composite actions. The fluent state update operation associates a current state to an updated state given some effect $\tilde{X}$ with its corresponding probability. Finally, a non-stationary policy captures PEC's time-conditioned action probabilities from p-propositions, representing distributions over action-taking situations rather than individual actions.

The PEC-MDP translation has been fully implemented in Python and may be found here. For a comprehensive overview of the functionalities, example domains, and usage instructions, readers are directed to the repository's README file.

4 Applications

The PEC-MDP enables applying MDP techniques to PEC domains, most notably reinforcement learning for optimal decision-making, while preserving PEC's original capability for temporal projection in narrative reasoning.

Previous implementations of temporal reasoning for PEC predominately calculate probabilities through summing over all possible worlds [3], or through approximate sampling [2]. Our proposed approach provides an exact solution through efficient matrix operations. We define policy-weighted transition matrices to propagate the initial distribution over PEC-MDP states forward to the queried time, to retrieve the distribution over states at that time. The probability that $\tilde{X}$ holds then is computed by summing probabilities over all fluent states that entail $\tilde{X}$. While a formal efficiency experiment has yet to be conducted, this method avoids the combinatorial explosion that comes with the computation of all possible worlds.

Next, to apply objective-directed strategies for the PEC-MDP, a desirability criterion must first be established in the form of an MDP reward function to guide agent behaviour. Once an appropriate quantitative reward signal is defined over outcomes, actions, or transitions, suitable reinforcement learning methods can be applied to discover optimal policies for the narrative domain. To preserve the interpretability advantages of PEC's original formalism, learned policies may be translated back into human-readable p-propositions. This requires deterministic policies since action-taking situations must be separated into individual actions for p-propositions, meaning that probabilistic dependencies between actions cannot be preserved.

The mapping of a policy over action-taking situations is trivial where each action in a situation is performed at the corresponding instant where the fluent state holds. However, as this generates a large number of p-propositions, refinements can be applied to reduce this set for interpretability while maintaining semantic equivalence. These include eliminating p-propositions for unreachable state-time combinations and generalising fluent state conditions to their minimal distinguishing features.

The complete formal translation of the PEC-MDP may be found at [15], alongside a more detailed outline of temporal reasoning and objective-directed strategies.

5 Conclusion

Finally, let us note that other probabilistic extensions of the Event Calculus have been proposed, focusing on event recognition using probabilistic logic programming and learning from noisy data in both offline and online settings, i.e., [14, 13, 8, 4, 7]. In contrast, our PEC-MDP framework inherits the main features of PEC, which is more expressive (see [2] for a comparison), compiling it into an MDP. This shift lays the groundwork for reinforcement learning, positioning our work toward goal-driven reasoning and policy optimisation.

While our current work focuses on the PEC-MDP's application to temporal projection and objective-directed learning, the PEC-MDP framework lays the groundwork for further applications that leverage MDP-based techniques within narrative domains. Beyond these broader applications, future work will include formal efficiency analyses and extending the framework to the Epistemic Probabilistic Event Calculus (EPEC) [2].

References

1 James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.

2 Fabio Aurelio D'Asaro, Antonis Bikakis, Luke Dickens, and Rob Miller. Probabilistic reasoning about epistemic action narratives. *Artificial Intelligence*, 287:103352, 2020. doi:10.1016/J.ARTINT.2020.103352.

3 Fabio Aurelio D'Asaro, Antonis Bikakis, Luke Dickens, and Rob Miller. Foundations for a probabilistic event calculus. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 57–63. Springer, 2017. doi:10.1007/978-3-319-61660-5_7.

4 Nikos Katzouris, Georgios Paliouras, and Alexander Artikis. Online learning probabilistic event calculus theories in answer set programming. *Theory and Practice of Logic Programming*, 23(2):362–386, 2023. doi:10.1017/S1471068421000107.

5 Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New generation computing*, 4:67–95, 1986. doi:10.1007/BF03037383.

6 Erwan Lecarpentier and Emmanuel Rachelson. Non-stationary markov decision processes, a worst-case approach using model-based reinforcement learning. *Advances in neural information processing systems*, 32, 2019. URL: https://dl.acm.org/doi/10.5555/3454287.3454935.

7 Periklis Mantenoglou. *Reasoning over Complex Temporal Specifications and Noisy Data Streams*. PhD thesis, Univeristy of Athens, 2024.

8 Periklis Mantenoglou, Alexander Artikis, and Georgios Paliouras. Online event recognition over noisy data streams. *International Journal of Approximate Reasoning*, 161:108993, 2023. doi:10.1016/J.IJAR.2023.108993.

9 John McCarthy. Programs with common sense, 1959.

10 John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and Donald Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

11 Rob Miller and Murray Shanahan. Some alternative formulations of the event calculus. In *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski Part II*, pages 452–490. Springer, 2002. doi:10.1007/3-540-45632-5_17.

12 Erick Sandewall. Features and fluents: A systematic approach to the representation of knowledge about dynamical systems. *Technical Report LiTH-IDA-R-92-30, Department of Computer and Information Science*, 1992.

13 Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, and Georgios Paliouras. A Probabilistic Logic Programming Event Calculus. *Theory and Practice of Logic Programming*, 15:213–245, March 2015. doi:10.1017/S1471068413000690.

14 Anastasios Skarlatidis, Georgios Paliouras, Alexander Artikis, and George A. Vouros. Probabilistic Event Calculus for Event Recognition. *ACM Transactions on Computational Logic (TOCL)*, 16(2):11, 2015. doi:10.1145/2699916.

15 Lyris Xu, Fabio Aurelio D'Asaro, and Luke Dickens. A translation of probabilistic event calculus into markov decision processes, 2025. arXiv:2507.12989.