28th International Symposium on Temporal Representation and Reasoning

TIME 2021, September 27–29, 2021, Klagenfurt, Austria (Virtual Conference)

Edited by

Carlo Combi

Johann Eder

Mark Reynolds

Editors

Carlo Combi
University of Verona, Italy
carlo.combi@univr.it

Johann Eder
University of Klagenfurt, Austria
johann.eder@aau.at

Mark Reynolds
University of Western Australia, Perth, Australia
mark.reynolds@uwa.edu.au

ACM Classification 2012
Theory of computation; Information systems; Computing methodologies → Artificial intelligence; Applied computing

ISBN 978-3-95977-206-8

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/978-3-95977-206-8.

Publication date
September, 2021

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

Digital Object Identifier: 10.4230/LIPIcs.TIME.2021.0

ISBN 978-3-95977-206-8 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

28th International Symposium on Temporal Representation and Reasoning (TIME 2021).
Editors: Carlo Combi, Johann Eder, and Mark Reynolds
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 28th International Symposium on Temporal Representation and Reasoning (TIME 2021) was planned to take place in Klagenfurt, Austria, but had to move to an online conference due to the insecurities and restrictions caused by the pandemic. Since its first edition in 1994, TIME Symposium is quite unique in the panorama of the scientific conferences as its main goal is to bring together researchers from distinct research areas involving the management and representation of temporal data as well as the reasoning about temporal aspects of information. Moreover, TIME Symposium aims to bridge theoretical and applied research, as well as to serve as an interdisciplinary forum for exchange among researchers from the areas of artificial intelligence, database management, logic and verification, and beyond.

Besides the three traditional tracks on

- Time in Artificial Intelligence
- Temporal Databases
- Temporal Logic and Reasoning

this year featured two additional special tracks on

- Temporal representation and reasoning for COVID-19
- Temporal explainability: connecting symbolic and sub-symbolic temporalities

Indeed, such a strange and long period of COVID-19 pandemic pushed for strong research efforts in some previously unexplored direction, namely the temporal issues in any context having to manage Covid-19 pandemic (healthcare, medicine, social contexts, school, and so on). On the other side, the need of explaining the results coming from sub-symbolic AI approaches is becoming more and more challenging and widespread. Thus, it is important to push for research dealing with some kind of temporal logics/system/rules/visualization for interpreting/explaining the results of machine learning algorithms considering temporally relevant problems.

The 2021 TIME edition, received a total of 28 paper submissions representing a wide range of research topics in the areas of artificial intelligence, databases, and theoretical computer science, some of them explicitly focusing on the topics of the two special tracks. Submissions came from Europe, North America, Africa, and Asia. We would like to thank all the authors of the submitted papers, as they have helped to build a successful TIME 2021 symposium.

As a result of the review process coordinated by the PC chairs, 13 papers were selected for full presentation at the symposium. The range of the considered topics is very wide – without trying to mention each specific topic, they run from temporal logics to temporal plans, to data models and query languages for new kinds of temporal data.

Most papers received 3 or more reviews, and each paper received at least two detailed reviews. All papers were discussed intensively by the program committee.

Besides having such a high number of high-quality reviews, the PC members and the program chairs have been involved in a deep additional discussion on many papers, to reach a final sound decision about rejection and acceptance.

We are pleased to include invited talks by leading scholars in our scientific communities: Torben Bach Pedersen (Aalborg University, Denmark), Johan Suykens (KU Leuven, Belgium), Roberto Posenato (University of Verona, Italy) and Luke Hunsberger (Vassar College, USA).

28th International Symposium on Temporal Representation and Reasoning (TIME 2021).
Editors: Carlo Combi, Johann Eder, and Mark Reynolds
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Finally, the program is completed by a panel coordinated by Alfredo Cuzzocrea, where experts in the field will share share their views and discuss research achievements and open challenges of Temporal Big Data Management.

We hope that the set of selected papers, their presentations, the invited talks, and the panel will help to stimulate and improve several research efforts in the area of temporal representation and reasoning.

The COVID-19 pandemic has created uncertainty and difficulties for people throughout the world. As we acknowledge the consequent concerns about health and safety, after having closely monitored the evolution of the pandemic, we decided to organize TIME 2021 completely on-line, as the evolution, still fast changing, unstable and different in different countries, did not allow a successful organization of an in-presence event.

We would like to thank here all the members of the Program Committee and the additional reviewers, who spent their time and volunteered their expertise to set up the final program. We want also to thank Marco Franceschetti, for his efforts in organizing a successful symposium.

Finally, we would like to acknowledge the generous support of the following institutions: Department of Informatics-Systems of the Alpen-Adria Universität Klagenfurt, Austria, and Department of Computer Science of the University of Verona, Italy. The open access publication of these proceedings was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

Even though an in-presence symposium would be the ideal way of sharing ideas, discussing and strengthening possible collaborations to advance our community, we are sure that the on-line event increased the outreach to make the current and next TIME editions even more attracting and with a long-lasting research impact.

Carlo Combi, University of Verona, Italy
Johan Eder, University of Klagenfurt, Austria
Mark Reynolds, University of Western Australia, Australia

TIME 2021 Program Co-chairs

TIME Steering Committee

28th International Symposium on Temporal Representation and Reasoning (TIME 2021).
Editors: Carlo Combi, Johann Eder, and Mark Reynolds
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

PC members

28th International Symposium on Temporal Representation and Reasoning (TIME 2021).
Editors: Carlo Combi, Johann Eder, and Mark Reynolds
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Tobias Schreck
Graz University of Technology
Austria
tobias.schreck@cgv.tugraz.at

Guido Sciavicco
Universitá di Ferrara
Italy
guido.sciavicco@unife.it

Francesca Zerbato
University of St. Gallen
Switzerland
francesca.zerbato@unisg.ch

List of Authors

M. A. Manazir Ahsan (14)
Department of Computer Science,
Utah State University, Logan, UT, USA

Gianluca Apriceno (11)
University of Trento, Italy;
Fondazione Bruno Kessler, Italy

Laura Bozzelli (9)
University of Napoli "Federico II", Italy

Florian Bruse (12)
School of Electrical Engineering and Computer Science, University of Kassel, Germany

Alexis Bédard (5)
Laboratoire d'informatique formelle,
Université du Québec à Chicoutimi,
Saguenay, Canada

Michael Cashmore (13, 15)
Strathclyde University, Glasgow, UK

Anasse Chafik (16)
CRIL, University of Artois & CNRS,
Arras, France

Fahima Cheikh-Alili (16)
CRIL, University of Artois & CNRS,
Arras, France

Alessandro Cimatti (13)
Fondazione Bruno Kessler, Trento, Italy

Jean-François Condotta (16)
CRIL, University of Artois & CNRS,
Arras, France

Alfredo Cuzzocrea (4)
iDEA Lab, University of Calabria, Rende, Italy;
LORIA, Nancy, France

Curtis E. Dyreson (14)
Department of Computer Science,
Utah State University, Logan, UT, USA

Luca Geatti (8)
University of Udine, Italy;
Fondazione Bruno Kessler, Trento, Italy

Nicola Gigante (8)
Free University of Bozen-Bolzano, Italy

Anastasios Gounaris (6)
Aristotle University of Thessaloniki, Greece

Sylvain Hallé (5)
Laboratoire d'informatique formelle,
Université du Québec à Chicoutimi,
Saguenay, Canada

Luke Hunsberger (1)
Department of Computer Science, Vassar College, Poughkeepsie, NY, USA

Andreas Kosmatopoulos (6)
Aristotle University of Thessaloniki, Greece

Martin Lange (12)
School of Electrical Engineering and Computer Science, University of Kassel, Germany

Oscar Lima (15)
DFKI German Research Center for Artificial Intelligence, Saabrücken, Germany

Daniele Magazzeni (13)
Kings College London, UK

Federico Manzella (7)
Dept. of Mathematics and Computer Science,
University of Ferrara, Italy

Andrea Micheli (13, 15)
Fondazione Bruno Kessler, Trento, Italy

Angelo Montanari (8, 9)
University of Udine, Italy

Giovanni Pagliarini (7)
Dept. of Mathematics and Computer Science,
University of Ferrara, Italy;
Dept. of Mathematical, Physical, and Computer Sciences, University of Parma, Italy

Andrea Passerini (11)
University of Trento, Italy

Torben Bach Pedersen (2)
Department of Computer Science,
Center for Data-intensive Systems,
Aalborg University, Denmark;
ModelarData, Copenhagen, Denmark

Adriano Peron (9)
University of Napoli "Federico II", Italy

Roberto Posenato (1)
Department of Computer Science,
University of Verona, Italy

Tomás Ribeiro (15)
Institute for Systems and Robotics,
Instituto Superior Tecnico, Lisbon, Portugal

28th International Symposium on Temporal Representation and Reasoning (TIME 2021).
Editors: Carlo Combi, Johann Eder, and Mark Reynolds
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sparsa Roychowdhury (17)
Indian Institute of Technology Bombay,
Mumbai, India

Vladislav Ryzhikov (10)
Department of Computer Science, Birkbeck,
University of London, UK

Pietro Sala (9)
University of Verona, Italy

Yury Savateev (10)
Department of Computer Science, Birkbeck,
University of London, UK;
HSE University, Moscow, Russia

Guido Sciavicco (7)
Dept. of Mathematics and Computer Science,
University of Ferrara, Italy

Luciano Serafini (11)
Fondazione Bruno Kessler, Italy

Alexandros Spitalas (6)
Aristotle University of Thessaloniki, Greece

Ionel Eduard Stan (7)
Dept. of Mathematics and Computer Science,
University of Ferrara, Italy;
Dept. of Mathematical, Physical, and Computer
Sciences, University of Parma, Italy

Johan Suykens (3)
ESAT-Stadius, KU Leuven, Belgium;
Leuven.AI Institute, Heverlee, Belgium

Kostas Tsichlas (6)
University of Patras, Greece

Ivan Varzinczak (16)
CRIL, University of Artois & CNRS,
Arras, France

Rodrigo Ventura (15)
Institute for Systems and Robotics,
Instituto Superior Tecnico, Lisbon, Portugal

Gabriele Venturato (8)
University of Udine, Italy
KU Leuven, Belgium

Michael Zakharyaschev (10)
Department of Computer Science,
Birkbeck, University of London, UK;
HSE University, Moscow, Russia

Parisa Zehtabi (13)
Kings College London, UK

Simple Temporal Networks: A Practical Foundation for Temporal Representation and Reasoning

Luke Hunsberger
Department of Computer Science, Vassar College, Poughkeepsie, NY, USA

Roberto Posenato
Department of Computer Science, University of Verona, Italy

Abstract

Since Simple Temporal Networks (STNs) were first introduced in 1991, there have been numerous theoretic and algorithmic advances that have made them practical for a wide variety of applications. However, the presentation of most of the important advances have been scattered across numerous conference papers and journal articles. As a result, it is too easy for even experienced researchers to be unaware of results that could positively impact their work. In this talk we review the most important results about STNs for researchers in Artificial Intelligence who are interested in incorporating the management of time and temporal constraints into their projects.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Theory of computation → Network optimization; Theory of computation → Dynamic graph algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Simple Temporal Networks, Consistency Checking, Restoring Consistency, Dispatchability, Temporal Decoupling Problem

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.1

Category Invited Talk

Funding *Luke Hunsberger*: This work was funded in part by the National Science Foundation (Grant Number 1909739).
Roberto Posenato: This work was partially supported by the Italian National Group for Scientific Computation (GNCS-INDAM) as part of project 2020 "Automated Reasoning about Time in Medical and Business Applications".

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Extended Abstract

Temporal networks are data structures for representing and reasoning about temporal constraints on activities. The most basic kind of temporal network is a Simple Temporal Network (STN) which can accommodate such constraints as release times, deadlines, precedence constraints, and duration constraints [16]. The fundamental computational tasks associated with STNs – checking consistency and managing execution – can be done in polynomial time [16, 40].

STNs are used as temporal reasoning tools in numerous research projects and applications. Currently, there are more than 2400 research papers in Google Scholar (more than 1200 in Scopus) that employ STNs, either as the main temporal reasoning model or as the basis for more expressive models. Considering only the most cited papers (more than 30 citations in Scopus) that present a tool or a methodology based on STNs, the use of STNs falls into the following two macro areas:

- *planning for robots* [30, 10, 25, 19, 1, 24, 27, 8, 7, 26, 31, 21],
- *industrial, business and health-care management systems* [17, 3, 2, 45, 37, 20, 5, 12, 11, 9]

An STN contains a set of real-valued variables, called *time-points,* that typically represent the starting or ending times of actions, together with a set of constraints on those time-points. Each constraint in an STN has the form, $Y - X \leq \delta$, where X and Y are time-points, and δ is a real number. Despite their limited form, the constraints in an STN can represent release times, deadlines, precedence constraints, and duration constraints. For example, if X and Y are the starting and ending times of an action, then $Y - X \leq 10$ represents that the duration of that action must be no more than 10. One advantage of STNs over earlier approaches is that they allow flexibility: time-points need not be assigned values in advance, but may be assigned in real time, during execution. In addition, STNs have an equivalent graphical form that enables the use of algorithms from the vast literature on labeled, directed graphs.

The *Simple Temporal Problem* (STP) is the problem of determining whether a given STN has a solution (i.e., is *consistent*, when viewed as a constraint satisfaction problem). Many polynomial algorithms have been presented for solving the STP, their performance differing depending on the structure of the STN graph. However, there are many other computational problems that users of STNs need to be able to solve. For example, in most applications, constraints are frequently inserted into the network incrementally, over time. The incremental STP checks the consistency as new constraints are inserted [36, 18, 34]. In addition, to take advantage of the flexibility offered by STNs, a system managing real-time execution needs to be able to quickly determine when to execute each time-point while preserving the network's consistency. And, in cases where spurious events introduce an inconsistency into the network, a system must be able to figure out modifications that will restore a network's consistency [14, 33]. For another example, in multi-agent scenarios with limited communication, temporal networks may need to be *decoupled* to enable agents to operate independently [22, 35, 4, 44]. All of these, and many other problems associated with STNs have polynomial algorithms for solving them, making STNs a practical temporal reasoning model for many real-world applications.

Because many real-world scenarios involve features that are not representable in STNs, the STN model has been extended in many different ways. For example, an STNU accommodates actions with *uncertain* durations (e.g., a taxi ride) [42, 43, 28], a CSTN accommodates *conditional* constraints and test actions that generate information in real time [41, 13, 23], and a DTN accommodates *disjunctive* constraints (e.g., action A must finish before action B starts *or* vice-versa) [38, 39, 32]. Other extensions of STNs accommodate probabilistic constraints [29, 15] and decisions (or choice points) [6].

Typically, the increase in expressiveness introduces a significant computational cost. (The main exception is the STNU, for which many polynomial algorithms exist.) However, recent advances have begun to make these extensions viable for practical applications. In each case, the techniques applied to the more expressive networks builds on the theory and algorithms for STNs.

This talk surveys the 30-year development of the theory and algorithms for Simple Temporal Networks for researchers and application developers interested in incorporating STNs into their projects.

References

1 Mitchell Ai-Chang, John Bresina, Len Charest, Adam Chase, Jennifer Cheng Jung Hsu, Ari Jhonson, Bob Kanefsky, Paul Morris, Kanna Rajan, Jeffrey Yglesias, Brian G. Chafin, William C. Dias, and Pierre F. Maldague. Mapgeri: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004. doi:10.1109/MIS.2004.1265878.

2 Claudio Bettini, Sushil Jajodia, X. Sean Wang, and Duminda Wijesekera. Provisions and Obligations in Policy Rule Management. *J. Netw. Syst. Manag.*, 11(3):351–372, 2003. doi:10.1023/A:1025711105609.

3 Claudio Bettini, Xiaoyang Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Dist. & Paral. Data.*, 11(3):269–306, 2002. doi:10.1023/A:1014048800604.

4 James C. Boerkoel and Edmund H. Durfee. Distributed reasoning for multiagent simple temporal problems. *J. of Artificial Intelligence Research*, 47:95–156, 2013. doi:10.1613/jair.3840.

5 Vittorio Brusoni, Luca Console, Paolo Terenziani, and Barbara Pernici. Later: Managing temporal information efficiently. *IEEE Expert. Syst. their Appl.*, 12(4):56–63, 1997. doi:10.1109/64.608197.

6 Massimo Cairo, Carlo Combi, Carlo Comin, Luke Hunsberger, Roberto Posenato, Romeo Rizzi, and Matteo Zavatteri. Incorporating Decision Nodes into Conditional Simple Temporal Networks. In *24th Int. Symp. on Temporal Representation and Reasoning (TIME-2017)*, 2017.

7 Filippo Cavallo, Raffaele Limosani, Alessandro Manzi, Manuele Bonaccorsi, Raffaele Esposito, Maurizio Di Rocco, Federico Pecora, Giancarlo Teti, Alessandro Saffiotti, and Paolo Dario. Development of a Socially Believable Multi-Robot Solution from Town to Home. *Cognit. Comput.*, 6(4):954–967, 2014. doi:10.1007/s12559-014-9290-z.

8 Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, and Angelo Oddi. Developing an end-to-end planning application from a Timeline Representation Framework. In *Proc. 21st Innov. Appl. Artif. Intell. Conf. IAAI-09*, pages 66–71, 2009.

9 Amedeo Cesta, Gabriella Cortellessa, Riccardo Rasconi, Federico Pecora, Massimiliano Scopelliti, and Lorenza Tiberio. Monitoring elderly people with the ROBOCARE domestic environment: Interaction synthesis and user evaluation. In *Comput. Intell.*, volume 27, pages 60–82, 2011. doi:10.1111/j.1467-8640.2010.00372.x.

10 Steve Chien, Benjamin Smith, Gregg Rabideau, Nicola Muscettola, and Kanna Rajan. Automated planning and scheduling for goal-based autonomous spacecraft. *IEEE Intell. Syst. Their Appl.*, 13(5):50–55, 1998. doi:10.1109/5254.722362.

11 Carlo Combi, Matteo Gozzi, Barbara Oliboni, Jose M. Juarez, and Roque Marin. Temporal similarity measures for querying clinical workflows. *Artif. Intell. Med.*, 46(1):37–54, 2009. doi:10.1016/j.artmed.2008.07.013.

12 Carlo Combi and Roberto Posenato. Controllability in temporal conceptual workflow schemata. In *BPM*, pages 64–79, 2009. doi:10.1007/978-3-642-03848-8_6.

13 Carlo Comin and Romeo Rizzi. Dynamic consistency of conditional simple temporal networks via mean payoff games: a singly-exponential time dc-checking. In *22st Int. Symp. on Temporal Representation and Reasoning (TIME 2015)*, pages 19–28, 2015. doi:10.1109/TIME.2015.18.

14 Ali Dasdan. Provably efficient algorithms for resolving temporal and spatial difference constraint violations. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):8:1–8:24, 2009. doi:10.1145/1455229.1455237.

15 Pedro Henrique de Rodrigues Quemel e Assis Santana, Tiago Vaquero, Cláudio Toledo, Andrew J. Wang, Cheng Fang, and Brian Charles Williams. PARIS: A Polynomial-Time, Risk-Sensitive Scheduling Algorithm for Probabilistic Simple Temporal Networks with Uncertainty. In *26th Int. Conf. on Automated Planning and Scheduling (ICAPS-2016)*, pages 267–275, 2016.

16 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.

17 Georg Duftschmid, Silvia Miksch, and Walter Gall. Verification of temporal scheduling constraints in clinical practice guidelines. *Artif. Intell. Med.*, 25(2):93–121, 2002. doi:10.1016/S0933-3657(02)00011-8.

18 A. Gerevini, A. Perini, and F. Ricci. Incremental algorithms for managing temporal constraints. In *8th IEEE Int. Conf. on Tools with Artificial Intelligence*, pages 360–363, 1996. doi:10.1109/TAI.1996.560477.

19 Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004. doi:10.1016/B978-1-55860-856-6.X5000-5.

20 Fredrik Heintz, Piotr Rudol, and Patrick Doherty. From images to traffic behavior - A UAV tracking and monitoring application. In *FUSION 2007 - 2007 10th Int. Conf. Inf. Fusion*, 2007. doi:10.1109/ICIF.2007.4408103.

21 Wolfgang Honig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Proc. Int. Conf. Autom. Plan. Sched. ICAPS*, pages 477–485. AAAI press, 2016.

22 Luke Hunsberger. Algorithms for a temporal decoupling problem in multi-agent planning. In *8th Nat. Conf. on Artificial Intelligence (AAAI-2002)*, 2002.

23 Luke Hunsberger, Roberto Posenato, and Carlo Combi. A Sound-and-Complete Propagation-based Algorithm for Checking the Dynamic Consistency of Conditional Simple Temporal Networks. In Fabio Grandi, Martin Lange, and Alessio Lomuscio, editors, *22nd Int. Symp. on Temporal Representation and Reasoning (TIME-2015)*, pages 4–18, 2015. doi:10.1109/TIME.2015.26.

24 Félix Ingrand, Solange Lemai-Chenevier, and Frederic Py. Decisional autonomy of planetary rovers. *J. F. Robot.*, 24(7):559–580, 2007. doi:10.1002/rob.20206.

25 Phil Kim, Brian C. Williams, and Mark Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI Int. Jt. Conf. Artif. Intell.*, pages 487–493, 2001.

26 Steven J. Levine and Brian C. Williams. Concurrent plan recognition and execution for human-robot teams. In *Proc. Int. Conf. Autom. Plan. Sched. ICAPS*, pages 490–498. AAAI press, 2014.

27 Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. A deliberative architecture for AUV control. In *Proc. - IEEE Int. Conf. Robot. Autom.*, pages 1049–1054, 2008. doi:10.1109/ROBOT.2008.4543343.

28 Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001)*, pages 494–502, 2001.

29 R. Morris, P. Morris, L. Khatib, and N. Yorke-Smith. Temporal constraint reasoning with preferences and probabilities. In *Multidisciplinary Workshop on Advances in Preference Handling in IJCAI-2005*, pages 150–155, 2005.

30 N. Muscettola, P.P. Nayak, B. Pell, and B.C. Williams. Remote Agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998. doi:10.1016/s0004-3702(98)00068-x.

31 Ernesto Nunes and Maria Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *Proc. Natl. Conf. Artif. Intell.*, volume 3, pages 2110–2116. AI Access Foundation, 2015.

32 A. Oddi and A. Cesta. Incremental forward checking for the disjunctive temporal problem. In *European Conf. on Artificial Intelligence (ECAI-2000)*, 2000.

33 Lian Ien Oei. *Resolving Disruptions in Simple Temporal Problems*. mathesis, Delft University of Technology, 2009.

34 Léon Planken, Mathijs de Weerdt, and Neil Yorke-Smith. Incrementally Solving STNs by Enforcing Partial Path Consistency. In *20th Int. Conf. on Automated Planning and Scheduling (ICAPS-2010)*, pages 129–136, 2010. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13417.

35 Léon R. Planken, Mathijs M. de Weerdt, and Cees Witteveen. Optimal temporal decoupling in multiagent systems. In *9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, volume 1, pages 789–796, 2010.

36 G. Ramalingam, J. Song, L. Joskowicz, and R. E. Miller. Solving Systems of Difference Constraints Incrementally. *Algorithmica*, 23(3):261–275, 1999. doi:10.1007/PL00009261.

37 Wheeler Ruml, Minh B. Do, and Markus P.J. Fromherz. On-line planning and scheduling for high-speed manufacturing. In *ICAPS 2005 - Proc. 15th Int. Conf. Autom. Plan. Sched.*, pages 30–39, 2005.

38 E Schwalb and Rina Dechter. Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, 93(1-2):29–61, 1997. doi:10.1016/S0004-3702(97)00009-X.

39 Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artif. Intell.*, 120(1):81–117, 2000. doi:10.1016/S0004-3702(00)00019-9.

40 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.

41 Ioannis Tsamardinos, Thierry Vidal, and Martha E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8:365–388, 2003. doi:10.1023/A:1025894003623.

42 Thierry Vidal and Hélène Fargier. Contingent durations in temporal csps: from consistency to controllabilities. In *4th Int. Workshop on Temporal Representation and Reasoning (TIME-1997)*, pages 78–85, 1997. doi:10.1109/TIME.1997.600786.

43 Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999. doi:10.1080/095281399146607.

44 Michel Wilson, Tomas Klos, Cees Witteveen, and Bob Huisman. Flexibility and decoupling in simple temporal networks. *Artificial Intelligence*, 214:26–44, 2014. doi:10.1016/j.artint.2014.05.003.

45 Hyun Joong Yoon and Doo Yong Lee. Online scheduling of integrated single-wafer processing tools with temporal constraints. *IEEE Trans. Semicond. Manuf.*, 18(3):390–398, 2005. doi:10.1109/TSM.2005.852103.

Extreme-Scale Model-Based Time Series Management with ModelarDB

Torben Bach Pedersen
Department of Computer Science, Center for Data-intensive Systems, Aalborg University, Denmark
ModelarData, Copenhagen, Denmark

Abstract

To monitor critical industrial devices such as wind turbines, high quality sensors sampled at a high frequency are increasingly used. Current technology does not handle these extreme-scale time series well [1], so only simple aggregates are traditionally stored, removing outliers and fluctuations that could indicate problems. As a remedy, we present a model-based approach for managing extreme-scale time series that approximates the time series values using mathematical functions (models) and stores only model coefficients rather than data values. Compression is done both for individual time series and for correlated groups of time series. The keynote will present concepts, techniques, and algorithms from model-based time series management and our implementation of these in the open source Time Series Management System (TSMS) ModelarDB[2, 3, 4] [1]. Furthermore, it will present our experimental evaluation of ModelarDB on extreme-scale real-world time series, which shows that that compared to widely used Big Data formats, ModelarDB provides up to 14× faster ingestion due to high compression, 113× better compression due to its adaptability, 573× faster aggregatation by using models, and close to linear scale-out scalability. ModelarDB is being commercialized by the spin-out company ModelarData[2].

2012 ACM Subject Classification Information systems → Data management systems

Keywords and phrases Model-based storage, approximate query processing, time series management, extreme-scale data

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.2

Category Invited Talk

Funding *Torben Bach Pedersen*: This work has been funded by Innovation Fund Denmark (the DiCyPS project), Independent Research Fund Denmark (the SEMIOTIC project) and Horizon 2020 (the MORE project).

Acknowledgements I want to thank Søren Kejser Jensen and Christian Thomsen for their major contributions to this work.

References

1 Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Time series management systems: A survey. *IEEE Trans. Knowl. Data Eng.*, 29(11):2581–2600, 2017. doi:10.1109/TKDE.2017.2740932.

2 Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Modelardb: Modular model-based time series management with spark and cassandra. *Proc. VLDB Endow.*, 11(11):1688–1701, 2018. doi:10.14778/3236187.3236215.

3 Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Demonstration of modelardb: Model-based management of dimensional time series. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 – July 5, 2019*, pages 1933–1936. ACM, 2019. doi:10.1145/3299869.3320216.

[1] https://github.com/skejserjensen/ModelarDB
[2] https://modelardata.com

4 Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Scalable model-based management of correlated dimensional time series in modelardb+. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19–22, 2021*, pages 1380–1391. IEEE, 2021. doi:10.1109/ICDE51399.2021.00123.

Kernel Machines in Time

Johan Suykens
ESAT-Stadius, KU Leuven, Belgium
Leuven.AI Institute, Heverlee, Belgium

Abstract

Kernel machines is a powerful class of models in machine learning with solid foundations and many existing application fields. The scope of this talk is kernel machines in time with a main focus on least squares support vector machines, and other related methods such as kernel principal component analysis and kernel spectral clustering. For dynamical systems modelling different possible input-output and state space model structures will be discussed. Applications will be shown on electricity load forecasting and temperature prediction in weather forecasting. Approximate closed-form solutions can be given to ordinary and partial differential equations. Kernel spectral clustering applications to identifying customer profiles, pollution modelling and detecting topological changes in time-series of bridges will be shown. Finally, new synergies between kernel machines and deep learning will be presented, leading for example to generative kernel machines, with new insights on disentangled representations, explainability and latent space exploration. Application of these models will be illustrated on out-of-distribution detection of time-series data.

2012 ACM Subject Classification Computing methodologies → Support vector machines

Keywords and phrases SVM, Time series analysis

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.3

Category Invited Talk

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

Temporal Big Data Analytics: New Frontiers for Big Data Analytics Research

Alfredo Cuzzocrea[1]
iDEA Lab, University of Calabria, Rende, Italy
LORIA, Nancy, France

Abstract

Big data analytics is an emerging research area with many sophisticated contributions in the actual literature. Big data analytics aims at discovering actionable knowledge from large amounts of big data repositories, based on several approaches that integrate foundations of a wide spectrum of disciplines, ranging from data mining to machine learning and artificial intelligence. Among the concrete innovative topics of big data analytics, *temporal big data analytics* covers a first-class role and it is attracting the attention of larger and larger communities of academic and industrial researchers. Basically, temporal big data analytics aims at modeling, capturing and analyzing temporal aspects of big data during analytics phase, including specialized tasks such as *big data versioning over time*, building temporal relations among ad-hoc *big data structures* (such as nodes of *big graphs*) and *temporal queries over big data*. It is worth to notice that temporal big data analytics research is characterized by several open challenges, which range from foundations, including temporal big data representation and processing, to applications, including smart cities and bio-informatics tools. Inspired by these considerations, this paper focuses on models, paradigms, techniques and future challenges of temporal big data analytics, by reporting on state-of-the-art results as well as emerging trends, with also criticisms on future work that we should expect from the community.

2012 ACM Subject Classification Information systems → Temporal data; Information systems → Data analytics

Keywords and phrases Big Data Analytics, Big Data Management, Temporal Big Data Analytics, Temporal Big Data Management

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.4

Category Panel Description

Funding This research has been partially supported by the French PIA project "Lorraine Université d'Excellence", reference ANR-15-IDEX-04-LUE.

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Big data analytics is an emerging research area with many sophisticated contributions in the actual literature (e.g., [28, 18, 7, 21]). Big data analytics aims at discovering actionable knowledge from large amounts of big data repositories, based on several approaches that integrate foundations of a wide spectrum of disciplines, ranging from data mining to machine learning and artificial intelligence. Among the concrete innovative topics of big data analytics, *temporal big data analytics* covers a first-class role and it is attracting the attention of larger and larger communities of academic and industrial researchers (e.g., [9, 26, 10, 24, 32]). Basically, temporal big data analytics aims at modeling, capturing and analyzing temporal

[1] This research has been made in the context of the Excellence Chair in Computer Engineering – Big Data Management and Analytics at LORIA, Nancy, France

aspects of big data during analytics phase, including specialized tasks such as *big data versioning over time*, building temporal relations among ad-hoc *big data structures* (such as nodes of *big graphs*) and *temporal queries over big data*. It is worth to notice that temporal big data analytics research is characterized by several open challenges, which range from foundations, including temporal big data representation and processing, to applications, including smart cities and bio-informatics tools.

Temporal big data analytics arise in many application scenarios. Consider, for instance, the case of social networks. Here, the user content (such as posts, pictures, videos, etc.) identify a very large big data repository, while the relationships among users and contents, users and users, etc., across time, define an *evolving* temporal big data set. Basically, for each reference timestamp, namely $t_i, t_{i+1}, t_{i+2}, \ldots$, we have a different big data set, namely $\mathcal{B} = \{B(t_i), B(t_{i+1}), B(t_{i+2}), \ldots\}$. How to make analytics *across* the various big data sets $B(t_i), B(t_{i+1}), B(t_{i+2}), \ldots$ in $\mathcal{B}$? For instance, what was the most frequent itemset pattern of posts of the user Bob at timestamp t_j and what the one at timestamp t_{j+k}? Furthermore, what is the intersection set of Facebook common friends of Bob and Alice at timestamp t_j and what the one at timestamp t_{j+k}? How common friends ave evolved (e.g., removing old friends or adding new friends) across time? The latter one are simplest temporal big data analytics patterns, which can be further developed towards more complex ones, for instance based on *multidimensional analytics* (e.g., [19, 8, 11]).

In all the described settings, *temporal big queries* play a critical role. How to query "historical" big data repositories? In order to to this, first a suitable representation model for capturing the temporal aspect of big data must be introduced. The most popular approach to this end is presented by straightforward application of classical models for temporal databases, but targeted to the specific big data environment, thus, considering, for instance, *scalability issues* (e.g., [22, 2, 20]). Here, big data objects are associated with a proper timestamp, according to three different schema, namely *valid time*, *transaction time*, and *decision time*. According to the first schema, given a big data object O_k, the reference timestamp associated to O_k, namely t_k, is the time period during which a fact is true in the real world. According to the second schema, t_k is the time at which a fact was recorded in the database. Finally, according to the third schema, t_k is the time at which the decision was made about the fact. Based on the given temporal data model, given a big data repository B, a temporal big query over B referring to timestamp t_k, denoted by $Q(B)_{t_k}$, retrieves from B all the big data objects in B associated to the timestamp t_k and that satisfy the query predicates in Q (e.g., aggregation predicates – [30, 15]). Formally, a temporal big data analytics $\mathcal{A}$ over B, is defined as a collection of analytical functions $\mathcal{A} = \{f_0, f_1, f_2, \ldots, f_{n-1}\}$ such that every analytical function f_j is defined on top of a collection of a set of temporal big queries $\mathcal{Q} = \{Q(B)_{t_k}, Q(B)_{t_{k+1}}, Q(B)_{t_{k+2}}, \ldots Q(B)_{t_{k+m-1}}\}$. f_j can be of different nature, for instance based on *OLAP analytics* (e.g., [6, 16, 13]).

The above described model based on temporal big queries can be easily extended to more complex procedures, including, for instance, *data mining programs* and *machine learning procedures*, still parameterized by timestamp. In this vest, the integration of the latter temporal big data analytics model with emerging *NoSQL databases* (e.g., [29]) turns to be very interesting.

Inspired by these considerations, this paper focuses on models, paradigms, techniques and future challenges of temporal big data analytics, by reporting on state-of-the-art results as well as emerging trends, with also criticisms on future work that we should expect from the community. The remaining part of this paper is organized as follows. In Section 2, we provide a brief overview of most relevant state-of-the-art proposal for temporal big data

analytics appearing in literature. Section 3 is devoted to the description of most relevant emerging challenges and open issues for temporal big data analytics. Finally, in Section 4, we derive conclusions of our research.

2 Temporal Big Data Analytics: State-Of-The-Art Proposals

Inspired by considerations reported above, temporal big data analytics is a rich area of research. As a consequence, there exist in literature a number of relevant proposals, with many interesting scientific as well as industrial outcomes. Here, we outline some of the most relevant ones.

[9] focuses on the issue of supporting *temporal analytics on big data for web advertising*. For instance, they consider display advertising that makes use of Behavioral Targeting (BT) to select ads for users based on prior searches, page views, etc. Previous work on BT has focused on techniques that scale well for offline data using Map-Reduce (M-R). However, this approach has limitations for BT-style applications that deal with temporal data: (1) many queries are temporal and not easily expressible in M-R, and moreover, the set-oriented nature of M-R frontends such as SCOPE is not suitable for temporal processing; (2) as commercial systems mature, they may need to also directly analyze and react to real-time data feeds since a high turnaround time can result in missed opportunities, but it is difficult for current solutions to naturally also operate over real-time streams. The contributions of the paper are twofold. First, authors propose a novel framework called *TiMR*, that combines a time-oriented data processing system with a M-R framework. Users perform analytics using temporal queries – these queries are succinct, scale-out-agnostic, and easy to write. They scale well on large-scale offline data using TiMR, and can work unmodified over real-time streams. They also propose new cost-based query fragmentation and temporal partitioning schemes for improving efficiency with TiMR. Second, they show the feasibility of this approach for BT, with new temporal algorithms that exploit new targeting opportunities. Experiments using real advertising data show that TiMR is efficient and incurs orders-of-magnitude lower development effort. The proposed BT solution is easy and succinct, and performs up to several times better than current schemes in terms of memory, learning time, and click-through-rate/coverage.

[26] considers, instead, *temporal event tracing on big healthcare data analytics*. The study presents a comprehensive method for rapidly processing, storing, retrieving, and analyzing big healthcare data. Based on NoSQL, a patient-driven data architecture is suggested to enable the rapid storing and flexible expansion of data. Thus, the schema differences of various hospitals can be overcome, and the flexibility for field alterations and addition is ensured. The timeline mode can easily be used to generate a visual representation of patient records, providing physicians with a reference for patient consultation. The sharding-key is used for data partitioning to generate data on patients of various populations. Subsequently, data reformulation is conducted as a first step, producing additional temporal and spatial data, providing cloud computing methods based on query-MapReduce-shard, and enhancing the search performance of data mining. Target data can be rapidly searched and filtered, particularly when analyzing temporal events and interactive effects.

[10] deals with the challenge of defining and implementing *temporal data analytics on COVID-19 data with ubiquitous computing*. Authors argue that, with technological advancements in computing and communications, huge amounts of big data are generated and collected at a very rapid rate from a wide variety of rich data sources. Embedded in these big data are useful information and valuable knowledge. An example is healthcare and

epidemiological data such as data related to patients who suffered from viral diseases like the coronavirus disease 2019 (COVID-19). Knowledge discovered from these epidemiological data via data science helps researchers, epidemiologists and policy makers to get a better understanding of the disease, which may inspire them to come up ways to detect, control and combat the disease. In the paper, authors present a temporal data science algorithm for analyzing big COVID-19 epidemiological data, with focus on the temporal data analytics with ubiquitous computing. The algorithm helps users to get a better understanding of information about the confirmed cases of COVID-19. Evaluation results show the benefits of the proposed system in temporal data analytics of big COVID-19 data with ubiquitous computing. Although the algorithm is designed for temporal data analytics of big epidemiological data, it would be applicable to other temporal data analytics of big data in many real-life applications and services.

[24] moves the attention on *large-scale smart grids, with specific temporal, functional and spatial big data computing features*. Indeed, with the deployment of monitoring devices, the smart grid is collecting large amounts of energy-related data at an unprecedented speed. The smart grid has become data-driven, which necessitates extracting meaningful data from a large dataset. The traditional approach of data extraction improves the computing efficiency in temporal dimension, but it is made for only one task in the smart grid. Moreover, the existing solutions neglect the geographical distribution of computing capacity in a large-scale smart grid. The future large-scale smart grid will run over the internet of energy where the dataset will be sent to a specific destination along power routers hop-by-hop. Consequently, authors design a novel temporal, functional and spatial big data computing framework for large-scale smart grid. In functional dimension, they divide every dataset into sub-groups, each of which has data items shared by different tasks. In spatial dimension, they determine which location the power router should be placed to harvest computing resources used for extracting the sub-group of data items. The proposed method achieves a promising computing efficiency approaching to the optimal solution with 95 percent convergence ratio, and it saves the in-path bandwidth with 81 percent improvement ratio over benchmarks.

Finally, [32] considers the specific application scenario represented by *scientific big data analytics* and, in particular, *text-based temporally linked event networks* in such a scenario. Authors recognize that events formulate the world of the human being and could be regarded as the semantic units in different granularities for information organization. Extracting events and temporal information from texts plays an important role for information analytics in big data because of the wide use of multilingual texts. Based on these considerations, the paper surveys existing research work on text-based event temporal resolution and reasoning including identification of events, temporal information resolutions of events in English and Chinese texts, the rule-based temporal relation reasoning between events and relevant temporal representations. For the scientific big data analytics, authors point out the shortcomings of existing research work and give the argument about the future research work for advancing identification of events, establishment of temporal relations and reasoning of temporal relations.

3 Temporal Big Data Analytics: Emerging Challenges and Open Issues

Temporal big data analytics opens several new and challenging research perspectives. In the next, we focus the attention on those that have been scored as relevant by our study.

Big Temporal Data Representation. As mentioned in Section 1, the first issue to face-off is represented by how to model and capture big temporal data for supporting temporal big data analytics effectively and efficiently. This involves in devising suitable *temporal big data models*, which must also consider scalability issue of big temporal data processing (e.g., [5]). On the other hand, another relevant aspect of big temporal data to consider is their clear *multi-granularity nature*, according to which the same (big) data appear in different scales and resolutions (e.g., year, quarter, month, and so forth – [4]). How to represent this special feature effectively and efficiently? Cloud computing environments, with well-known *elastic metaphors* (e.g., [1]), seem to be the most promising computational solutions to be considered by future research efforts.

Big Queries on Big Temporal Data Repositories. Big queries on big temporal data repositories is an exciting new challenge that is critical for temporal big data analytics, like in some related cases (e.g., [31]). Basically, considering that big temporal data repositories are characterized by a *strong heterogeneity*, big queries appear in several formats (e.g., tree-like queries, graph-like queries, and so forth) and, as a consequence, the big query optimization and evaluation layer of a hypothetical temporal big data analytics engine should include several characteristics and functionalities, such as query translation and query rewriting (e.g., [23]). In this respect, the implementation on top of MapReduce framework seems a promising direction to follow.

Machine-Learning-Based Temporal Big Data Analytics. Temporal big data analytics are usually based on a *core methodology* that characterizes their analytical functions (see Section 1). Among several alternatives, *machine-learning-based temporal big data analytics* should be considered as the most sophisticated collection of techniques available in literature (e.g., [27]). Indeed, machine learning techniques are flexible enough to deal with big temporal data, and to support the relevant knowledge discover process from such big data repositories effectively and efficiently. A wide family of proposals are available in actual literature, and they can be applied to the issue of supporting temporal big data analytics in real-life application scenarios (e.g., smart cities, sensor networks, IoT systems, and so forth).

Uncertainty and Imprecision in Temporal Big Data Analytics. Big temporal data are naturally affected by *uncertainty and imprecision* (e.g., [17]), due to several reasons, among which data transmission errors and human data entry errors are just two possible instances. How uncertainty and imprecision of big temporal data impact on the accuracy and the global-quality of temporal big data analytics procedures built on top of them? This is a critical question that future research efforts must consider seriously. To this end, *probabilistic temporal big data analytics models* (e.g., [12]) seem to be a solid paradigm to be considered.

Privacy-Preserving Temporal Big Data Analytics. Temporal big data analytics usually access *sensitive data*. To be convinced of this, consider the case of temporal big data analytics developed in the contest of federated healthcare systems. Here, a massive amount of *personal data* (e.g., [3]) are accessed and processed in order to derive suitable analytics for decision making. How to design and devise *privacy-preserving temporal big data analytics* models capable of preserving the privacy of sensitive (e.g., personal) data while not lowering the degree of accuracy and decision-making-support of the target temporal big data analytics? The latter question will be an annoying challenge for many years to come (e.g., [14]).

4 Conclusions

In this paper, we provided a comprehensive overview of state-of-the-art temporal big data analytics techniques and algorithms, by highlighting their benefits and limitations. As a further contribution of our work, we have also provided a discussion on open research challenges and future directions in this scientific field, aiming at achieving a significant milestone to be exploited by forthcoming research efforts. Last but not least, we firmly believe that innovative and emerging application scenarios (e.g., [25]) will provide more insights and inspiration to the research community for the future years.

References

1 Divyakant Agrawal, Amr El Abbadi, Sudipto Das, and Aaron J Elmore. Database scalability, elasticity, and autonomy in the cloud. In *International Conference on Database Systems for Advanced Applications*, pages 2–15. Springer, 2011.

2 Sara Alghunaim and Heyam H Al-Baity. On the scalability of machine-learning algorithms for breast cancer prediction in big data context. *IEEE Access*, 7:91535–91546, 2019.

3 Rasim M Alguliyev, Ramiz M Aliguliyev, and Fargana J Abdullayeva. Privacy-preserving deep learning algorithm for big personal data analysis. *Journal of Industrial Information Integration*, 15:1–14, 2019.

4 Safaa Alwajidi and Li Yang. Multi-resolution hierarchical structure for efficient data aggregation and mining of big data. In *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*, pages 153–159. IEEE, 2019.

5 Ladjel Bellatreche, Alfredo Cuzzocrea, and Soumia Benkrid. $\mathcal{F}\&\mathcal{A}$: A methodology for effectively and efficiently designing parallel relational data warehouses on heterogenous database clusters. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 89–104. Springer, 2010.

6 Boualem Benatallah, Hamid Reza Motahari-Nezhad, et al. Scalable graph-based olap analytics over process execution data. *Distributed and Parallel Databases*, 34(3):379–423, 2016.

7 Alina Campan, Alfredo Cuzzocrea, and Traian Marius Truta. Fighting fake news spread in online social networks: Actual trends and future research directions. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4453–4457. IEEE, 2017.

8 Michelangelo Ceci, Alfredo Cuzzocrea, and Donato Malerba. Effectively and efficiently supporting roll-up and drill-down olap operations over continuous dimensions via hierarchical clustering. *Journal of Intelligent Information Systems*, 44(3):309–333, 2015.

9 Badrish Chandramouli, Jonathan Goldstein, and Songyun Duan. Temporal analytics on big data for web advertising. In *2012 IEEE 28th international conference on data engineering*, pages 90–101. IEEE, 2012.

10 Yubo Chen, Carson K Leung, Siyuan Shang, and Qi Wen. Temporal data analytics on covid-19 data with ubiquitous computing. In *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 958–965. IEEE, 2020.

11 Alfredo Cuzzocrea. Accuracy control in compressed multidimensional data cubes for quality of answer-based olap tools. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*, pages 301–310. IEEE, 2006.

12 Alfredo Cuzzocrea. Retrieving accurate estimates to olap queries over uncertain and imprecise multidimensional data streams. In *International Conference on Scientific and Statistical Database Management*, pages 575–576. Springer, 2011.

13 Alfredo Cuzzocrea. Analytics over big data: Exploring the convergence of datawarehousing, olap and data-intensive cloud infrastructures. In *2013 IEEE 37th Annual Computer Software and Applications Conference*, pages 481–483. IEEE, 2013.

14 Alfredo Cuzzocrea. Privacy and security of big data: current challenges and future research perspectives. In *Proceedings of the first international workshop on privacy and secuirty of big data*, pages 45–47, 2014.

15 Alfredo Cuzzocrea. Aggregation and multidimensional analysis of big data for large-scale scientific applications: models, issues, analytics, and beyond. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, pages 1–6, 2015.

16 Alfredo Cuzzocrea, Carmen De Maio, Giuseppe Fenza, Vincenzo Loia, and Mimmo Parente. Olap analysis of multidimensional tweet streams for supporting advanced analytics. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 992–999, 2016.

17 Alfredo Cuzzocrea, Carson Kai-Sang Leung, and Richard Kyle MacKinnon. Mining constrained frequent itemsets from distributed uncertain data. *Future Generation Computer Systems*, 37:117–126, 2014.

18 Alfredo Cuzzocrea and Il-Yeol Song. Big graph analytics: the state of the art and future research agenda. In *Proceedings of the 17th International Workshop on Data Warehousing and OLAP*, pages 99–101, 2014.

19 Alfredo Cuzzocrea, Il-Yeol Song, and Karen C Davis. Analytics over large-scale multidimensional data: the big data revolution! In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*, pages 101–104, 2011.

20 Alfredo Cuzzocrea and Wei Wang. Approximate range–sum query answering on data cubes with probabilistic guarantees. *Journal of Intelligent Information Systems*, 28(2):161–197, 2007.

21 Timothée Dubuc, Frederic Stahl, and Etienne B Roesch. Mapping the big data landscape: technologies, platforms and paradigms for real-time analytics of data streams. *IEEE Access*, 9:15351–15374, 2020.

22 Diego García-Gil, Sergio Ramírez-Gallego, Salvador García, and Francisco Herrera. A comparison on scalability for batch big data processing on apache spark and apache flink. *Big Data Analytics*, 2(1):1–11, 2017.

23 Rihan Hai, Christoph Quix, and Chen Zhou. Query rewriting for heterogeneous data lakes. In *European Conference on Advances in Databases and Information Systems*, pages 35–49. Springer, 2018.

24 Weigang Hou, Zhaolong Ning, Lei Guo, and Xu Zhang. Temporal, functional and spatial big data computing framework for large-scale smart grid. *IEEE Transactions on Emerging Topics in Computing*, 7(3):369–379, 2017.

25 Gang-Hoon Kim, Silvana Trimi, and Ji-Hyong Chung. Big-data applications in the government sector. *Communications of the ACM*, 57(3):78–85, 2014.

26 Chin-Ho Lin, Liang-Cheng Huang, Seng-Cho T Chou, Chih-Ho Liu, Han-Fang Cheng, and I-Jen Chiang. Temporal event tracing on big healthcare data analytics. In *2014 IEEE International Congress on Big Data*, pages 281–287. IEEE, 2014.

27 Ives Cavalcante Passos, Benson Mwangi, and Flávio Kapczinski. Big data analytics and machine learning: 2015 and beyond. *The Lancet Psychiatry*, 3(1):13–15, 2016.

28 Philip Russom et al. Big data analytics. *TDWI best practices report, fourth quarter*, 19(4):1–34, 2011.

29 Michael Stonebraker. Sql databases v. nosql databases. *Communications of the ACM*, 53(4):10–11, 2010.

30 Allen Van Gelder. The well-founded semantics of aggregation. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 127–138, 1992.

31 Xiaochun Yun, Guangjun Wu, Guangyan Zhang, Keqin Li, and Shupeng Wang. Fastraq: A fast approach to range-aggregate queries in big data environments. *IEEE Transactions on Cloud Computing*, 3(2):206–218, 2014.

32 Junsheng Zhang, Changqing Yao, Yunchuan Sun, and Zengquan Fang. Building text-based temporally linked event network for scientific big data analytics. *Personal and Ubiquitous Computing*, 20(5):743–755, 2016.

Model Checking of Stream Processing Pipelines

Alexis Bédard
Laboratoire d'informatique formelle, Université du Québec à Chicoutimi, Saguenay, Canada

Sylvain Hallé
Laboratoire d'informatique formelle, Université du Québec à Chicoutimi, Saguenay, Canada

Abstract

Event stream processing (ESP) is the application of a computation to a set of input sequences of arbitrary data objects, called "events", in order to produce other sequences of data objects. In recent years, a large number of ESP systems have been developed; however, none of them is easily amenable to a formal verification of properties on their execution. In this paper, we show how stream processing pipelines built with an existing ESP library called BeepBeep 3 can be exported as a Kripke structure for the NuXmv model checker. This makes it possible to formally verify properties on these pipelines, and opens the way to the use of such pipelines directly within a model checker as an extension of its specification language.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases stream processing, model checking

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.5

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.5386462`

Funding Canada Research Chair on Software Specification, Testing and Verification

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Information systems generate logs from a variety of sources: business process management engines, web servers and instrumented programs alike produce sequences of data elements called *event streams*. The analysis of these logs, either offline or in real-time, can be put to numerous uses: computation of various metrics, evaluation of compliance with respect to a policy [34], detection of anomalous patterns or presence of bugs [36]. The field of Complex Event Processing (CEP) concentrates on the real-time evaluation of expressive queries over streams of events [28]. Typically, CEP scenarios involve not only the expression of temporal patterns, but also arithmetical (counts, sums) or even statistical computations over event data. The development of tools and libraries for the processing of event streams has seen a rapid growth in the past decade, with popular products such as Siddhi [33], Esper [2] or Apache Flink [1].

As we shall see, none of these stream processing frameworks is directly amenable to the formal verification of properties on their execution. This is an important gap, as multiple tasks related to the management of event pipelines, which would require the establishment of invariants, are currently impossible. The first obvious example is correctness, which is the guarantee that a processing pipeline produces the expected result for all possible input streams. The static verification of a processing pipeline can also be put to other uses: identifying "dead paths" (parts of a pipeline that never contribute to the output and are therefore useless), verifying the equivalence of two implementations of the same computation, or making sure that I/O buffers allocated to each part of the chain are of sufficient size.

This paper addresses this issue, by describing a formalization of event stream processing pipelines as Kripke structures that can be handled by a model checker. Section 2 briefly describes an existing stream processing library called BeepBeep [24]. Section 3 introduces an extension to this library that makes it possible to export a BeepBeep pipeline into an input file for the NuXmv model checker [15]. Section 4 illustrates the use of this extension by presenting a few scenarios that require the model checking of stream processing programs; an overview of the performance, in terms of execution time, is provided by experiments discussed in Section 5. Section 6 gives a broad portrait of the state of the art in event stream processing related to this question, while Section 7 touches upon future research opportunities.

2 Stream Processing with BeepBeep

Taken in its broadest sense, event stream processing can be defined as the application of a computation over a sequence of data elements called *events*. This computation is typically executed in a "streaming" fashion: its output (generally another event sequence) is progressively produced as input events of the sequence are consumed. Classical applications of this model include the realtime calculation of descriptive statistics (such as the average of values over a sliding window) and the detection of sequential patterns of events in the sequence.

We now describe BeepBeep, an open source event stream processing engine implemented in Java [20]. Over the years, BeepBeep has been involved in multiple case studies [10,11,21,25,36]. We briefly provide here a formal description of the operation of the system's core elements. For further details, the reader is referred to a complete textbook describing the system [24].

Let Σ be a set of arbitrary data elements $\{\sigma_1, \sigma_2 \dots\}$ called *events*; an event stream, noted $\overline{\sigma}$, is an element of the set Σ^*. The notation $\overline{\sigma}[i]$ is used to denote the i-th event of $\overline{\sigma}$. We denote by $\overline{\sigma} \preceq \overline{\sigma}'$ the fact that $\overline{\sigma}$ is a prefix of $\overline{\sigma}'$. If $\Sigma_1, \dots, \Sigma_n$ are event alphabets, a stream vector $\vec{v} = \langle \overline{\sigma}_1, \dots, \overline{\sigma}_n \rangle$ is an element of $(\Sigma_1 \times \cdots \times \Sigma_m)^*$; note that this imposes that each stream within the vector is of the same length. A stream vector $\langle \overline{\sigma}_1, \dots, \overline{\sigma}_n \rangle$ is a prefix of another vector $\langle \overline{\sigma}'_1, \dots, \overline{\sigma}'_n \rangle$ if each $\overline{\sigma}_i$ is a prefix of $\overline{\sigma}'_i$. Given a n-stream vector $\vec{v} = \langle \overline{\sigma}_1, \dots, \overline{\sigma}_n \rangle$ and an n-uple $v = (\sigma_1, \dots, \sigma_n)$, the concatenation $\vec{v} \cdot v$ is the n-uple $\langle \overline{\sigma}_1 \cdot \sigma_1, \dots, \overline{\sigma}_n \cdot \sigma_n \rangle$; this notion can then easily be extended to the concatenation of n-stream vectors.

What in BeepBeep are called "processors" are functions $\pi : (\Sigma_1 \times \cdots \times \Sigma_m)^* \to (\Sigma'_1 \times \cdots \times \Sigma'_n)^*$, with the condition that $\vec{v} \preceq \vec{v}'$ implies $\pi(\vec{v}) \preceq \pi(\vec{v}')$. The values of m and n are the input and output *arity* of the processor, which is often represented with the notation $m{:}n$. The core of the BeepBeep engine is made of a handful of basic processors performing elementary operations on streams. Each of these processors is represented graphically as a box with input and output "pipes", and its operation is symbolized by a standardized pictogram. The most common of these processors are shown in Figure 1.

First, the *ApplyFunction* processor lifts any function $f : \Sigma_1 \times \cdots \times \Sigma_m \to \Sigma'_1 \times \cdots \times \Sigma'_n$ into a processor $\pi : (\Sigma_1 \times \cdots \times \Sigma_m)^* \to (\Sigma'_1 \times \cdots \times \Sigma'_n)^*$ defined as $\pi(\vec{v} \cdot (\sigma_1, \dots, \sigma_m)) \triangleq \pi(\vec{v}) \cdot f(\sigma_1, \dots, \sigma_m)$. *CountDecimate* is a 1:1 processor that keeps one event every k, and is defined as $\pi(\vec{v}) \triangleq \langle \vec{v}[0], \vec{v}[k], \vec{v}[2k], \dots \rangle$. *Trim* removes the first k events of the stream and is defined as $\pi(\vec{v}) \triangleq \langle \vec{v}[k], \vec{v}[k+1], \vec{v}[k+2], \dots \rangle$; *Fork* is a 1:$n$ processor that simply replicates its input on n independent outputs: $\pi(\vec{v}) \triangleq \langle \vec{v}, \dots, \vec{v} \rangle$. *Filter* is a processor $\pi : (\Sigma \times \{\top, \bot\})^* \to \Sigma^*$ that discards events based on a stream of Boolean values. The event at position n in the first stream is sent to the output, if and only if the event at the same position in the second stream is the Boolean value true; formally: $\pi(\vec{v} \cdot (\sigma, b)) \triangleq \pi(\vec{v}) \cdot \sigma$ if $b = \top$, and $\pi(\vec{v})$ otherwise.

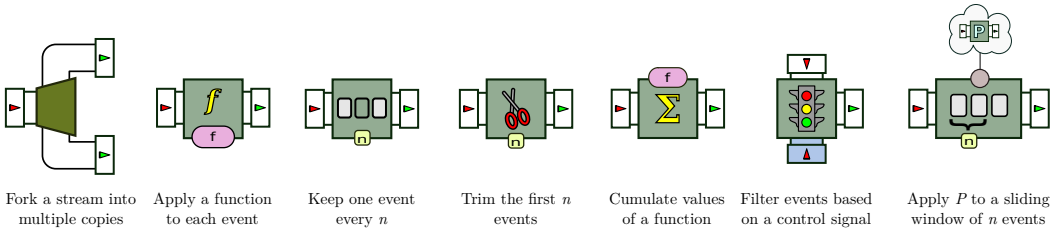
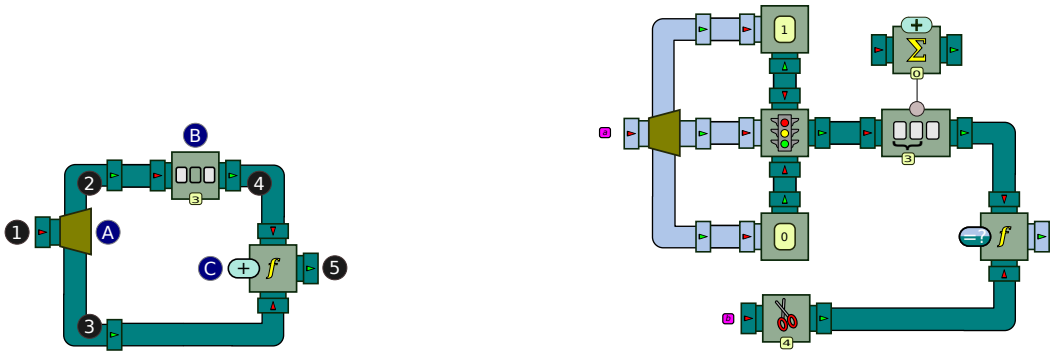


Figure 1 Pictorial representation of BeepBeep's core processors.



(a) Sum of $\overline{\sigma}[i]$ and $\overline{\sigma}[3i]$.

(b) A more complex pipeline.

Figure 2 Two examples of processor pipelines discussed in the text.

As its name implies, the *Cumulate* processor is designed to "accumulate" the successive values of a binary function. Given a function $f : \Sigma^2 \to \Sigma$ and an implicit initial value $\sigma_0 \in \Sigma$, the processor is defined recursively as: $\pi(\langle\sigma\rangle) \triangleq \langle f(\sigma_0, \sigma)\rangle$, and $\pi(\langle\overline{\sigma} \cdot \sigma\rangle) = \pi(\langle\overline{\sigma}\rangle) \cdot \langle f(\pi(\langle\overline{\sigma}\rangle)[-1], \sigma)\rangle$, where $\pi(\langle\overline{\sigma}\rangle)[-1]$ stands for the last event produced by π on the input stream $\overline{\sigma}$. This generic construction can represent various types of computations depending on the function used. For example, if f is addition and $\sigma_0 = 0$ is used as the start value, π produces an output stream where the i-th event is the sum of all input events up to the i-th. If f is Boolean conjunction and $\sigma_0 = \top$, π produces an output stream where the i-the event is the conjunction of all input events up to the i-th.

Finally, the *Window* processor is probably the most complex included in BeepBeep's core. It is parameterized by another processor $\pi' : \Sigma^* \to \Sigma'^*$, which is used to evaluate a sub-stream of k successive events in the global input stream:

$$\pi(\langle\sigma_0 \cdots \sigma_n\rangle) \triangleq \langle\pi'(\langle\sigma_0 \ldots \sigma_{k-1}\rangle)[-1], \pi'(\langle\sigma_1 \ldots \sigma_k\rangle)[-1], \ldots, \pi'(\langle\sigma_{n-k} \ldots \sigma_n\rangle)[-1]\rangle$$

Intuitively, the first output event of π is the last event produced by π' on the window of width k running from input events 0 to $k-1$; The second output event of π is the last event produced by π' on the window of width k running from input events 1 to k, and so on. Note that this processor produces no output event until it receives its first k input events. This construction is very generic, and distinguishes BeepBeep from other stream processing engines, as typically, sliding windows only apply to aggregations over numerical values; in BeepBeep any computation can be put in a sliding window, as π' is an arbitrary *processor*.

Not represented here is the *Group* processor, which makes it possible to encapsulate a complete pipeline and give it as an argument to another processor, as if it were a single "black box".

In order to perform more complex computations, processors can be composed (or "piped") together, by letting the output of one processor be the input of another. This piping is possible as long as the type of the first processor's output matches the second processor's input type. Figure 2a shows a graphical rendition a possible pipeline, where events flow from left to right. It represents a calculation made of three processors (A–C) connected by five pipes (1–5), and where the i-th output event is the sum of input events at positions $3i$ and i.

It is important to note that BeepBeep processors are not defined on tuples of streams, but are rather in terms of streams of tuples. For processors with an input arity of 2 or more, this entails that the processing of their input is done *synchronously*: a computation step is performed if and only if an event can be consumed from each input stream. This is a strong assumption; many other CEP engines allow events to be processed asynchronously, meaning that the output of a query may depend on what input stream produced an event first. As a consequence, processors must implicitly manage buffers to store input events until a result can be computed. This buffering is implicit: it is absent from both the formal definition of processors and any graphical representation of their piping. Consider for example the input stream $0, 1, 2, \ldots$ fed to the previous pipeline. The fork A sends the event to the input of B and the lower pipe of C, and processor B lets the event through to the upper pipe of C; therefore, the sum $0 + 0$ is computed. However, feeding the next event (1) to the chain results in no output. Fork A sends the event through pipes 2 and 3, but processor B discards this event. As a result, processor C cannot consume an event from both its inputs, and event 1 is stored in the input queue associated to its second input pipe.

3 A Formal Modeling of BeepBeep Processors

This section presents a framework that enables the model checking of stream processing models, based on the synchronous formal semantics of BeepBeep processors described in Section 2. More precisely, it shows how a processor chain can be turned into a Kripke structure specified in the form of an input model for the NuXmv model checker [15].

3.1 Design Hypotheses

The translation of a processor pipeline into a finite-state model rests on a number of hypotheses, which are necessary in order to address some of the limitations incurred by the use of a model checker.

The first obvious constraint is related to the use of event queues by processors. These queues are theoretically unbounded, as is exemplified in the pipeline of Figure 2a; in this scenario, the size of the input queue for the bottom pipe of processor C grows indefinitely. The translation to a NuXmv model must therefore impose a fixed maximum size Q to the input queues of each processor. Event types are restricted to the scalars supported by the target model checker, namely Booleans, integers and symbolic constants. Integers themselves are bounded to the range $[0, N-1]$, so all arithmetic operations are implicitly assumed to be performed modulo N.

BeepBeep allows events to be generated in two modes. In *pull* mode, the handling of events in the processor pipe is triggered by requesting for a new output event, which propagates upstream. In order to produce this output event, the processor may require itself to fetch new events from its input(s), which in turn may ultimately lead to fetching events from the original event stream. On the contrary, in *push* mode, output events are produced by signaling the arrival of new events at the input side of the processor pipe. This

may trigger the computation of an output event by the processor, propagating push signals downstream. Our proposed modeling currently simulates a processor chain that operates in push mode only.

Computations in the target model are performed in a lockstep fashion. For example, in the pipeline of Figure 2a, we have seen that pushing an event in pipe 1 results in this event being pushed through pipes 2 and 3; this, in turn, triggers the evaluation of this event through the *CountDecimate* processor, which may result in the event being pushed in pipe 4, and so on. In the generated model, all computations in a chain resulting from a single upstream push operation occur in a single transition. This design choice reduces the potential state space of the resulting system. Were it defined in such a way that each processor along the chain required its own transition, an event pushed through a straight chain of n processors would result in $n + 1$ successive states of the model; our definition rather conflates them into a single transition between two states.

However, this also brings challenges of its own. First, the definition of some processors becomes more intricate, as some operations that are easily described through loops must somehow be "flattened" into a single operation (we shall see that the *Window* processor presents a particular challenge in this respect). Second, this design only handles processors that produce 0 or 1 output event for a given input event. Fortunately, most processors satisfy this condition, including all those presented in this paper.

3.2 Pipeline Modeling

Based on these hypotheses, it is now possible to define the translation of a BeepBeep pipeline into a corresponding NuXmv model. The first element to simulate is that of a pipe connecting two processors. Each pipe is represented by a triplet of variables in the model, called p_c, p_b and p_ℓ. First, p_c is the variable that carries the events themselves: its value at a given computation step indicates the event that this pipe carries from upstream to downstream at that moment. Variable p_b is a Boolean flag set to value $\top$ when the pipe does contain an event, and to $\bot$ otherwise. Finally, variable p_ℓ is also a Boolean flag that indicates whether the event being pushed in the pipe is the last of the chain. Some processors use this signal to perform a different action when receiving what is announced as the last event of a stream.

The modeling of processors takes advantage of NuXmv's concept of *modules*, which makes it possible to encapsulate a set of internal variables and transitions into a self-contained computation unit. Each module instance can accept a number of parameters, which it can read from or write to, in addition to its own internal variables. Each module corresponding to a m:n processor has the same signature, made of $3m + 3n + 1$ parameters. Its first $3m$ parameters are the triplets that define its m input pipes; its next $3n$ parameters define its n output pipes; a final parameter is a Boolean *reset* flag used to signal the processor that it should revert its internal state to a predefined initial state.

The composition of processors in a pipeline is achieved by creating as many triplets of variables as there are pipes in the chain, and passing to each module instance the appropriate pipe variables either as inputs or outputs. An example is shown in Figure 3, where a fragment of the NuXmv code for the pipeline of Figure 2a is presented. Variables $p_{c,i}$, $p_{b,i}$ and $p_{\ell,i}$, for $i \in [1, 5]$, and a global reset flag *prst* are first declared. Then, the three processors are represented by variables π_i, whose type is a module corresponding to the appropriate processor. Finally, one can observe by the input and output parameters of each processor that their connections are made in accordance with the illustration.

```
MODULE main
  VAR
    pc_1: 0..N;
    pb_1: boolean;
    pl_1: boolean;
    prst: boolean;
    ...
    pi_1: Fork_2(pc_1, pb_1, pl_1, pc_2, pb_2, pl_2, pc_3, pb_3, pl_3, prst);
    pi_2: Decimate_3(pc2, pb_2, pl_2, pc_4, pb_4, pl_4, prst);
    pi_3: ApplyFuction(pc_4, pb_4, pl_4, pc_3, pb_3, pl_3, pc_5, pb_5, pl_5, prst);
```

Figure 3 Creating the pipeline of Figure 2a in NuXmv.

Concretely, this process has been implemented by additions to a fork of the original BeepBeep library.[1] BeepBeep already provides a class that allows one to navigate through a processor pipeline using the *Visitor* design pattern [19]. Our implementation adds a new object, called `SmvCrawler`, which traverses a pipeline and makes an inventory of all the processors encountered, along with their respective connections. Each connection results in a unique triplet of model variables, and the processor variables are instantiated according to their type and their input/output relationships. The resulting NuXmv code is then printed to a file.

3.3 Processor Modeling

Any processor that can be exported as a NuXmv module must implement a Java interface called `SmvPrintable`, which consists of a single method called `printSmv()`. This method receives as arguments the user-specified values of Q and N described in Section 3.1. It prints to the output the NuXmv code snippet defining its corresponding module. For some processors, such as *Trim* and *CountDecimate*, the translation is straightforward and does not depend on Q and N. We describe in the following the details of the translation for processors presenting particular challenges.

The first such processor is *ApplyFunction*, for functions of input arity $m > 1$. As explained earlier, this processor must handle input queues, whose behavior needs to be simulated through arrays. Concretely, the queue associated to an input pipe is represented by two array variables, q_c and q_b. The role of these two variables is similar to p_c and p_b: the first contains the actual events inside the queue, while the second is used to record whether an event is contained in the queue at a given position. Depending on the combination of input pipes where an event is pushed at a given computation step, the transition relation of the processor must take care of shifting queue contents forwards or backwards into the arrays. The behavior of this processor when an event must be added to a full queue is left undefined (this shall be discussed later).

The *ApplyFunction* processor is also special in that it is parameterized by another object (the function f to apply). Therefore, BeepBeep functions themselves are also represented as stateless NuXmv modules: a m:n function becomes a module with $m + n$ arguments, and whose transition relation in each state sets to the n outputs the result of applying f on the m inputs. Consequently, each instance of *ApplyFunction* module needs only to be passed the module created for its function f; however, this is done in a modular fashion, as its module merely sets inputs and reads outputs from another abstract module corresponding to the function to apply. A similar process is done for *Cumulate* which, in addition to the module corresponding to the function f to accumulate values from, also has an internal variable *last* to store the value produced by f on the last push of an event.

[1] `https://doi.org/10.5281/zenodo.5386462`

Finally, the *Window* processor deserves some discussion. We recall from Section 2 that sliding windows in BeepBeep can be applied on any processor π –including stateful processors. However, the design hypotheses elicited in Section 3.1 stipulate that each push action on a pipeline must be accounted for in a single transition of the model. This rules out the "naïve" way of handling windows, which would be to wait for k events to be received, push them into a fresh instance of π, collect the last event it produces, and push it downstream.

The solution we propose works differently, and makes use of the *reset* flag that each processor receives. For a window of width k, the *Window* module maintains as internal variables k instances $P_1, \ldots, P_k$ of the processor π to be applied on a window. Each successive processor receives the same input stream, with an increasing number of events trimmed from the beginning. Hence, P_1 receives the whole stream, P_2 receives all events starting from the second, and so on. An internal variable c determines which of these instances is due to be considered for the next output event. Once an instance P_c has received k events, its output is collected and set as the output of the *Window* processor, and that instance is then reset to its initial state by setting its *reset* flag to $\top$. The value of c then shifts to the next processor instance. In such a way, any processor can be applied to a sliding window, yet still use a fixed number of variables and be processed in a single transition of the model. To the best of our knowledge, this implementation is the first to consider stream processing pipelines with cumulative functions and arbitrary sliding windows, which makes it possible to represent complex operations on streams, such as "windows on windows", and the like.

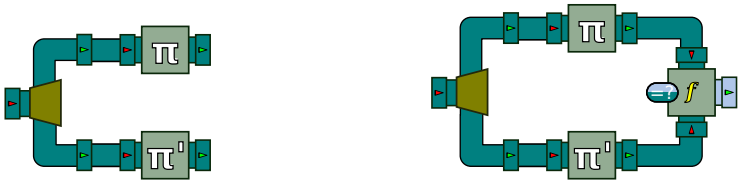
Finally, the *Group* processor can be handled easily. Each group is exported as an independent NuXmv module, where a fresh instance of `SmvCrawler` is instructed to generate the contents of the group (including any other module it may contain). Its piping is then taken care of in a similar way as the `main` module, with the exception that the group is exported as a module with a unique name, and with parameters corresponding to its open input and output pipes.

4 Applications

Equipped with the capability of performing model checking on stream processing pipelines, a number of applications become possible. A first obvious case is the verification of an arbitrary invariant on the execution of a processor chain, expressed as a temporal logic formula involving the state variables that represent the inputs and the outputs of the chain. For example, liveness is a property stating that a pipeline may always output one more event in the future; if p_b is the Boolean variable associated to the chain's output pipe, this can be expressed in CTL as $\mathbf{AG}\,\mathbf{EF}\,p_b$. Bounded liveness is a variant of liveness stipulating that the processor pipeline never remains silent for more than k successive computation steps. This stronger condition is written as an LTL property $(\mathbf{G}\,(p_b \vee \mathbf{X}\,(p_b \vee \ldots)))$.

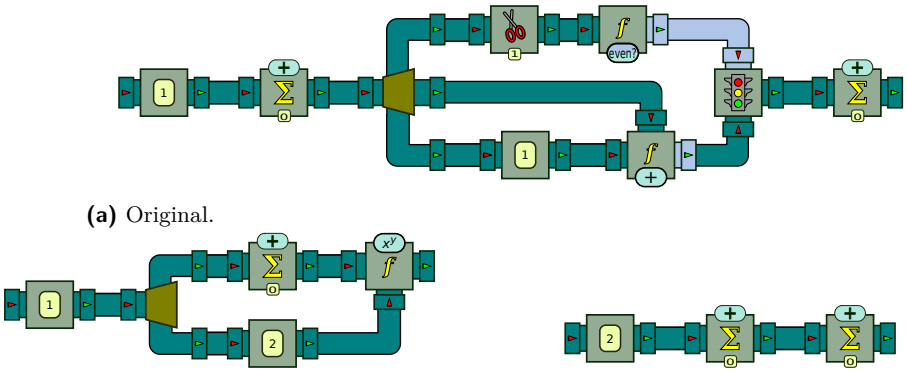
4.1 Implementation Comparison

We shall now mention a few other, less trivial applications that rest on model checking. A first possibility is to check that two pipelines perform the same computation. The basic setup for doing so is illustrated in Figure 4a. If p_c and p_b represent the pair of variables corresponding to the output pipe of π, and p'_c and p'_b represent the pair of variables corresponding to the output pipe of π', the equivalence between the two implementations can be expressed as the LTL formula $\mathbf{G}\,(p_b = p'_b \wedge (p_b \rightarrow (p_c = p'_c)))$. This is what we call *stepwise equivalence*: $\pi(\vec{v}) = \pi'(\vec{v})$ for every stream vector $\vec{v}$; that is, π and π' produce events at the same time, and the same event is produced in such a case. In case the equivalence is not verified, the model



(a) Step-wise equivalence. **(b)** Sequence equivalence.

Figure 4 Basic setup for comparing two implementations π and π' of the same pipeline.



(a) Original.

(b) Simplification of (a). **(c)** Variant for odd.

Figure 5 A processing pipeline and two simplified versions.

checker produces a counter-example which shows a possible input violating the condition. This can be used by the developer to help pinpoint the cause of the discrepancy and fix the issue. However, this setup is not restricted to strict equality. For example, if π and π' represent pipelines evaluating a Boolean condition on an input stream, one could verify that π' is a (stepwise) conservative approximation of π by checking that $\mathbf{G}\,(p_b = p'_b \wedge (p_b \rightarrow (p'_c \rightarrow p_c)))$.

There are a few situations where comparing two pipelines can be useful. First, one can check that a simplification applied to an original pipeline does not disturb its operation. For instance, Figure 5a shows a relatively convoluted pipeline, formed of 8 processors and 12 pipes; examining its operation, one can discover that this computation actually amounts to producing sequence of square numbers $(1, 4, 9, 16, \ldots)$.[2] In contrast, Figure 5b shows a much simpler chain of processors performing this computation formed of 4 processors and 7 pipes. In order to make sure that the two are actually equivalent, they could be plugged in place of π and π' in Figure 4a.

Step-wise equivalence imposes that π and π' produce output events at the same time: at each computation step, they either both remain silent, or they both produce the same output. A looser condition, called *sequence equivalence*, asserts that π and π' produce the same sequence of events, discarding in each any computation step where no event is produced. This condition can be stated formally as the fact that for any stream vector $\vec{v}$, either $\pi(\vec{v}) \preceq \pi'(\vec{v})$ or $\pi'(\vec{v}) \preceq \pi(\vec{v})$. Such a property is hard to express directly in temporal logic: one must take into account the fact that either π or π' may not produce an output at a given computation step, and then keep track of the relative offset between the output values of π and π'. However, it can be easily verified (within the bounds for Q) by modifying

[2] The rightmost processor is given the sequence of odd numbers, whose sum for n terms is n^2.

the original setup to the one shown in Figure 4b. This time, the outputs of π and π' are piped into a binary processor that evaluates equality between its inputs. By virtue of the operation of processors, the events produced by π and π' will be buffered until two values can be compared on each input. This has for effect that events at matching positions in both output streams will be compared for equality, regardless of the number of computation steps required to produce them.

Using this setup, it is possible to discover that the pipelines of Figures 5a and 5b are sequence equivalent, but are *not* step-wise equivalent. Indeed, although both produce the same stream of output values, in the case of the first pipeline, one must push *two* input events in order to receive the first output event (1). In contrast, the pipeline of Figure 5b produces the first output event immediately after the first push. In other words, the first output stream is $\epsilon, 1, 4, 9, \ldots$, while the second one is $1, 4, 9, \ldots$ The problem of finding simpler equivalents to existing pipelines is far from trivial. Case in point, if one simply replaces the `even` condition by `odd` in the pipeline of Figure 5a, the simplified pipeline becomes that of Figure 5c. Indeed, the pipeline now computes the sum of successive even numbers, which can be accomplished with only 3 processors and 4 pipes. One can see that both simplifications are largely unrelated, in terms of structure, to the original pipeline, and that even a slight change in the original layout can result in drastically different simplifications. The discovery of "dead paths" mentioned in the introduction is simply the special case where a pipeline is compared with a truncated version of itself –the truncation representing the elements of the pipeline that can be deleted.

4.2 Reasoning on Buffers

Input buffers in BeepBeep are unbounded: they are simply instances of the `Queue` interface that can be dynamically resized on demand. However, embedded systems and many resource-critical systems disallow the use of dynamically allocated memory, which entails that all data structures must be of a fixed size. Therefore, it may be desirable to verify that the memory required to evaluate a pipeline never exceeds some given threshold, which can then be used to bound the underlying data structures.

We recall that queues within a BeepBeep processor are represented by module variables q_b, which are arrays of size Q. A full queue therefore corresponds to a state where $q_b[Q] = \top$. If $\{q_{b,1}, \ldots, q_{b,n}\}$ is the set of all module variables representing queue state inside a pipeline, the full queue condition becomes straightforward to express as: $\mathbf{G}\ (\bigwedge_{i=1}^{n} \neg q_{b,i}[Q])$. This property is false exactly when there exists a sequence of input events given to the pipeline that is such that a processor has one of its buffers reaching its maximum size, Q. On the contrary, if the condition is true, the buffers can safely be bounded to a size of $Q - 1$. One can even adopt an iterative approach, where the value of Q is progressively increased or decreased in order to discover the existence of a minimal value for buffer size.

There exist pipelines whose queue size cannot be bounded; one example is Figure 2a. As we have seen, its i-th output is the sum of the input events at positions $3i$ and i; therefore, when the i-th output event is produced, $3i - 1$ events are buffered in the processor's second input queue. Memory consumption grows linearly in the length of the input stream, and therefore no upper bound on buffer size will ever satisfy the previous condition. However, performing this verification with the pipeline of Figure 5a will reveal that a value of $Q \geq 1$ satisfies the condition, while $Q \geq 0$ is sufficient for the pipeline of 5b.

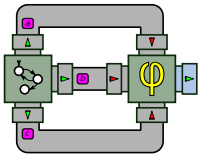


Figure 6 Model checking a system observed by a monitor.

4.3 Stream-Based Formal Verification

So far, the applications we introduced are focused on the model checking of stream pipelines themselves. However, these pipelines can also be fed as input the sequence of states of another system that executes within the model checker environment. This is illustrated in Figure 6. The leftmost box represents a standard Kripke structure K, defined with three state variables x, y and z. The values of these variables at each execution step are taken as streams of values that are fed to a BeepBeep processor pipeline φ, which in turn produces a stream of Boolean values from these input streams.

Intuitively, φ represents a *monitor* that observes a run of K and evaluates a condition on this run; the monitor emits $\bot$ at the moment where the observed run is considered to violate the property, if ever. This corresponds to a classical setup of runtime verification [8], but where both the system and the monitor observing it are simulated within a model checker. Therefore, while a monitor can only provide a verdict for a single execution at a time, its presence within the model checker makes it possible to obtain guarantees for all executions of K. If p_b and p_c are the variables modeling the output of the pipeline, asserting that the property monitored by φ is true for all runs of K simply amounts to model checking $\mathbf{G}\,(p_b \rightarrow p_c)$.

It turns out that such a setup can be useful to perform model checking of properties that would be very inconvenient to express directly as temporal logic formulas on the state variables of K. Consider the following property: at any moment, b contains the number of times a has been true in the k previous states. Expressing this as an LTL formula is possible, but results in a clumsy expression of the form: $\mathbf{G}\,(a \leftrightarrow \mathbf{X}\,(a \leftrightarrow \mathbf{X}\,(a \leftrightarrow \mathbf{X}\,b = k \vee \neg a \leftrightarrow \mathbf{X}\,b = k - 1) \cdots))$. In comparison, the corresponding pipeline that monitors this property is shown in Figure 2b. In addition to being arguably simpler to express, it is also easier to modify: changing the value of k only requires a change on window width, instead of a complete rewrite of the corresponding LTL specification.

5 Experimental Results

We shall now briefly discuss an experimental evaluation of the proposed implementation, by measuring the execution time of NuXmv on a number of BeepBeep pipelines and for a sample of generic properties, with a special focus on the impact of parameters Q and N. Experiments are bundled into a LabPal [23] experimental package that is publicly available online.[3] Overall, the combinations of queue sizes, domain sizes and values of parameter k represent 122 NuSMV input models, which, combined with the set of properties to model check, correspond to 162 distinct model checking problems. Processor pipelines contain between 1 and 22 processors, resulting in Kripke structures with up to 21 distinct modules and 91 variables; for the sake of completeness, they are listed in the appendix. Experiments were run on a Intel Xeon 12-core 3.6 GHz running Ubuntu 18.04, using NuXmv version 2.0.0.

[3] `https://github.com/alexisBedard/nusmv-beepbeep-lab`

Query	No full queues	Liveness	Bounded liveness
Output if smaller than k	53	56	88
Passthrough	24	25	26
Product of 1 and k-th	387	342	1139
Sum of 1s on window	1317	1330	5231
Sum of doubles	1269	1116	11220
Sum of odds	612	526	2075
Sum of window of width 3	4842	4948	15641

(a) Verification time for various properties.

Queue size	Sum of odds	Sum of 1s on window	Pass-through	Sum of doubles	Prod-uct of 1 and k-th	Sum of window of width 3	Output if smaller than k
1	719	4761	23	175	78	23216	49
2	2992	4525	23	1694	206	23385	58
3	2249	4513	24	17012	823	22901	67
4	8207	4458	23	95619	4158	23018	95

(b) Impact of queue size on verification.

Query	Stepwise equivalence	Sequence equivalence
Passthrough	26	34
Passthrough vs delay comparison	31	44
Product of 1 and k-th	176	450
Sum of 1s on window	2085	10868
Sum of odds	1259	1423
Sum of window of width 3	468	10966
Window sum of 2 comparison	488	963
Window sum of 3 comparison	16107	116959

(c) Implementation comparison.

Figure 7 A summary of experimental results. All table entries are in milliseconds.

A first experiment measures the relative time taken to evaluate the properties discussed in Section 4 on the same processor pipeline and the same values of Q and N. These results are plotted in Table 7a. These results show that formal guarantees on pipelines can indeed be checked in reasonable time; one can see that for all processor chains, bounded liveness checking dominates verification time.

Table 7b shows the impact of queue size Q for all pipelines, on the property *No full queues*. It shows, unsurprisingly, that verification time grows exponentially for pipelines having queues, while it remains constant for processor pipelines where no queuing of events ever happens. In the case where the property is false, we observed that NuXmv indeed provides a counter-example input stream that results in one of the processors filling one of its queues; moreover, for all counter-examples we examined, this stream is of the shortest possible length for this to happen. Although not shown, a similar behavior has been observed for the impact of parameter N. Over all models, maximum verification time is 275057 ms, for the pipeline *Sum of window of width 3* on the property *Liveness*.

Finally, Table 7c shows the time taken to verify implementation comparison. This is done here by putting two copies of the same pipeline side by side; in each case, the model checker is asked to verify that they are either step-wise or sequence equivalent. Two other pipelines have also been added, which compare two different implementations of the same pipeline (a sum over a sliding window). These experimental results reveal that step-wise equivalence, although a stronger condition than sequence equivalence, is actually easier to verify.

6 Related Work

Stream processing has been the subject of various formal frameworks and implementations in the recent past. We end this paper by providing a broad overview of existing works related to the verification of properties on streams.

6.1 Stream Processing Engines

A variety of stream processing software and theoretical frameworks have been developed over the years, which all differ in a number of dimensions. For example, TelegraphCQ [13] was built to fix the problem of continuous stream of data coming from networked environments; it shares similarities with the earlier STREAM system [7]. SASE [37] was brought as a solution to meet the needs of a range of RFID-enabled monitoring applications. On its side, Siddhi [33] focuses

on the multi-threading aspect of evaluating CEP queries. Among other popular software, we shall also mention Borealis [5], Cayuga [12], StreamBase SQL [3], StreamInsight [26], and VoltDB [4]. Recently, stream processing engines have started adopting an SQL-like declarative language called the Event Processing Language (EPL) [9]; such systems include Esper [2] and Apache Flink [1]. Despite this large number of concrete implementations, few of them have a formally-defined semantics, or have been studied under the angle of static verification. This absence of formal grounds has for consequence that establishing properties of computations made using these systems is difficult. The use of a model checker has been suggested to solve the scheduling problem in a Synchronous Data Flow (SDF) pipeline [31]. However, to the best of our knowledge, we are not aware of the use of model checking to establish formal properties of the pipeline itself, down at the individual event and buffer level.

We finally mention Lola [16], a formal stream processing language where new event streams are created by combining existing streams through stream operators, and where complex processing is achieved by systems of equations on stream variables. In this context, a system of equations is said to be *efficiently monitorable* when the worst-case memory requirement for the online evaluation algorithm is constant in the length of the input stream; a sufficient condition is demonstrated, by calculating a dependency graph between streams and showing that it contains no cycle of positive weights. It has been shown that features of this language can be reproduced by composing appropriate BeepBeep processor pipelines [22]; therefore, our proposed approach can be seen as a means to indirectly verify properties on Lola models. Section 4.2 has also shown that a constant-memory requirement can be established through model checking on BeepBeep pipelines.

6.2 Formal Models

Since some of these tools are designed as independent computation units passing events downstream in a pipeline, it is somewhat sensible to compare them to formal models of finite-state systems that communicate with each other. Input-output automata [29] are a model of computation in asynchronous distributed networks, composed of a disjoint set of inputs, outputs, and internal actions. The theory of communicating automata (CA) considers networks of such units exchanging messages between each other. In this context, the reachability problem consists of determining if there exists a sequence of interactions in a given network such that each automaton can reaches an internal final state from an initial state [32].

Compared to the BeepBeep event stream processing engine that is the focus of our work, CAs present some important differences. First, CAs are asynchronous models, while BeepBeep is strictly synchronous. Second, CAs define concurrent systems. Finally, CAs use unbounded channels, while BeepBeep's computation units may only send and receive one and only one message at each computation step. A further line of research concentrates on the impact of *lossy channels*, where a message sent by a CA may or may not reach its destination point [6]. Although this problem is relevant in situations involving actual point-to-point (e.g. network) communications, the message exchanges in ESP systems is typically internal through shared memory, where such an issue is much less important.

Modular Petri nets is another model in which individual modules interact via shared places and transitions [14]. Previous work combined Linear Time Logic with modular Petri nets to show how LTL-X model checking can be done on the synchronization graph [27]: the goal is to find an illegal execution of the modular net by synchronizing the Büchi automaton with the visible transitions. Similarly, some works described how model checking can be

using the net unfoldings approach [17, 18, 30], which use partial order techniques to verify concurrent and distributed system. Communicating Transaction Processes (CTP), based on Message Sequence Charts (MSCs), is another model of computation describing a network of communicating processes interacting via common action labels [35]. Essentially, a CTP is a Petri net where the places are the states inside each process. This transition system models non-atomic inter-process communications (event structure) and describes intra-process control flow (transaction scheme); in other words, a process interacts with another before performing some internal computation. The problem of whether a CTP satisfies a specification expressed as a Live Sequence Chart is then discussed.

In comparison to these approaches, BeepBeep processor pipelines are simpler, as communications are unidirectional, and are always performed in global transitions for the whole pipeline at once. However, they allow for more complex forms of computation, such as sliding windows and aggregation, which cannot easily be modeled as PNs. In addition, while the previous works focus on the verification of reachability properties or compliance with a predefined pattern of message exchanges between computational units, our approach is more interested in the verification of input/output relationships of the global pipeline, and not on the intermediate events that are passed within it.

7 Conclusion and Discussion

This paper has shown how a stream processing pipeline can be turned into a finite-state model that simulates its execution, which makes it amenable to the model checking of properties on the original chain. Although the proposed work focuses on the BeepBeep event stream engine, the set of basic operators it handles are common to a large variety of other stream processors. Worthy of mention is that this set includes aggregation and sliding windows; to the best of our knowledge, the formal verification of such expressive types of computations is being studied here for the first time. The specifics of this translation, however, remains system-dependent.

This modeling presents a number of limitations that should be addressed in future work. First are constraints inherent to the underling model checker: only scalar data types are currently handled, whereas a few BeepBeep processors operate on richer types (character strings and variable-sized associative maps) and had to be left out. Our model also supports processors that produce at most one output event for each input event tuple received, which excludes yet a few more processors. Input queues must be given the same (finite) size across all processors of a chain, which makes it impossible to reason about queue size for individual processors.

In counterpart, the paper has shown through a few examples the potential applications that are opened by the model checking of stream pipelines. In addition to straightforward invariants, we shall mention the simplification of processor chains, the calculation of minimum buffer size, and the verification of equivalence between two implementations. The "monitor within a model checker" concept introduced in Section 4.3 especially warrants further scrutiny, as it presents the potential to easily verify properties of a system that would be cumbersome to write directly using temporal logic.

Since the NuXmv model is based on the formal semantics of each processor, it also becomes possible to use a combination of testing and model checking to assess whether the actual Java code implementing each processor is faithful to that semantics. Finally, although the lossy channel problem has not been addressed, it would be easy to include it in our model by having pipes between processors non-deterministically "leak" events, and track the consequences of such an action.

References

1 Apache Flink. URL: https://flink.apache.org, Accessed April 26th, 2021.

2 Esper. URL: http://espertech.com.

3 StreamBase SQL. URL: http://streambase.com.

4 VoltDB. URL: http://voltdb.com.

5 Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stanley B. Zdonik. The design of the Borealis stream processing engine. In *CIDR*, pages 277–289, 2005. URL: http://www.cidrdb.org/cidr2005/papers/P23.pdf.

6 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996. doi:10.1006/inco.1996.0053.

7 A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab, 2004. URL: http://ilpubs.stanford.edu:8090/641/.

8 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2018. doi:10.1007/978-3-319-75632-5_1.

9 Edmon Begoli, Tyler Akidau, Fabian Hueske, Julian Hyde, Kathryn Knight, and Kenneth Knowles. One SQL to rule them all - an efficient and syntactically idiomatic approach to management of streams and tables. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1757–1772. ACM, 2019. doi:10.1145/3299869.3314040.

10 Quentin Betti, Raphaël Khoury, Sylvain Hallé, and Benoît Montreuil. Improving hyper-connected logistics with blockchains and smart contracts. *IT Prof.*, 21(4):25–32, 2019. doi:10.1109/MITP.2019.2912135.

11 Mohamed Recem Boussaha, Raphaël Khoury, and Sylvain Hallé. Monitoring of security properties using BeepBeep. In Abdessamad Imine, José M. Fernandez, Jean-Yves Marion, Luigi Logrippo, and Joaquín García-Alfaro, editors, *FPS*, volume 10723 of *Lecture Notes in Computer Science*, pages 160–169. Springer, 2017. doi:10.1007/978-3-319-75650-9_11.

12 Lars Brenna, Johannes Gehrke, Mingsheng Hong, and Dag Johansen. Distributed event stream processing with non-deterministic finite automata. In Aniruddha S. Gokhale and Douglas C. Schmidt, editors, *DEBS*. ACM, 2009. doi:10.1145/1619258.1619263.

13 Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003. URL: http://www-db.cs.wisc.edu/cidr/cidr2003/program/p24.pdf.

14 Søren Christensen and Laure Petrucci. Modular analysis of petri nets. *Comput. J.*, 43(3):224–242, 2000. doi:10.1093/comjnl/43.3.224.

15 Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A new symbolic model checker. *Int. J. Softw. Tools Technol. Transf.*, 2(4):410–425, 2000. doi:10.1007/s100090050046.

16 Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 166–174. IEEE Computer Society, 2005. doi:10.1109/TIME.2005.26.

17 Javier Esparza and Keijo Heljanko. A new unfolding approach to LTL model checking. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 475–486. Springer, 2000. doi:10.1007/3-540-45022-X_40.

18 Javier Esparza and Keijo Heljanko. Implementing LTL model checking with net unfoldings. In Matthew B. Dwyer, editor, *SPIN*, volume 2057 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 2001. doi:10.1007/3-540-45139-0_4.

19 Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

20 Sylvain Hallé. When RV meets CEP. In Yliès Falcone and César Sánchez, editors, *RV*, volume 10012 of *Lecture Notes in Computer Science*, pages 68–91. Springer, 2016. doi:10.1007/978-3-319-46982-9_6.

21 Sylvain Hallé, Sébastien Gaboury, and Bruno Bouchard. Activity recognition through complex event processing: First findings. In Bruno Bouchard, Sylvain Giroux, Abdenour Bouzouane, and Sébastien Gaboury, editors, *Artificial Intelligence Applied to Assistive Technologies and Smart Environments, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 12, 2016*, volume WS-16-01 of *AAAI Workshops*. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12561.

22 Sylvain Hallé and Raphaël Khoury. Writing domain-specific languages for BeepBeep. In Christian Colombo and Martin Leucker, editors, *RV*, volume 11237 of *Lecture Notes in Computer Science*, pages 447–457. Springer, 2018. doi:10.1007/978-3-030-03769-7_27.

23 Sylvain Hallé, Raphaël Khoury, and Mewena Awesso. Streamlining the inclusion of computer experiments in a research paper. *Computer*, 51(11):78–89, 2018. doi:10.1109/MC.2018.2876075.

24 Sylvain Hallé. *Event Stream Processing With BeepBeep 3: Log Crunching and Analysis Made Easy*. Presses de l'Université du Québec, 2018.

25 Raphaël Khoury, Sylvain Hallé, and Omar Waldmann. Execution trace analysis using LTL-FO+. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA*, volume 9953 of *Lecture Notes in Computer Science*, pages 356–362, 2016. doi:10.1007/978-3-319-47169-3_26.

26 Ramkumar Krishnan, Jonathan Goldstein, and Alex Raizman. A hitchhiker's guide to StreamInsight queries, version 2.1, 2012. URL: http://support.sas.com/documentation/onlinedoc/dfdmstudio/2.4/dfU_ELRG.pdf.

27 Timo Latvala and Marko Mäkelä. LTL model checking for modular petri nets. In Jordi Cortadella and Wolfgang Reisig, editors, *ICATPN*, volume 3099 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 2004. doi:10.1007/978-3-540-27793-4_17.

28 David C. Luckham. *The power of events – An introduction to complex event processing in distributed enterprise systems*. ACM, 2005.

29 Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In Fred B. Schneider, editor, *PODC*, pages 137–151. ACM, 1987. doi:10.1145/41840.41852.

30 Agnes Madalinski and Victor Khomenko. Predictability verification with parallel ltl-x model checking based on petri net unfoldings. *IFAC Proceedings Volumes*, 45(20):1232–1237, 2012.

31 Avinash Malik and David Gregg. Orchestrating stream graphs using model checking. *ACM Trans. Archit. Code Optim.*, 10(3):19:1–19:25, 2013. doi:10.1145/2512435.

32 Anca Muscholl. Analysis of communicating automata. In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 50–57. Springer, 2010. doi:10.1007/978-3-642-13089-2_4.

33 Srinath Perera, Sriskandarajah Suhothayan, Mohanadarshan Vivekanandalingam, Paul Fremantle, and Sanjiva Weerawarana. Solving the grand challenge using an opensource CEP engine. In Umesh Bellur and Ravi Kothari, editors, *DEBS*, pages 288–293. ACM, 2014. doi:10.1145/2611286.2611331.

34 Stefanie Rinderle-Ma and Sonja Kabicher-Fuchs. An indexing technique for compliance checking and maintenance in large process and rule repositories. *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, 11:2:1–2:24, 2016. doi:10.18417/emisa.11.2.

35 Abhik Roychoudhury and P. S. Thiagarajan. Communicating transaction processes: An MSC-based model of computation for reactive embedded systems. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 789–818. Springer, 2003. doi:10.1007/978-3-540-27755-2_22.

36 Simon Varvaressos, Kim Lavoie, Sébastien Gaboury, and Sylvain Hallé. Automated bug finding in video games: A case study for runtime monitoring. *Computers in Entertainment*, 15(1):1:1–1:28, 2017. doi:10.1145/2700529.

37 Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *SIGMOD Conference*, pages 407–418. ACM, 2006. doi:10.1145/1142473.1142520.

A Appendix: Pipelines

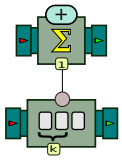
Here are illustrated the pipelines included in the experiments.

Passthrough

The passthrough is a processor that simply outputs whatever it receives as its input.

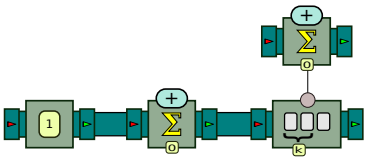
Sum on window of width 3

The pipeline sums three successive input events.



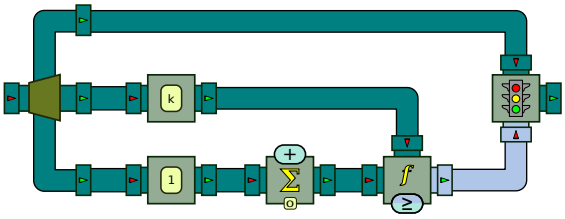
Sum of 1s on window

This pipeline turns every event into a 1, and then sums these values over a window of width k. Therefore, it always returns k as its output events.



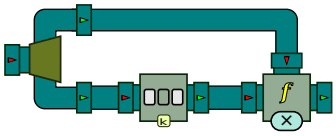
Output if smaller than *k*

This pipeline turns every event into a 1, and then outputs these values only if they are smaller than some constant k. As a result, it only outputs k events, and remains silent after that.



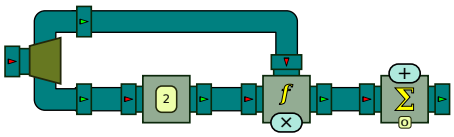
Product of 1 and *k*-th

This pipeline corresponds to a variant of Figure 2a, with multiplication replacing addition.



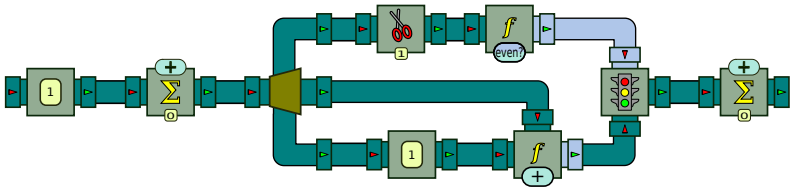
Sum of doubles

This chain multiplies an input event by two, and sums the resulting stream.



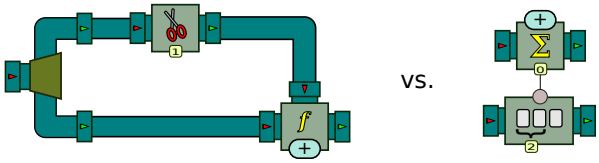
Sum of odds

This example was discussed as Figure 5a.



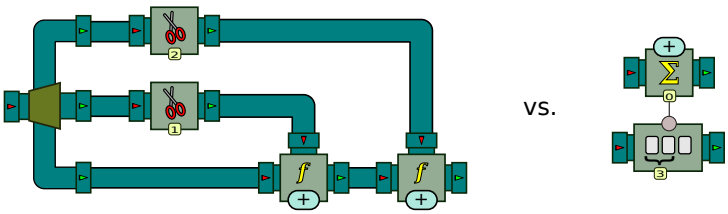
Compare window sum of 2

This model compares two implementations of the sum over a sliding window of width 2.



Compare window sum of 3

This model compares two implementations of the sum over a sliding window of width 3.



Investigation of Database Models for Evolving Graphs

Alexandros Spitalas
Aristotle University of Thessaloniki, Greece

Anastasios Gounaris
Aristotle University of Thessaloniki, Greece

Kostas Tsichlas
University of Patras, Greece

Andreas Kosmatopoulos
Aristotle University of Thessaloniki, Greece

Abstract

We deal with the efficient implementation of storage models for time-varying graphs. To this end, we present an improved approach for the HiNode vertex-centric model based on MongoDB. This approach, apart from its inherent space optimality, exhibits significant improvements in global query execution times, which is the most challenging query type for entity-centric approaches. Not only significant speedups are achieved but more expensive queries can be executed as well, when compared to an implementation based on Cassandra due to the capability to exploit indices to a larger extent and benefit from in-database query processing.

2012 ACM Subject Classification Information systems → Database design and models

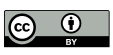
Keywords and phrases Temporal Graphs, Indexing

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.6

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

During a conference in 2009 some of the attendees (405 out of 1200 attendees in total) were carrying a RFID tag that could detect close contacts for two days [16]. They aggregated all daily contact information into two networks (snapshots) for the two consecutive days of the conference. They finally generated snapshots of longer timescales by repeating these two networks and adding some stochastic noise. Their goal was to study a SEIR epidemiological model on the contact network. Such an approach, where a series of aggregated snapshots of the same graph is analyzed, has two main advantages. Firstly, ease of modeling: aggregating the dynamic contact information at a coarser time granularity, e.g., per day, renders the modelling simpler albeit at the expense of losing some information, such as in which exact time period within a day two people met. Secondly, ease of management: storing and managing such time-evolving graphs for long periods is not an easy task, thus having fewer snapshots facilitates their processing and analysis. In this paper, we focus on the second aspect, related to the efficient data management of time-varying graphs. Improving the data management efficiency also mitigates the problem of information loss given that the more efficient the management of a series of snapshots, the higher the frequency at which snapshots can be generated.



Introducing the time dimension in the analysis of networks has been of increasing interest the last years in various scientific fields. This is the reason why such networks have been called dynamic networks, adaptive networks, time-varying networks, evolving networks and temporal networks that essentially refer to the same idea. In [1], a unified framework called TVG (Time-Varying Graph) was proposed and all different formalisms have been shown to be expressed easily in such a framework. Various time-related operations are discussed with respect to their implementation. However, there is no discussion as to how these networks are stored. In general, it seems that there is a lack of a comprehensive framework to actually handle these time evolving graphs, meaning that one should start from the storage model and go all the way to the actual implementation of the algorithm for a specific problem.

TVGs constitute a *graph data structure* with entities corresponding to *vertices* and the relationships between them corresponding to *edges*; both the vertex and edge elements may be annotated by *attributes*, such as name and weight respectively. The distinctive feature of TVGs is their dynamic nature with vertices and edges constantly being inserted, removed or altered as time progresses and entities interact with each other. By studying the evolution of these dynamic graphs we can obtain useful information and metrics regarding the nature of the originating network itself. As a result, one of the greatest challenges that arises in the presence of such *evolving graphs* is maintaining the state of the graph at different time instances (referred to as *snapshots*) in a spatially and temporally efficient way.

1.1 Background and Related Work

There have been two main approaches with regard to a TVG system's design [10, 3, 4], the *time-centric* approach and the *entity-centric* approach (see [2] for a related discussion with a comparison between them). In the former case, the system is indexed according to the time instances (i.e. changes are organized by the time instance they occur in), while in the latter case the system is indexed according to the entities, their relationships and their respective history throughout the snapshots (i.e. changes are organized based on the vertex or edge they refer to). Most of the previous research work conducted so far aims at storing the changes themselves (known as *deltas*) that occur between different snapshots. A system that maintains sets of deltas is thus able to reconstruct any particular snapshot by sequentially applying all the deltas up to the desired time instance. This framework can be used in both approaches but lends itself more naturally to a time-centric approach.

Another viewpoint concerning a system's design is based on the type of queries that the system should be able to evaluate. *Local queries* are based on a particular vertex or a limited selection of adjacent vertices (e.g., the 2-hop neighborhood of a vertex) while *global queries* consider the majority or the entirety of a graph's vertices (e.g., global clustering coefficient). Furthermore, both local and global queries should be able to be executed on either a single snapshot or on a range of snapshots (e.g., average shortest path length between two vertices in the ten first snapshots).

There have been two main research directions over the previous years with regards to evolving graph storage processing. Systems for non-evolving graphs, such as Trinity [14], Pregel [9], and others can be leveraged to support historical queries through explicitly storing each snapshot, but apparently, such solutions are inefficient. A comprehensive survey for evolving graph data management can be found in [5] with the most notable proposals being those in [3, 8, 15, 11]. In general, these techniques rely on storage of snapshots and deltas (logging), which exhibits a trade-off between space and time. Having a large number of snapshots results in deltas of small size but the space cost is substantial since we need to maintain many copies of the graph. On the other hand, having a handful of snapshots means

that deltas will be quite large and queries at specific time instances may require a long time to execute. Three of these proposals operate in a parallel or distributed setting, i.e., DeltaGraph [3], TGI [4] and G^* [8]. Notably, the G^* parallel system takes advantage of the commonalities that exist between snapshots by only storing each version of a vertex once and avoids storing redundant information that is not modified between different snapshots. Furthermore, G^* achieves substantial data locality since each G^* server is assigned its own set of vertices and corresponding entities. On the other hand, G^* uses some form of logging to store connection information between different entities.

We take an entity-centric approach, the storage model of which has appeared in [7]. This new storage model is more space efficient and in most of the cases more time efficient. This model departs from the logging framework (snapshots + deltas) by storing the history at the level of the nodes instead of storing snapshots. It has been attached in the G^* prototype [15] for handling TVGs. Unfortunately, this prototype is incomplete and with severe restrictions that render its use rather impractical (see [6] for a related discussion). There also exist solutions for specific problems that cannot be generalized to arbitrary operations, including historical reachability queries [13], mining the most frequently changing component [17], continuous pattern detection [17] or shortest path distance queries [12]. To tackle these limitations, an early attempt to depart from G^* and use NoSQL has appeared in [6]. In this work, we further improve upon [6] by replacing Cassandra with MongoDB with a view to exploiting additional indexing options and techniques to perform query processing.

1.2 Our contribution

Based on the aforementioned discussion, someone can expect that the time-centric approach is more suited towards evaluating global queries over a few snapshots. At the same time, in order to efficiently handle local queries, an entity-centric approach seems to be the natural choice. While there has been plenty of work revolving around the usage of deltas and (variants of) the time-centric approach, entity-centric systems are at their infancy and have not been thoroughly studied. This paper describes our work on devising efficient storage solutions for the entity-centric model; our work capitalizes on our previous work in [7, 6]. In particular, in Section 2 we describe the vertex-centric storage model given by the authors in [7], and we provide details for two completed implementation approaches of the vertex-centric schema in Cassandra as described in [6], which are shown to outperform solutions based on tailored graph management systems, such as Neo4j. We describe our new approach using MongoDB, which better exploits in-database query processing mechanisms in Section 3. To be more precise, our main motivation behind using MongoDB is to exploit the wider range of indexing options and the capabilities provided to reduce the client involvement when processing queries. Our proposal, which is freely available, is thoroughly evaluated in Section 4 and the results show significant improvement especially for global queries, whereas we manage to run more expensive queries on the same infrastructure. Our focus on global queries is justified by the fact that local queries can be easily and efficiently handled by our purely entity-centric approach. Finally, we conclude in Section 5.

2 The HiNode storage model and its initial implementation

Let $G = (V, E)$ be a graph consisting of a set of *vertices* V and a set of *edges* E. The state of the graph G at a particular time instance t, that is, the active vertices and edges of G at a time instance t, is termed as *snapshot* and is denoted by G_t. We regard time as strictly increasing quantities of indivisible time intervals that follow a linear order. Under this notion of time $\mathcal{G} =< G_1, G_2, \ldots >$ corresponds to a constantly *evolving graph sequence* of snapshots that are to be stored and maintained appropriately.

In [7], the first purely entity-centric, and more specifically, vertex-centric model for maintaining graph historical data, termed as *HiNode* is introduced. Its strongest point is that it builds upon a theoretical storage model that is asymptotically space-optimal, time efficient and supports a general notion of time that needs not be contrained to linear as previously described. The core idea behind HiNode's solution is that a vertex history throughout all snapshots is combined into a set of collections called *diachronic node*. HiNode supports adding or removing vertices and attributes as fundamental operations upon which more complex operations and queries (e.g., graph traversal, shortest path evaluation etc.) are constructed. In HiNode, each change is stored $O(1)$ times, resulting in an asymptotically optimal total space cost. Furthermore, due to the local handling of history, HiNode performs well on local queries and the authors further demonstrate that HiNode on top of G^* is competitive regarding global queries as well when compared to G^* [7].

2.1 Data Structure Overview

A vertex $v \in G_i$ is characterized by a set of attributes (e.g., color), a set of incoming edges from the other vertices of G_i and a set of outgoing edges to the other vertices of G_i. We construct an external interval tree $\mathcal{I}$ that maintains a set of intervals $\{\mathcal{T}^v_{t_s,t_e}\}$ where an interval $\mathcal{T}^v_{t_s,t_e}$ signifies the "lifetime" of a vertex v, i.e. from time instance t_s to time instance t_e. We mark a vertex to be "active" (alive) up until the latest time instance, by setting the t_e value to be $+\infty$. Finally, each interval $\mathcal{T}^v_{t_s,t_e}$ is augmented with a pointer (handle) to an additional data structure for each vertex v, called *diachronic node*.

A diachronic node $\mathcal{D}_v$ of a vertex v maintains a collection of data structures corresponding to the full vertex history in the sequence $\mathcal{G}$, i.e. when that vertex was inserted, all corresponding changes to its edges or attributes and finally its deletion time (if applicable). More formally, a diachronic node $\mathcal{D}_v$ maintains an external interval tree $\mathcal{I}_v$ which stores information regarding all of v's characteristics (attributes and edges) throughout the entire $\mathcal{G}$ sequence. An interval in $\mathcal{I}_v$ is stored as a quadruple $(f, \{\ell_1, \ell_2, \ldots\}, t_s, t_e)$, where f is the identifier of the attribute that has values ℓ_1, ℓ_2, ... during the time interval $[t_s, t_e]$. Note that an edge belonging to v (i.e. one endpoint of the edge is v), can be represented as an attribute of v by using one value ℓ_i to denote the other end of the edge, another value ℓ_j to mark the edge as incoming or outgoing and a last value ℓ_h that is used as a handle to the diachronic node corresponding to the vertex in the other end of the edge. The remaining ℓ values can be used to store the attributes of the edge themselves (e.g., weight). Additionally, the diachronic node maintains a B-Tree for each attribute and for each individual edge of the vertex. Full details are in [7].

2.2 Initial implementation in Cassandra

The first HiNode implementation, hereafter termed as *HiNode-G**[1], was based on extensions to the G^* [8, 15] parallel graph database. This design choice incurred severe limitations regarding the efficiency and scalability of the *HiNode-G** prototype (see [6] for a detailed discussion). In a follow-up work [6], we proposed to leverage NoSQL as the underlying database technology providing preliminary results about two different implementation approaches in Cassandra. These approaches consist of the *Single Table (ST)* and *Multiple Table (MT)* implementations. In the former case, the entire history of a vertex is stored in a single table with each vertex

[1] Source code available at `https://github.com/hinodeauthors/hinode`.

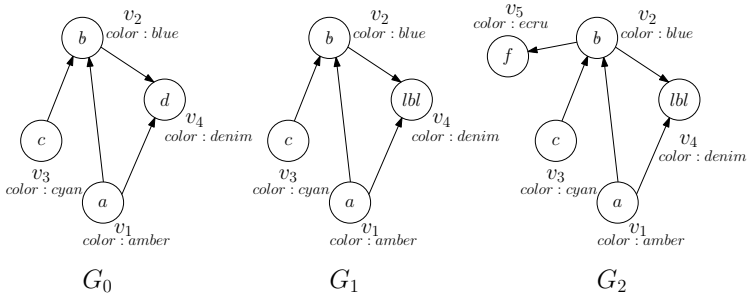


Figure 1 Each vertex possesses two attributes: a name and a color. Additionally, vertices are connected by labelled edges. Three consecutive snapshots are depicted. Snapshot G_1 is obtained by changing the name of v_4 in G_0 from d to lbl. Snapshot G_2 is obtained from G_1 by inserting v_5 and an edge from v_2 to v_5.

Table 1 ST (Single Table) representation for the graph sequence shown in Fig. 1. The fields "start" and "end" correspond to the time range in which the corresponding value is valid.

id	tart	nd	name	color	incoming_edges	outgoing_edges
1	0	∞	[{value: 'a', start: '0', end: '∞'}]	[{value: 'amber', start: '0', end: '∞'}]	null	'2': [{label: 'elbl1', start: '0', end: '∞'}], '4': [{label: 'elbl2', start: '0', end: '∞'}]
2	0	∞	[{value: 'b', start: '0', end: '∞'}]	[{value: 'blue', start: '0', end: '∞'}]	'1': [{label: 'elbl1', start: '0', end: '∞'}], '3': [{label: 'elbl5', start: '0', end: '∞'}]	'4': [{label: 'elbl3', start: '0', end: '∞'}], '5': [{label: 'elbl4', start: '2', end: '∞'}]
3	0	∞	[{value: 'c', start: '0', end: '∞'}]	[{value: 'cyan', start: '0', end: '∞'}]	null	'2': [{label: 'elbl5', start: '0', end: '∞'}]
4	0	∞	[{value: 'd', start: '0', end: '1'}, {value: 'lbl', start: '1', end: '∞'}]	[{value: 'denim', start: '0', end: '∞'}]	'1': [{label: 'elbl2', start: '0', end: '∞'}], '2': [{label: 'elbl3', start: '0', end: '∞'}]	null
5	2	2	[{value: 'f', start: '2', end: '∞'}]	[{value: 'ecru', start: '2', end: '∞'}]	'2': [{label: 'elbl4', start: '2', end: '∞'}]	null

corresponding to a single table row, while in the latter case the data of each vertex is stored in multiple tables with each table corresponding to a single vertex attribute. Tables 1 and 2 show the single table and multi table implementations for the toy example shown in Figure 1.[2]

In order to adequately support global type of queries (i.e. queries that involve a significant part of a snapshot's vertices), the two models offer two querying modes for the retrieval of all vertices relevant to a specified query. Let $[t_s, t_e]$ be a specified time range for which a query is about to be executed. In the first mode (termed *retrieve_all* (*RA*)), and regardless of the given time range, we retrieve all vertices from each model and then perform a client-side filtering operation, where we discard any vertices that do not belong in $[t_s, t_e]$. In the second mode (termed *retrieve_relevant* (*RR*)), in each model, we first determine the vertices that are "alive" at $[t_s, t_e]$ and then, we retrieve them.

While in ST, the implementation of *RR* is straightforward, MT requires additional work since retrieving a particular (set of) attribute(s) during a certain time interval $[t_s, t_e]$ would translate to a range query and the retrieval of all data with a "timestamp" value between t_s

[2] Source code available at `https://github.com/akosmato/HinodeNoSQL`

Table 2 The MT (Multiple Table) representation of the graph sequence shown in Fig. 1. The fields "start" and "end" correspond to the time range in which the corresponding value is valid. "vid" corresponds to the id of the vertex while "sourceid" and "targetid" correspond to the source and the target of a directed edge respectively.

(a) vertex

vid	start	end
1	0	∞
2	0	∞
3	0	∞
4	0	∞
5	2	∞

(b) vertex_name

vid	start	name
1	0	a
2	0	b
3	0	c
4	0	d
4	1	lbl
5	2	f

(c) vertex_color

vid	start	name
1	0	amber
2	0	blue
3	0	cyan
4	0	denim
5	2	ecru

(d) edge_incoming

targetid	start	end	sourceid
2	0	∞	1
2	0	∞	3
4	0	∞	1
4	0	∞	2
5	2	∞	2

(e) edge_outgoing

sourceid	start	end	targetid
1	0	∞	2
1	0	∞	4
2	0	∞	4
2	2	∞	5
3	0	∞	2

(f) edge_label_incoming

targetid	start	sourceid	label
2	0	1	elbl1
2	0	3	elbl5
4	0	1	elbl2
4	0	2	elbl3
5	2	2	elbl4

(g) edge_label_outgoing

sourceid	start	targetid	label
1	0	2	elbl1
1	0	4	elbl2
2	0	4	elbl3
2	2	5	elbl4
3	0	2	elbl5

and t_e (i.e. we are not interested in any updates that occur outside $[t_s, t_e]$). Since Cassandra does not natively permit double-bounded range queries for the sake of efficiency, we fetch the relevant data with a timestamp larger than t_s and then filter all data with a timestamp larger than t_e at the client side. In [6] there is extensive experimental evaluation. The conclusion is that the choice of a particular vertex-centric implementation is not straightforward and exhibits different trade-offs depending on the query at hand.

3 A MongoDB implementation

Our main motivation behind using MongoDB is to exploit the wider range of indexing options and the capabilities provided to reduce the client involvement when processing queries. Additionally, in Cassandra, data are saved as strings and, as such, they are being serialized when returned to the client, while in MongoDB, we have the ability to store the elements of the nodes with a combination of lists and documents. Overall, we are able to perform more complex in-database queries and decrease the client involvement in query processing. Finally, in the new implementation, instead of getting the documents from the database in a single large batch, we have the option to employ a `foreach` approach (when this is expected to be more efficient) and as a result, to mitigate intermediate client-side storage requirements.[3]

3.1 Schema alternatives

Both the ST and MT models shown in Tables 1 and 2 have been transformed to comply with MongoDB's JSON format in a straightforward manner. In addition, we developed an alternative schema for both models, where the elements of the primary key are inserted as characteristics in the document; as primary key, we insert the standard key assigned by MongoDB automatically. The reason for this schema is to further simplify the client-side tasks (i.e., the processing refers to the document content exclusively) with no difference in the capability of answering specific types of queries.

[3] Source code available at `https://github.com/alexspitalas/HiNode-MongoDB/`.

In the ST model, a document is a representation of a diachronic node and consists of the primary key as a triple (`vid, start` and `end` of the node), the incoming and outgoing edges and the vertex attributes. The features forming the key are stored as atomic string values, while the vertex attributes are stored as a list of sub-documents, where each document is a triple. The incoming and outgoing edge metadata are stored as a sub-document containing a list of triples (where each triple is a MongoDB sub-document). The former document is essentially a hashmap structure with the key corresponding to the vertex id, while the nested sub-document stores the attributes and the period for each edge. The following 3 indices are built on: (i) `vid`; (ii) `start` and `end`; the complete key. The first index allows quick retrieval of a specific vertex, while the second and third indices facilitate stabbing queries. Finally, as already mentioned, we have experimented with an alternative ST model created (termed as NoKey), where the key is the default `_id` provided by MongoDB, and `vid, start` and `end` are inserted as characteristics of the document, while the indices are the same.

In the MT model, we split the diachronic node into 3 sets of collections, one about the vertex ((a)-(c) in Table 2), one about incoming edges ((d),(f)) and one about outgoing edges ((e)-(g)). Each set consists of one collection about the time period and the existence of the vertex or edge and one collection for every attribute. The standard indices are on `vid` and `vid,start` in the first set of collections ((a)-(c) in Table 2). For the edge collection sets, the multikey indices are on `sourceID` (or `targetID`) and the `start` timestamp. In summary, the main difference with the Cassandra-based implementation in [6] in terms of modelling is the increased flexibility regarding indices and the fact that sub-documents are stored without being serialized as strings.

3.2 Query processing

For local queries, the server (database) side is straightforward, while most of the work is performed on the client side. The local queries we investigate in this paper are retrieving the history of a vertex and one hop queries. In the former case, we retrieve the history of a specified vertex for some time period. In the latter query, all neighboring vertex ids of the query vertex at a specified time period are returned. Both tasks are supported by the two implementation models in a straightforward manner.

Due to the vertex-centric approach, we investigate global queries since local queries can be supported very efficiently. Global query processing comprises two phases. The first is concerned with the retrieval of the data, while in the second one, the processing of the retrieved data takes place. These phases can be intertwined. In our implementation and experiments, the two phases are separated so that the client's side is the same for all ST-based and all MT-based techniques. Regarding the retrieval of the data, three variants have been developed, *retrieve_relevant* (*RR*), *retrieve_all* (*RA*) and *in-database* (*ID*).

In *RR*, the main objective is to find the relevant documents by retrieving only their necessary characteristics. In *RA*, we retrieve all the characteristics of the document, checking at the same time whether the document is needed for the query. Compared to *RR*, we perform only one read at the database, but we retrieve more data than necessary if the document is not needed for the query; as a result, *RR* is expected to perform better when the amount of data stored per node is much higher than the data needed to establish the necessity of the node. The necessity check, along with the rest of the query execution, is performed on the client. In the new MongoDB implementation, contrary to the initial Cassandra one, we adopt a more incremental (iterative) approach instead of returning all data in a single batch; this has increased the scope of global queries that can be executed without throwing an out-of-memory error.

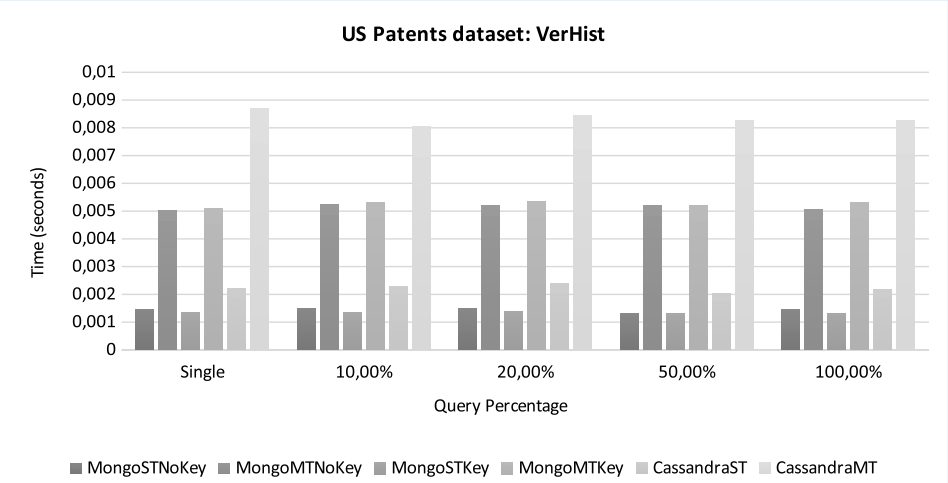


Figure 2 Results for the vertex history query on the US Patents dataset.

However, the most notable difference between the two implementations is that MongoDB naturally lends itself to *in-database* query processing, so that the client gets only the data needed to compute the final results. This is achieved by submitting more complex queries that are supported by the MongoDB driver. To this end, we use the in-database MongoDB mechanisms to perform the necessity checks mentioned in the *RR* technique. Similarly to *RR*, the data needed for the final answer computations are returned incrementally to the client. As such, this approach has even lower space requirements on the client-side and at the same time, it allows for both the server and client working in parallel. It should be mentioned, that in some local queries (like onehop query), it may make sense to adopt an in-database query processing rationale, but this is beyond the scope of this paper.

4 Experiments

In the experiments, we use the same 4 queries as in [6] (Vertex History, One Hop, Average Vertex Degree, Vertex Degree Distribution) in 3 different datasets (hep-Th with 27.77K vertices, 352.8K edges and 156 snapshots; hep-Ph with 34.5K vertices, 421.6K edges and 132 snapshots; and US Patents with 3,774.8K vertices, 16.5M edges and 444 snapshots). We experiment with all MT and ST Cassandra and MongoDB combinations. For MongoDB, we test both key and nokey flavors and all modes of global query processing (*RA*, *RR* and *ID*). Each query is executed referring to a range of snapshots from 1 to all. We use a client application written in Java, and all the experiments were executed on a single node system with i5-3210M, 16GB RAM, and a 500GB SSD, while the client and the databases are collocated on the same machine.

4.1 Local queries

Regarding local queries, we investigated the vertex history query and one hop query query at the 3 datasets. We repeated each query 500 times and we report the average values. For each set of such runs we have a cold start. Fig. 2 presents the results for the US Patents dataset,

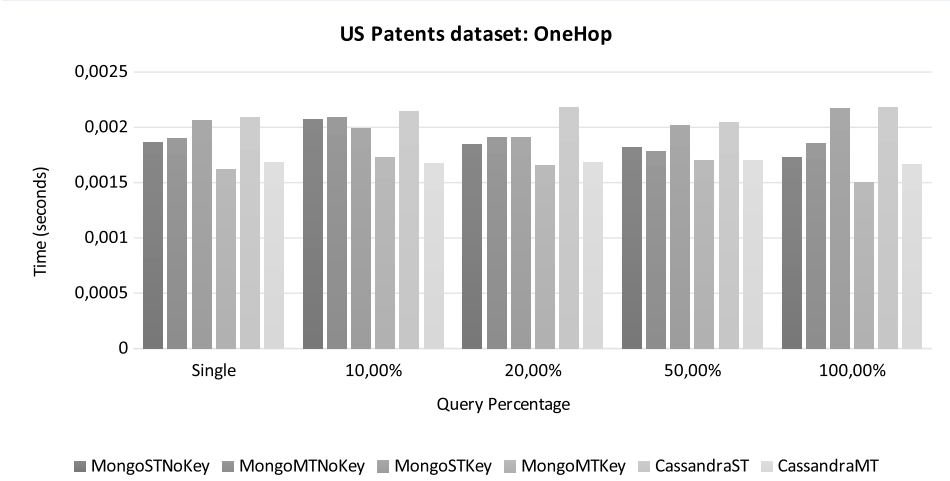


Figure 3 Results for the 1-hop query on the US Patents dataset.

but the results are similar for all datasets. In summary, for the vertex history query, ST outperforms MT by more than 70% (as also mentioned at [6]) while we observe an increase in performance at the best performing ST models executed in MongoDB by up to 42% (in average, it is 39.6% across all snapshot amounts) compared to the Cassandra ones. The best performing MT model in MongoDB is faster by up to 42% compared to its Cassandra counterpart, as well. Another observation is that the performance of the models does not depend on the amount of snapshots in the query. Finally, Key and NoKey settings (recall that in NoKey, the key is the default `_id` provided by MongoDB) have small differences (Key is faster by up to 8.5 %)

In the oneHop query, the observations are mixed but are still consistent across the three datasets (see Fig. 3 for the large dataset). Firstly, the differences between Cassandra and MongoDB are smaller with no dominant database system. More specifically, the differences between the two databases for any range of snapshots are up to 20 % when MongoDB performs better, and up to 6% when the dominant model is the Cassandra-based one, i.e., the differences between the two database systems are smaller than previously. Secondly, MT is better than ST in all cases by up to 26%, while the average performance improvement is 13.2% (due to the fact that only edge info needs to be retrieved). When considering only MT models, the differences between MongoDB and Cassandra do not exceed 10%. Finally, Key and NoKey have larger differences, up to 23 %. In all cases except one, for the MT model, the key version is the dominant one.

4.2 Global queries

For global queries, we demonstrate the results for both Hep-Th and US Patents datasets, since hep-Ph has similar results as hep-Th. The graphs include more query processing modes than previously, since we distinguish between *RA*, *RR* and *ID*.

Average Vertex Degree Query. The results are shown in Figure 4. Our observations for the two datasets are summarized below.

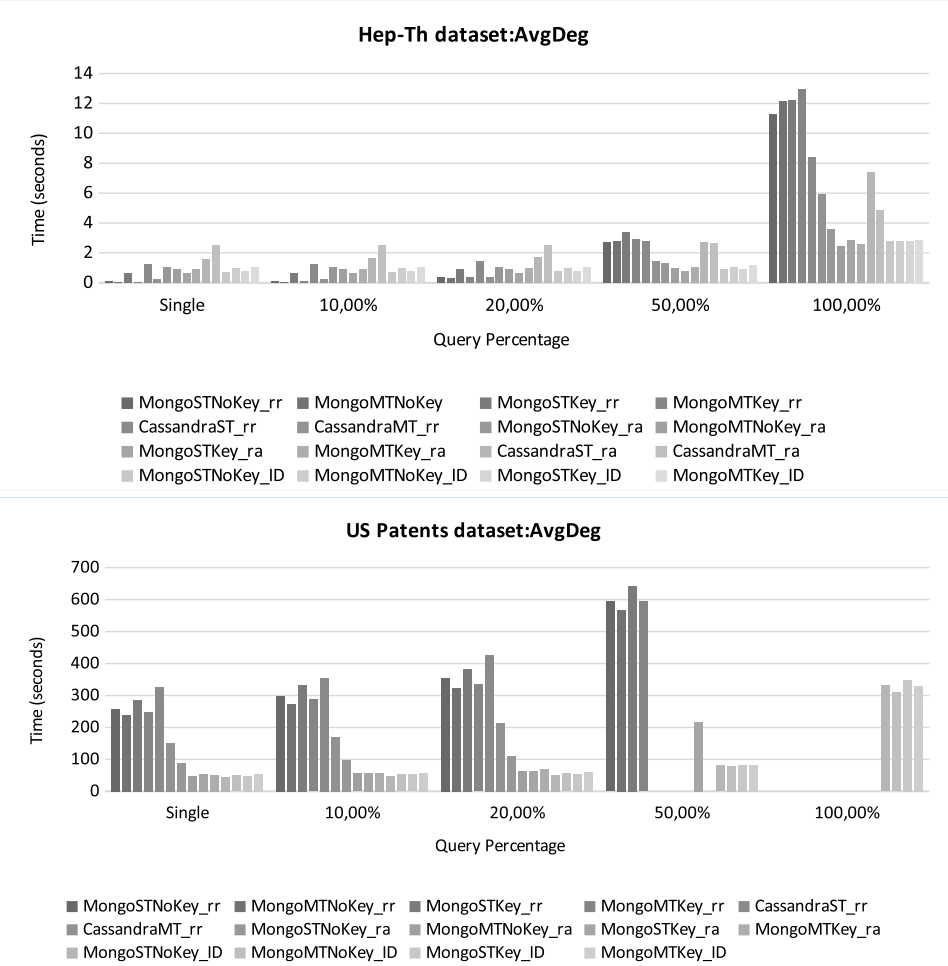


Figure 4 Results for the average degree query on the hep-Th (top) and US Patents (bottom) datasets.

1. For the big dataset (US Patents), the MongoDB in-database techniques are always the most efficient. Regarding the exact storage model, for snapshots in the range of [1-50%), ST models behave better, while for more snapshots, MT is superior. On average, *ID* MongoDB techniques improve upon the best performing Cassandra models (which manage to handle up to 20% snapshots) by up to 75.9% (speedup factor of 4×).
2. For the smaller datasets, when accessing 50% or 100% of the snapsots, the MongoDB *RA* approach is better than the *ID* approach, albeit by a small margin (on average, 11.5%). On the other hand, when accessing [1 to 20 %] of the graph snapshots, MongoDB *RR* approach is better than every other approach. When compared to *ID*, they improve the performance on average by 84%, mostly due to the small intermediate results.
3. MongoDB solutions are superior to those of Cassandra, with the margin in performance increasing with more snapshots. When accessing all snapshots, the MongoDB speedup is 1.96× for the smaller dataset; for the larger dataset, MongoDB managed to return results when employing the *ID* technique, while Cassandra did not manage to run.
4. The *RR* and *RA* query processing methods perform differently in Cassandra and in MongoDB. In Cassandra's ST model, *RA* outperformed *RR* in 50% and 100% of snapshots by less than 12%. In the MT model, it outperformed only when accessing 100% of the graph

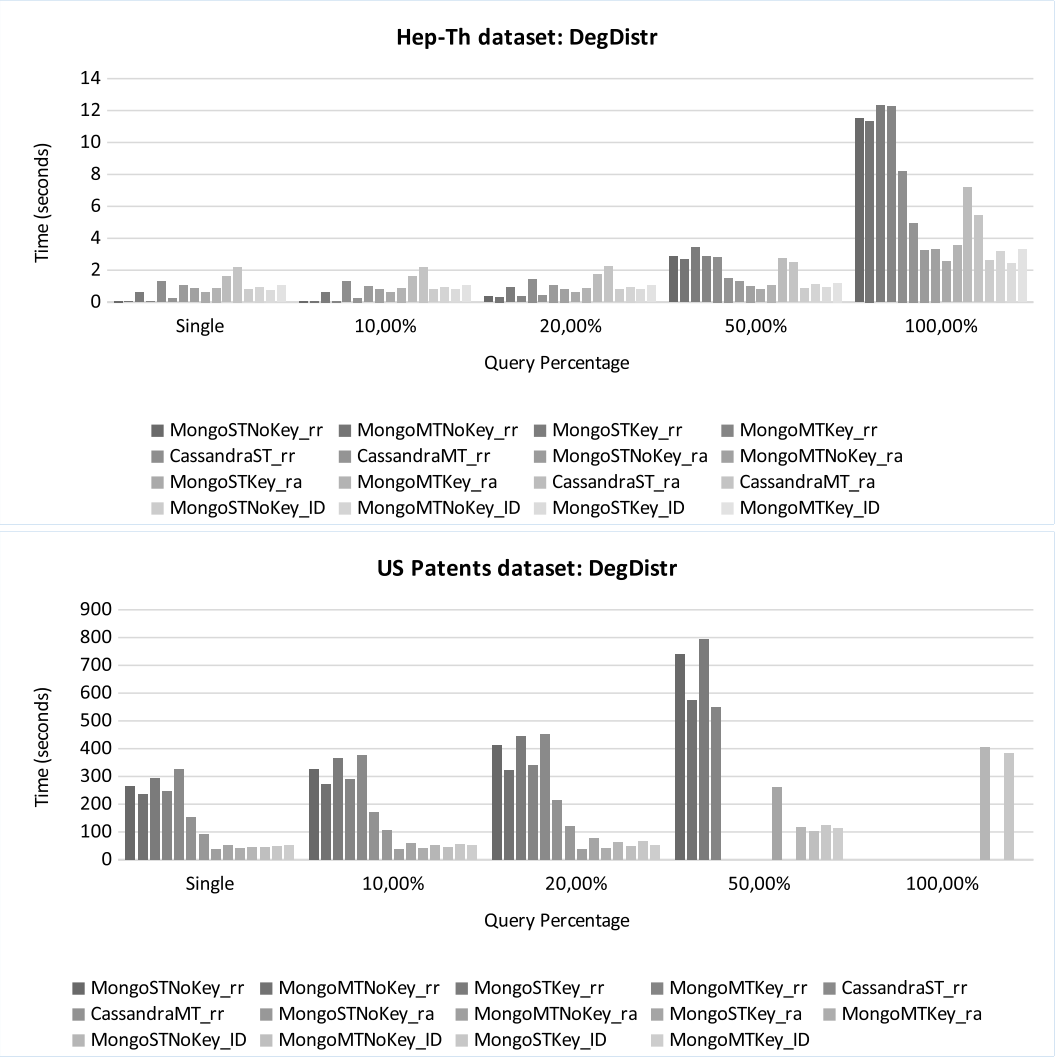


Figure 5 Results for the vertex degree distribution query on the hep-Th (top) and US Patents (bottom) datasets.

(by 18.5%). On the other hand, in MongoDB, both for ST and MT, *RA* outperformed *RR* when the queries were applied on over 50% of the snapshots; the differences for ST were on average 73%, while, for MT, were 71.5%.

5. Overall, while MongoDB is always the main option, the best performing model differs. In smaller datasets, when the query accesses less than half of the snapshots, ST combined with *RR* is more efficient; for more snapshots MT combined with *RA* dominates. In the large dataset, *ID* is always the main option, but ST is more efficient when accessing less than 50% of the snapshots, while MT performs better otherwise.

Vertex Degree Distribution Query. The results are shown in Figure 5. Our main observations are the following:

1. MongoDB ST model combined with the *ID* technique manages to execute all queries over all snapshot ranges; no other combination of choices achieves this, e.g., Cassandra-based solutions can run queries only up to 20% snapshots.

2. For the big dataset, the MongoDB *RA* approach is the most efficient. When accessing up to 20% snapshots, MT combined with *RA* behave better and improve the performance by 77.3% on average. On the other hand, the only implementation that was able to execute on all snapshots was MongoDB MT combined with *ID*, which improved upon the best performing Cassandra models by 74% on average.
3. For the smaller datasets, MongoDB ST combined with *ID* is the best when accessing all snapshots, by up to 51% compared to the Cassandra implementation. When accessing 50% of the snapshots, MongoDB ST combined with *RA* is better than the best-performing Cassandra implementation by 48%. Finally, when the query access up to 20% of the snapshots, MongoDB MT combined with *RR* improves upon their Cassandra counterparts by 58.6% on average.
4. In Cassandra, *RA* does not increase the performance with minor exceptions. On the other hand, in MongoDB, *RA* always increases the performance by up to 80%.
5. The NoKey option performs more efficiently than Key in most cases but by a small margin, i.e., it does not decrease the best performing times by more than 10%.

Finally, to assess the impact of indices, we experimented with a MongoDB model without using indexes on any non-key characteristic. The experiment was performed on the Hep-Th dataset with global queries using only the ST model. The results show a serious performance degradation. More specifically, when we use indexes, the execution time is reduced by up to 98% (improvement by a factor of 41×). This speedup is observed for both global queries.

Space issues. While Cassandra requires less space to store the data since it builds fewer indices and adopts a different storage approach, MongoDB requires less memory on the client while executing the query. This is the result of the iterative approach that was adapted in MongoDB as well as from the *ID* query processing method. The space required for the three datasets (in increasing size) in Cassandra ST was 31.0 MB, 37.4 MB and 1.83 GB, and for MT 45.7 MB, 55.5 MB and 3.10 GB. The space for MongoDB ST was 89.70 MB, 107.37 MB, 4.84 GB, and for MT 218.34 MB, 260.87 MB and 10.96 GB, respectively. On the other hand, MongoDB has exhibited a speedup above 2× for insertions.

5 Summary

In this work, we have shown how the HiNode vertex-centric approach for storing time-varying graphs can be implemented in MongoDB. We have achieved significant improvements for global queries compared to the previous NoSQL based implementation (over 4X in some cases); the speedups were lower for local queries, but such queries are already performed efficiently in any vertex-centric implementation. Our vision is broader. We aim to develop a complete historical graph data management system through extending the described storage layer with more sophisticated query processing and optimization techniques.

References

1 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

2 U. Khurana and A. Deshpande. Storing and analyzing historical graph data at scale. In *EDBT*, 2016.

3 Udayan Khurana and Amol Deshpande. Efficient snapshot retrieval over historical graph data. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 997–1008, 2013.

4 Udayan Khurana and Amol Deshpande. Storing and analyzing historical graph data at scale. In *EDBT*, pages 77–88, 2016.

5 Andreas Kosmatopoulos, Kalliopi Giannakopoulou, Apostolos N Papadopoulos, and Kostas Tsichlas. An overview of methods for handling evolving graph sequences. In *Algorithmic Aspects of Cloud Computing*, pages 181–192. Springer, 2016.

6 Andreas Kosmatopoulos, Anastasios Gounaris, and Kostas Tsichlas. Hinode: implementing a vertex-centric modelling approach to maintaining historical graph data. *Computing*, March 2019. doi:10.1007/s00607-019-00715-6.

7 Andreas Kosmatopoulos, Kostas Tsichlas, Anastasios Gounaris, Spyros Sioutas, and Evaggelia Pitoura. Hinode: an asymptotically space-optimal storage model for historical queries on graphs. *Distributed and Parallel Databases*, 2017. doi:10.1007/s10619-017-7207-z.

8 Alan G. Labouseur, Jeremy Birnbaum, Paul W. Olsen, Sean R. Spillane, Jayadevan Vijayan, Jeong-Hyon Hwang, and Wook-Shin Han. The g* graph database: efficiently managing large distributed dynamic graphs. *Distributed and Parallel Databases*, 33(4):479–514, 2015.

9 Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

10 Jayanta Mondal and Amol Deshpande. Managing large dynamic graphs efficiently. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 145–156, 2012.

11 Chenghui Ren, Eric Lo, Ben Kao, Xinjie Zhu, and Reynold Cheng. On querying historical evolving graph sequences. *PVLDB*, 4(11):726–737, 2011.

12 Betty Salzberg and Vassilis J Tsotras. Comparison of access methods for time-evolving data. *ACM Computing Surveys (CSUR)*, 31(2):158–221, 1999.

13 Konstantinos Semertzidis, Evaggelia Pitoura, and Kostas Lillis. Timereach: Historical reachability queries on evolving graphs. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 121–132, 2015.

14 Bin Shao, Haixun Wang, and Yatao Li. Trinity: a distributed graph engine on a memory cloud. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, pages 505–516, 2013.

15 Sean R. Spillane, Jeremy Birnbaum, Daniel Bokser, Daniel Kemp, Alan G. Labouseur, Paul W. Olsen, Jayadevan Vijayan, Jeong-Hyon Hwang, and Jun-Weon Yoon. A demonstration of the G_* graph database system. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 1356–1359, 2013.

16 Juliette Stehlé, Nicolas Voirin, Alain Barrat, Ciro Cattuto, Vittoria Colizza, Lorenzo Isella, Corinne Régis, Jean-François Pinton, Nagham Khanafer, Wouter Van den Broeck, and et al. Simulation of an seir infectious disease model on the dynamic contact network of conference attendees. *BMC Medicine*, 9(1), 2011.

17 Yajun Yang, Jeffrey Xu Yu, Hong Gao, Jian Pei, and Jianzhong Li. Mining most frequently changing component in evolving graphs. *World Wide Web*, 17(3):351–376, 2014.

Interval Temporal Random Forests with an Application to COVID-19 Diagnosis

Federico Manzella
Dept. of Mathematics and Computer Science, University of Ferrara, Italy

Giovanni Pagliarini
Dept. of Mathematics and Computer Science, University of Ferrara, Italy
Dept. of Mathematical, Physical, and Computer Sciences, University of Parma, Italy

Guido Sciavicco
Dept. of Mathematics and Computer Science, University of Ferrara, Italy

Ionel Eduard Stan
Dept. of Mathematics and Computer Science, University of Ferrara, Italy
Dept. of Mathematical, Physical, and Computer Sciences, University of Parma, Italy

Abstract

Symbolic learning is the logic-based approach to machine learning. The mission of symbolic learning is to provide algorithms and methodologies to extract logical information from data and express it in an interpretable way. In the context of temporal data, interval temporal logic has been recently proposed as a suitable tool for symbolic learning, specifically via the design of an interval temporal logic decision tree extraction algorithm. Building on it, we study here its natural generalization to *interval temporal random forests*, mimicking the corresponding schema at the propositional level. Interval temporal random forests turn out to be a very performing multivariate time series classification method, which, despite the introduction of a functional component, are still logically interpretable to some extent. We apply this method to the problem of diagnosing COVID-19 based on the time series that emerge from cough and breath recording of positive versus negative subjects. Our experiment show that our models achieve very high accuracies and sensitivities, often superior to those achieved by classical methods on the same data. Although other recent approaches to the same problem (based on different and more numerous data) show even better statistical results, our solution is the first logic-based, interpretable, and explainable one.

2012 ACM Subject Classification Computing methodologies → Machine learning algorithms

Keywords and phrases Interval temporal logic, decision trees, random forests, sound-based diagnosis

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.7

Acknowledgements We thank the INdAM GNCS 2020 project *Strategic Reasoning and Automated Synthesis of Multi-Agent Systems* for partial support, the PRID project *Efforts in the uNderstanding of Complex interActing SystEms*, the University of Udine (Italy), the University of Gothenburg (Sweden), and the Chalmers University of Technology (Sweden) for providing the computational resources, and the University of Cambridge (UK) for sharing their data. Moreover, the open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Machine Learning (ML) is at the core of modern Artificial Intelligence. It can be defined as the process of automatically extracting the theory that underlies a phenomenon, and expressing it in machine-friendly terms, so that it can be later used in applications. The potential of ML is limitless, and it ranges from learning rules that classify patients at some risk, to formalizing the factors that influence pollution in a certain area, to recognizing voices, signatures, images, and many others. The most iconic and fundamental separation between the sub-fields of ML is the one between *functional* and *symbolic* learning. Functional

learning is the process of learning a *function* that represents such underlying theory; functions can be as simple as *linear functions*, or as complex as *deep neural networks*. Symbolic learning, on the other hand, is the process of learning a *logical description* that represents a phenomenon. Symbolic learning is sometimes statistically less accurate than functional one, but its results can be interpreted and explained by humans, while, in general, functional models are considered *black-boxes*. Until very recently, symbolic learning models were limited by their underlying logical language, that is, propositional logic, and temporal, spatial, and, in general, non-propositional data were usually dealt with propositional learning by *flattening* the non-propositional dimension using global features (e.g., the average temperature instead of all values within the monitored period). This resulted in the possibility of using off-the-shelf methods after a phase of data abstraction, but severely hampered the interpretability of the results, and in many cases the statistical performances of the extracted model as well.

Interval temporal logic decision trees are a first step in the direction of improving the expressive power of symbolic methods by replacing propositional logic with a more expressive formalism in a classical, well-known schema. They were introduced in [13, 38], and have shown great potential as a method to classify multivariate time series. Temporal logic decision trees are part of a bigger project that aims to establish a whole new area, generally known as *modal symbolic learning*, whose driving principle is precisely the study of symbolic learning schemata and methods based on propositional modal logic. Propositional decision trees can be generalized into *bags of trees* and then into *random forests* [7] to obtain classifiers based on several trees instead of a single one. Sets of trees tend to be more performing than single trees, and while they are considered at the verge between symbolic and functional learning, their symbolic nature is still evident: sets of trees, as single trees, can be analyzed and discussed, and, although the process of extracting rules is not as immediate as in single trees, is still possible [17, 31]. Building on this idea, in this paper we present an approach to *interval temporal logic random forests*, based, as single temporal logic trees are, on interval temporal logic. Interval temporal forests follow the same principles as the propositional ones: as a general rule, a forest is a schema based on the idea that different trees are built from different subsets of the training set and different subsets of the attributes; in the temporal case, moreover, they may differ by the subset of interval relations that are allowed in the learning (in [29], the problem of selecting subsets of relations in the learning phase, treated as feature selection problem, has been studied). We use interval temporal random forests in the same way as interval temporal decision trees, that is, to solve multivariate time series classification problems. Classification of time series is an active area of research: air quality control and prediction in climate science, prices and rates of inflation in economics, infectious diseases trends and spreading patterns in medicine, pronunciation of word signs in linguistics, sensor recordings of systems in aerospace engineering, among many others, are all problems that can be expressed in terms of time series classification. But the intrinsic versatility of time series allows us to act on less immediate applications, including, for example, interpreting *breath* and *cough* recordings as a medium for diagnosis of respiratory diseases. *COVID-19* is a respiratory disease onset by the virus *SARS-CoV2*, classified in 2019, which has caused a pandemic in the years 2020 and 2021; the current literature on this topic is huge, and spans every field from the medical, economical, sociological, up to and including applications of artificial intelligence. In early 2020, via an application and a website specifically built to this purpose, several cough and breath sound samples were recorded from anonymous volunteers, along with a small medical history and their declaration on positivity/negativitiy w.r.t. a recent COVID-19 test. Such recordings were used in [11] to train a classical, functional, adimensional classification system with the purpose of designing a prototype automatic

diagnosis system. The data were also made publicly available. In this paper, we use the same recordings by treating them as multivariate time series, and we test the interval temporal logic random forest model on them, serving two purposes: contributing to the fight against the pandemic with yet another system for automatic diagnosis and proving the effectiveness of our model.

This paper is organized as follows. In Section 2 we give some necessary background on time series and their classification, interval temporal logic, and sound-based diagnosis of respiratory diseases. In Section 3 we briefly recall interval temporal decision trees and introduce interval temporal random forests. Then, in Section 4 we discuss the data, the applied transformations, the experimental setup, and the results, before concluding.

2 Background

Time series and their classification. A *time series* T is a set of $n \geq 1$ variables that evolve over time, where each variable is an ordered collection of N_T numerical or categorical values described as follows:

$$T = \begin{cases} A_1 = a_{1,1}, & a_{1,2}, & \ldots, & a_{1,N_T} \\ A_2 = a_{2,1}, & a_{2,2}, & \ldots, & a_{2,N_T} \\ \quad \vdots & & & \\ A_n = a_{n,1}, & a_{n,2}, & \ldots, & a_{n,N_T}. \end{cases}$$

A time series is called *multivariate*, if $n > 1$; otherwise it is *univariate*. A univariate time series with categorical values is also known as a *time* (or *temporal*) *sequence*; we use the term *time series* to denote multivariate, mixed (numerical and categorical) set of temporal variables. Categorical values are fairly uncommon in time series, and typical temporal data sets are usually numerical. A *temporal data set* is a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of m temporal *instances* defined over a set of n attributes $\mathcal{A} = \{A_1, \ldots, A_n\}$, each of which is a univariate time series T having N_T points. A *categorical labeled* temporal data set is a temporal data set where the instances are associated to a *target variable* $\mathcal{C} = \{C_1, \ldots, C_l\}$, also known as *class variable*. In this paper, we assume that temporal data sets have no missing values, or that missing values are simply substituted by placeholders. Implicitly, we are also assuming that temporal attributes are sampled at the same granularity.

Time series are very versatile and we can use them to represent very diverse situations. For example, we may want to describe the clinical history of a patient; the set of numerical attributes may include *fever* and *level of pain*, which change along time on a scale, say, of minutes, and the problem may be to distinguish between *relapsing* and *non-relapsing* patients. Similarly, we can use time series to describe the behaviour of complex machines to which sensors are attached; they may measure *temperature* and *pressure* on a scale of seconds, and the problem may be to distinguish between machines that *need* from those which *do not need* preemptive maintenance. The *multivariate time series classification* is the problem of finding a formula or a set of formulas (i.e., *symbolic classification*), or a function (i.e., *functional classification*) that associates multivariate time series to classes. Several approaches for time series classification have been proposed in the literature, that span from purely functional ones [16, 24, 25, 30, 39], to distance-based ones [28], to symbolic ones [2, 3, 15, 41], to shapeled-based ones [12].

Interval temporal logic. While several different interval temporal logics have been proposed in the recent literature [18], *Halpern and Shoham's (HS)* [19] is certainly the formalism that has received the most attention, being the most natural modal logic for temporal intervals. From a logical point of view, HS and its fragments have been studied on the most important classes of linearly ordered sets, from the class of all linear orders, to the classes of linear orders that can be built on classical sets such as $\mathbb{N}, \mathbb{Q}$ and $\mathbb{R}$ [9, 10, 19]. Nevertheless, from the learning point of view, temporal (and static) data sets are finite, fully represented structures; therefore, we focus our attention on finite domains. Let $[N]$ be a finite, initial subset of $\mathbb{N}_+$ of cardinality $N > 1$, that is, $[N] = \{1, 2, \ldots, N\}$. A *strict interval* over $[N]$ is an ordered pair $[x, y]$, where $x, y \in [N]$ 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* [1]: the six relations R_A (*adjacent to*), R_L (*later than*), R_B (*begins*), R_E (*ends*), 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 modality $\langle \overline{X} \rangle$ corresponding to the inverse relation $R_{\overline{X}}$ of R_X. Now, let $\mathcal{X} = \{A, \overline{A}, L, \overline{L}, B, \overline{B}, E, \overline{E}, D, \overline{D}, O, \overline{O}\}$; Halpern and Shoham's interval temporal logic (HS) [19] is a multi-modal logic with formulas built from a finite, non-empty set $\mathcal{AP}$ of *atomic propositions* (also referred to as *proposition letters*), the propositional connectives $\vee$ and $\neg$, and a modality for each Allen's interval relation, and well-formed formulas of HS are generated by the grammar:

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

where $p \in \mathcal{AP}$ and $X \in \mathcal{X}$. The other propositional connectives and constants (e.g., $\psi_1 \wedge \psi_2 \equiv \neg\psi_1 \vee \neg\psi_2, \psi_1 \rightarrow \psi_2 \equiv \neg\psi_1 \vee \psi_2$ 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.

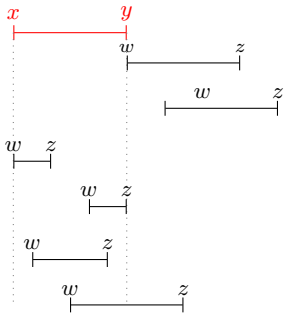
The strict semantics of HS is given in terms of *timelines* (or, more commonly, *interval models*) $T = \langle \mathbb{I}([N_T]), V \rangle$[1], where $[N_T] = \{1, 2, \ldots, N_T\}$ is a finite linear order, $\mathbb{I}([N_T])$ is the set of all *(strict) intervals* over $[N_T]$ with cardinality $N_T(N_T - 1)/2$, and V is a *valuation function* $V : \mathcal{AP} \rightarrow 2^{\mathbb{I}([N_T])}$ which assigns to every atomic proposition $p \in \mathcal{AP}$ 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 T, denoted by $T, [x, y] \Vdash \varphi$, is defined by structural induction on the complexity of formulas as follows:

$T, [x, y] \Vdash p$	if and only if	$[x, y] \in V(p)$, for each $p \in \mathcal{AP}$;
$T, [x, y] \Vdash \neg\psi$	if and only if	$T, [x, y] \nVdash \psi$ (i.e., it is not the case that $T, [x, y] \Vdash \psi$);
$T, [x, y] \Vdash \psi_1 \vee \psi_2$	if and only if	$T, [x, y] \Vdash \psi_1$ or $T, [x, y] \Vdash \psi_2$;
$T, [x, y] \Vdash \langle X \rangle \psi$	if and only if	there exists $[w, z]$ s.t. $[x, y] R_X [w, z]$ and $T, [w, z] \Vdash \psi$;

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

[1] We deliberately use the symbol T to indicate both a time series and a timeline.

Table 1 Allen's interval relations and HS modalities.

HS modality	Definition w.r.t. the interval structure	Example
$\langle A \rangle$ (after)	$[x,y]R_A[w,z] \Leftrightarrow y = w$	
$\langle L \rangle$ (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)	$[x,y]R_O[w,z] \Leftrightarrow x < w < y < z$	

Sounds and respiratory diseases. Human sounds, both audible (such as cough and breath) and non-audible (such as heartbeat) have been long considered useful tools for diagnosis. The recent use of sounds, and, in particular, cough, for the automatic, ML-based, diagnosis of respiratory diseases includes many examples (see [35] for a systematic review). Because of the current sanitary emergency, the case of COVID-19 has received a lot of attention. Very recent approaches include [20, 23, 26] and [11], from which the data used in this paper are borrowed. A common denominator to these approaches is their classical treatment of the dimensional data, such as breath and cough, that consists of extracting (even complex) numerical attributes from them, and then using such attributes to train classical classifier(s). Moreover, following the recent trends, the existing systems share the use of functional, complex classification learning systems (mainly deep neural networks). As it often happens, such black-box solutions tend to outperform symbolic approaches in terms of accuracy, but they give up, at the same time, the possibility of interpreting, discussing, and explaining the models. We, on the contrary, treat cough and breath as time series, and use native temporal methods to classify them. This results in an interesting compromise: our numerical performances are comparable to those of functional classifiers, and yet our models are explicit. While extracting a *theory of the sound of COVID-19* is outside the scope of this paper, the fact remains that we are able to pinpoint interval temporal logic formulas that in some way describe the differences between negative and positive cases; these formula may be even translated into diagnostic principles, and, in this particular case, audible sounds.

3 A Theory of Temporal Decision Trees and Forests

Temporal decision trees. Let $\mathcal{T}$ be a temporal data set described by n attributes $\{A_1, \ldots, A_n\}$. Given a time series $T \in \mathcal{T}$ and a time point t, we denote by $A(t)$ the value of A at the point t, and by $dom(A)$ the domain of A. A temporal data set entails a propositional alphabet $\mathcal{AP}$ defined as follows:

$$\mathcal{AP} = \{A \bowtie_{\sim\gamma} a \mid A \in \mathcal{A}, \bowtie \in \{<, \leq, =, \geq, >\}, \sim \in \{<, \leq, \geq, >\} \text{ and } a \in dom(A)\}.$$

The set $\mathcal{AP}$ is the natural generalization of the set of propositional letters that implicitly emerges in inductive processes from static data (e.g., *fever greater than* 38 *degrees*). The main difference between the two cases, propositional and temporal, is that in the latter case propositions in $\mathcal{AP}$ are given an interval semantics, that is, they are evaluated over intervals of time; this is a natural choice that depends from the fact that time series describe continuous

processes, in which evaluations based on punctual values have little sense. Intuitively, consider an interval of time $[x, y]$ and an attribute A that varies on it. We can ask the question $A \bowtie a$ over the entire interval, which is positively answered if *every value* of A in the interval $[x, y]$ respects the given constraint; but, to enhance an interval-based semantics we soften the meaning of $A \bowtie a$ using the expression $A \bowtie_{\sim\gamma} a$, with $\gamma \in (0, 1]$, that is interpreted as true if, assuming for example $\sim$ to be $\geq$, *at least the* γ *fraction* of the values in $[x, y]$ respect the given constraint. More formally, we say that:

$$T, [x, y] \Vdash A \bowtie_{\sim\gamma} a \quad \text{if and only if} \quad \frac{|\{z | x \leq z \leq y \text{ and } A(z) \bowtie a\}|}{y - x + 1} \sim \gamma.$$

In the particular case of propositional learning the set of propositional letter is complete, that is, for every letter p there exists a letter q that behaves as $\neg p$. The set $\mathcal{AP}$, as defined above, is complete as well in this sense; however, completeness is not necessary in the learning phase, and subsets of $\mathcal{AP}$ may be used to improve the experimental efficiency of the process. In the context of temporal decision trees and forests we do not ask if $A \bowtie_{\sim\gamma} a$ holds only in the *current interval* but also if *there exists an interval*, related to the current one, in which that holds. Thus, the language of temporal decision trees encompasses a set of *temporal existential decisions*:

$$\mathcal{S}_{\Diamond} = \{\langle X \rangle (A \bowtie_{\sim\gamma} a) \mid X \in \mathcal{X}, A \in \mathcal{A} \text{ and } a \in dom(A)\},$$

a set of *temporal universal decisions*:

$$\mathcal{S}_{\Box} = \{[X] (A \bowtie_{\sim\gamma} a) \mid X \in \mathcal{X}, A \in \mathcal{A} \text{ and } a \in dom(A)\},$$

and a set of *atemporal decisions*:

$$\mathcal{S}_{=} = \{A \bowtie_{\sim\gamma} a \mid A \in \mathcal{A} \text{ and } a \in dom(A)\}.$$

Together, they form a set of *temporal and atemporal decisions* $\mathcal{S}$, defined as:

$$\mathcal{S} = \mathcal{S}_{\Diamond} \cup \mathcal{S}_{\Box} \cup \mathcal{S}_{=}.$$

So, binary *temporal decision trees* τ are formulas of the following grammar:

$$\tau ::= (S_{=} \wedge \tau) \vee (\neg S_{=} \wedge \tau) \mid (S_{\Diamond} \wedge \tau) \vee (\neg S_{\Diamond} \wedge \tau) \mid C,$$

where $S_{=} \in \mathcal{S}_{=}$ is an atemporal decision, $S_{\Diamond} \in \mathcal{S}_{\Diamond}$ is a temporal existential decision, and $C \in \mathcal{C}$ is a class. Thus, a temporal decision tree is a rooted tree whose leaves are labeled with classes, and whose edges are labeled with temporal or atemporal decisions. We denote by $root(\tau)$ the *root* of τ, and we use $\ell_1, \ell_2, \ldots$ (resp., $\nu_1, \nu_2, \ldots$) to denote the *leaves* (resp., *nodes*, both leaf and non-leaf ones). Each non-leaf node ν of τ has a *left* (resp., *right*) *child* $L(\nu)$ (resp., $R(\nu)$) whose edge is decorated with $S \in \mathcal{S}_{=} \cup \mathcal{S}_{\Diamond}$ (resp., $\neg S \in \mathcal{S}_{=} \cup \mathcal{S}_{\Box}$), each non-root node ν has a *parent* $P(\nu)$, and each leaf ℓ is labeled with a class, denoted by $C(\ell)$. A *path* of *length* h between two nodes of τ is a finite sequence of nodes $\nu_h, \nu_{h-1}, \ldots, \nu_0$ such that $\nu_{i+1} = P(\nu_i)$, for each $i = 0, \ldots, h-1$; if ν_h is the root $root(\tau)$ and ν_0 is a leaf ℓ, then the path $root(\tau) \rightsquigarrow \ell$ is called *branch*. In general, a path of length h is *decorated* with h temporal and atemporal decisions on its edges, denoted by $\nu_h \overset{S_h}{\rightsquigarrow} \nu_{h-1} \overset{S_{h-1}}{\rightsquigarrow} \ldots \overset{S_1}{\rightsquigarrow} \nu_0$, where $S_i \in \mathcal{S}$, for each $i = 1, \ldots, h$.

In order to define the semantics of temporal decision trees, we need the notions of temporal path-formula, satisfiability of a temporal path-formula, and temporal data set splitting. A *temporal path-formula* $\varphi_{\nu_h \rightsquigarrow \nu_0}$ of a path $\nu_h \overset{S_h}{\rightsquigarrow} \nu_{h-1} \overset{S_{h-1}}{\rightsquigarrow} \ldots \overset{S_1}{\rightsquigarrow} \nu_0$, where $S_i \in \mathcal{S}$, in a temporal decision tree τ, is inductively defined on h:

- if $h = 0$, then $\varphi_{\nu_0 \rightsquigarrow \nu_0} = \top$;
- if $h > 0$, then let $\varphi_{\nu_{h-1} \rightsquigarrow \nu_0} = \xi'_{h-1} \wedge \ldots \wedge \xi'_1 \wedge \top$, and let us call ξ'_i *positive* if it has the form $\xi'_i = \langle X \rangle (A \bowtie_\gamma a \wedge \psi_i), \xi'_i = (\langle X \rangle (A \bowtie_\gamma a) \wedge \psi_i), \xi'_i = (A \bowtie_\gamma a \wedge \psi_i)$ or $\xi' = (A \bowtie_\gamma a \wedge \psi_i)$, with $X \in \mathcal{X}$, and *negative* otherwise. Then $\varphi_{\nu_h \rightsquigarrow \nu_0}$ is defined by cases:
  - if $\nu_{h-1} = left(\nu_h)$, then $\varphi_{\nu_h \rightsquigarrow \nu_0} = S_h \wedge \xi_{h-1} \wedge \ldots \wedge \xi_1 \wedge \top$, where, for $1 \leq i \leq h-1$:
    - $\xi_i = \langle X \rangle (A \bowtie_\gamma a \wedge \xi'_i)$, if $S_h = \langle X \rangle (A \bowtie_\gamma a)$ and ξ'_i is positive;
    - $\xi_i = (A \bowtie_\gamma a \wedge \xi'_i)$, if $S_h = A \bowtie_\gamma a$ and ξ'_i is positive;
    - $\xi_i = (\langle X \rangle (A \bowtie_\gamma a) \wedge [X](A \bowtie_\gamma a \rightarrow \xi'_i))$, if $S_h = \langle X \rangle (A \bowtie_\gamma a)$ and ξ'_i is negative;
    - $\xi_i = (A \bowtie_\gamma a \rightarrow \xi'_i)$, if $S_h = A \bowtie_\gamma a$ and ξ'_i is negative;
  - if $\nu_{h-1} = right(\nu_h)$, then $\varphi_{\nu_h \rightsquigarrow \nu_0} = (S_h) \wedge \xi_{h-1} \wedge \ldots \wedge \xi_1 \wedge \top$, where, for $1 \leq i \leq h-1$,
    - $\xi_i = \xi'_i$, if ξ'_i is positive;
    - $\xi_i = (S_h \wedge \xi'_i)$, if ξ'_i is negative.

Temporal path-formulas generalize their propositional counterpart, where propositional path-formulas are simply conjunctions of the decisions. Now, we need to define how they are actually interpreted. In the static case, from a data set $\mathcal{D}^\nu$ associated to a node ν of a (static) decision tree τ one computes immediately the two data sets $\mathcal{D}^{L(\nu)}$ and $\mathcal{D}^{R(\nu)}$ that are entailed by a propositional decision $S \in \mathcal{S}$. In the temporal case, however, this step requires a bigger effort. We start by assuming that each temporal instance T is *anchored* to a set of intervals in the set $\mathbb{I}([N_T]) \cup [0,1]$, denoted $T.refs$. At the beginning of the learning phase, $T.refs = \{[0,1]\}$ for every T, where $[0,1]$ is an external interval that we add to the domain of every time series, and that we interpret as a privileged observation point from which the learning takes place. Temporal decision tree learning is a *local* learning process; the local nature of decision trees does not transpire at the static level, but it becomes evident at the modal one. Every decision entails, potentially, new reference intervals for every instance of a data set. In particular, given a time series T with associated $T.refs$, and given a decision S, we can compute a set of *new reference intervals* $f(T.refs, S)$ as:

$$\{[w,z] \in \mathbb{I}([N_T]) \mid \exists [x,y] \in T.refs \wedge [x,y] R_X [w,z] \wedge T, [w,z] \Vdash A \bowtie_{\sim\gamma} a\}$$

if $S = \langle X \rangle (A \bowtie_{\sim\gamma} a)$, and as:

$$\{[w,z] \in T.refs \mid T, [w,z] \Vdash A \bowtie_{\sim\gamma} a\}$$

if $S = A \bowtie_{\sim\gamma} a$. When S is clear from the context, we use $T.refs'$ to denote $f(T.refs, S)$. For a decision $S \in \mathcal{S}_= \cup \mathcal{S}_\Diamond$, we use the notation $T \Vdash S$ or $T, T.refs \Vdash S$ (respectively, $T \Vdash \neg S$ or $T, T.refs \Vdash \neg S$) to identify the members of $\mathcal{T}^{L(\nu)}$ (respectively, $\mathcal{T}^{R(\nu)}$). The notion of a time series satisfying a decision allows us to discuss the instance semantics of a temporal decision tree. Given a temporal decision tree τ and a temporal instance $T \in \mathcal{T}$ anchored to $T.refs$ at $root(\tau)$, the *class assigned by* τ *to* T, denoted by $\tau(T, T.refs)$, is inductively defined as:

$$\begin{array}{lll} C & \text{if} & \tau = C, \\ \tau^L(T, T.refs') & \text{if} & \tau = (S \wedge \tau^L) \vee (\neg S \wedge \tau^R) \text{ and } T, T.refs \Vdash S; \\ \tau^R(T, T.refs) & \text{if} & \tau = (S \wedge \tau^L) \vee (\neg S \wedge \tau^R) \text{ and } T, T.refs \Vdash \neg S; \end{array}$$

where $S \in \mathcal{S}_= \cup \mathcal{S}_\Diamond$. Moreover, we denote by $\tau(T) = \tau(T, \{[0,1]\})$, where $[0,1]$ is the privileged observation point; we call $\tau(T)$ the *instance semantics* of τ. As a whole, a temporal decision tree is interpreted over a labeled data set $\mathcal{T}$ via the *dataset semantic* relation $\Vdash_\theta$, which generalizes $\models$ from single instances to data sets. The parameter θ can

represent any suitable measure of statistical performances of τ on $\mathcal{T}$, and it can be obtained by systematic application of the instance semantics to (sub)sets of $\mathcal{T}$; we simply say that $\mathcal{T}$ *θ-satisfies* τ, and denote it by:

$$\mathcal{T} \Vdash_{\theta} \tau.$$

Information-based learning. Propositional decision trees date back to Belson's [4] seminal work, based on which in [33] the authors proposed their innovative solution as an alternative to functional regression. The algorithm proposed in [32] is the first implementation of a decision tree for classification, but *CART* [8], *ID3* [36], and *C4.5* [37], are the most well-known. Because all share the same principles, they call can be generalized following the above schema, ending up in what we can generically call *information-based learning* of *temporal decision trees*. Information based learning is a general, greedy, sub-optimal approach to decision tree induction (optimal decision tree induction is knowingly NP-hard [22]). *Entropy-based learning* of (temporal) decision trees is a particular case of information-based learning, and the most common one. It works as follows. Let π_i be the fraction of instances labelled with class C_i in a dataset $\mathcal{T}$ with l distinct classes. Then, the *information conveyed* by $\mathcal{T}$ (or *entropy* of $\mathcal{T}$) is computed as:

$$Info(\mathcal{T}) = -\sum_{i=1}^{l} \pi_i \log \pi_i.$$

Intuitively, the entropy is inversely proportional to the purity degree of $\mathcal{T}$ with respect to the class values. In binary trees, *splitting*, which is the main greedy operation, is performed over a specific attribute A, a threshold value $a \in dom(A)$, a value γ, and the operators $\sim$ and $\bowtie$. Let $S(A, a, \gamma, \sim, \bowtie)$ be the decision entailed by $A, a, \gamma, \sim$, and $\bowtie$, and let $(\mathcal{T}_e, \mathcal{T}_u)$ be the partition of $\mathcal{T}$ entailed by $S(A, a, \gamma, \sim, \bowtie)$ (as defined above). The *splitting information* of $S = S(A, a, \gamma, \sim, \bowtie)$ is defined as:

$$InfoSplit(\mathcal{T}, S) = \frac{|\mathcal{T}_e|}{|\mathcal{T}|} Info(T_e) + \frac{|\mathcal{T}_u|}{|\mathcal{T}|} Info(T_u).$$

In this way, we can define the *entropy gain of a decision* as:

$$InfoGain(\mathcal{T}, S) = Info(\mathcal{T}) - InfoSplit(\mathcal{T}, S).$$

Existing open-source implementations of decision trees include, for example, the classes *DecisionTreeClassifier* in *Scikit-learn* [34] and *J48* in *WEKA* [40] learning frameworks, and the *DecisionTree* package [5] written in the *Julia* [6] programming language. In [38], an implementation of the algorithm for temporal decision tree learning based on the WEKA implementation of *C4.5*, that is, J48, and called *TemporalJ48* was presented. In recent years, the Julia programming language is becoming increasingly popular for scientific computing and, although the language is still young, there exists a stable Julia package for decision tree learning. Due to the performance gains that Julia, as a compiled language, enables, we developed an implementation of a temporal decision tree learning algorithm, called *TCART*, starting from the existing Julia package. Besides the language, *TCART* and *TemporalJ48* differ in implementation details only, and they can be considered the same algorithm to all intents and purposes.

Algorithm 1 High-level description of *TCART*.

function $TCART(\mathcal{T}, n_{att}, n_{lan})$**:**
  $\tau \leftarrow$ initialize an empty decision tree
  $Preprocess(\mathcal{T})$
  $root(\tau) \leftarrow Learn(\mathcal{T}, n_{att}, n_{lan})$
  return τ
end
function $Learn(\mathcal{T}, n_{att}, n_{lan})$**:**
  if *a stopping condition applies* **then return** $CreateLeafNode(\mathcal{T})$
  $S \leftarrow FindBestDecision(\mathcal{T}, n_{att}, n_{lan})$ ◁ `using` n_{att} `attr.,`n_{lan} `modal op.`
  $(\mathcal{T}_e, \mathcal{T}_u) \leftarrow Split(\mathcal{T}, S)$
  $\nu \leftarrow CreateNode(\mathcal{T})$
  $L(\nu) \leftarrow Learn(\mathcal{T}_e)$
  $R(\nu) \leftarrow Learn(\mathcal{T}_u)$
  return ν
end

Algorithm 2 High-level description of *TRF*.

function $TRF(\mathcal{T}, k, n_{att}, n_{lan})$**:**
  $Preprocess(\mathcal{T})$
  $\mathcal{F} \leftarrow \emptyset$
  foreach $i \in [1, \ldots, k]$ **do** $\mathcal{T}' \leftarrow SubsetSample(\mathcal{T}, \lceil \frac{k}{m} \rceil)$ ◁ `choose dataset`
  $\tau \leftarrow TCART(\mathcal{T}', n_{att}, n_{lan})$
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{\tau\}$
  return $\mathcal{F}$
end

Temporal random forests. In the propositional case, the generalization from single trees to forests of trees is relatively easy. The idea that underlies the so-called *random forests* model [7] is the following one: different trees can be learned from different subsets of the training set, using different subsets of attributes. Each tree is precisely a propositional decision tree; a *random forest classifier*, however, is a classifier whose instance semantics depends on many trees, and it is computed via some *voting* function. So, introducing temporal random forest models can be done in the same way. A *temporal random forest* is pair $(\mathcal{F}, v)$, where $\mathcal{F}$ is a collection of k temporal decision trees, that is, $\mathcal{F} = \{\tau_1, \ldots, \tau_k\}$, and $v : \mathcal{C}^k \rightarrow \mathcal{C}$ is *voting aggregation function* of all the unit votes of each temporal decision tree $\tau \in \mathcal{F}$.

Given a temporal random forest $(\mathcal{F} = \{\tau_1, \ldots, \tau_k\}, v)$ and a temporal instance $\mathcal{T} \in \mathcal{T}$, the *class assigned by* $\mathcal{F}$ *to* T, denoted by $\mathcal{F}(T)$, and called *instance semantics* of $\mathcal{F}$, is defined as:

$$v(\tau_1(T), \ldots, \tau_k(T)).$$

For a random forest $(\mathcal{F}, v)$ and a temporal data set $\mathcal{T}$, the notion $\mathcal{T} \Vdash_\theta \mathcal{F}$ is obtained, as in the case of the single tree, by the systematic application of the instance semantics to a certain (sub)set of the training dataset. Random forests differ from simple deterministic decision trees in many subtleties, all related to the learning algorithm. Such differences, along with the nature of the model, transform a purely symbolic method, such as decision trees, into a hybrid symbolic-functional approach. A first attempt towards random forests was made in [21], using the so-called *random subspace method*. Breiman's proposal [7], which

can be considered the standard approach to random forests, was later introduced in the *R* learning package [27]. Julia [6] incorporates a class to generalize trees into forests [5]; we used such a class to create a *temporal random forest (TRF)* learning algorithm. *TRF* is based on a generalized version of *TCART* that allows one to use, at each step, only n_{att} attributes and n_{lan} modal operators to find the best split, as shown in Algorithm 1; as a matter of fact, this is a *randomized* version of the interval temporal logic decision tree learning strategy, which degenerates into the deterministic version when $n_{att} = n$ and $n_{lan} = 12$. This solution generalizes the propositional case of random forests in which, at each step of building a tree, only a subset of attributes is used as shown in the high-level description of *TRF* in Algorithm 2. In terms of implementation, both *TCART* and *TRF* need special attention to the supporting data structures. As a matter of fact, both the propositional and the temporal versions of the information-based decision tree learning algorithm run in polynomial time w.r.t. the size of the dataset, but the overhead introduced in the temporal case can be quite relevant, because of the high number of decisions that can be taken at each split. To solve this issue, the function *Preprocess* entails, among other steps, building a hash table keyed on the tuple $(T, [x, y], X, A, a, \sim)$ for specific values of γ, that returns the truth value of the decision $\langle X \rangle (A \bowtie_{\sim\gamma} a)$. In this way, at learning time, checking the information conveyed by a decision takes (virtual) constant time plus the time to compute the information function. Interestingly enough, such a structure is particularly useful for *TRF*: as a matter of fact, it can be computed beforehand and then shared by all instances of *TCART* without recomputing, effectively improving the overall experimental complexity with respect to k independent executions of *TCART*.

4 Data and Experiments

Data and preparation. Breath and cough open data gathered in [11] have the following structure. The entire dataset is composed by 9986 samples, recorded by 6613 volunteers. Out of all volunteers, 235 declared to be positive for COVID-19. The subjects were quasi-normally distributed by age, with an average between 30 and 39 and a frequency curve slightly left-skewed towards younger ones; the data is not gender-balanced, with more than double as many male subjects than female ones. Besides recording sound samples, subjects were asked to fill in a very small clinical history, plus information about their geographical location. The static data (that is, history and location) is used both here and in [11] to create specific datasets, called *tasks*, from the original one. In particular, the location of the subject has been used to distinguish among those that, at the moment of the recording, were living in almost-COVID-free countries; by combining this information with the subjects' declaration concerning a COVID-test, the negative subjects could be considered reliable. Of the three different tasks considered in [11], we focused on the first one, which is the problem distinguishing between subjects who were declared positive to COVID-19, from non-positive subjects with a *clean medical history*, who have *never smoked*, have *no symptoms*, and live in countries in which the virus spread at that moment was very low (and thus who can reliably be considered negative to the virus). As a result, this task counts 141 positive and 298 negative instances. In [11] the tasks were declined into nine versions, which differ by how subjects are represented, that is, using only their cough sample, only their breath sample, or both (giving rise to three different problems) and how data are preprocessed. Unfortunately, by treating breath and cough as time series, we can describe each instance with only one multivariate time series at the time; this is not a limit when using classical, static methods. Therefore, here we can approach only two of the above versions (referred to as *cough* version

and *breath* version, respectively) of the problem. The raw audio data is encoded in the *Waveform Audio File (WAV)* format, and consists of a discrete sampling of the perceived sound pressure caused by (continuous) sound waves.

Despite the fact that this representation being already in the form of a time series, it is customary in audio signal processing to extract *spectral* representations of sounds, which facilitates their interpretation in terms of audio frequencies. To this end, we adopt a variation of a widespread representation technique, which goes under the name of *Mel-Frequency Cepstral Coefficients* (*MFCC*). MFCC, first proposed in [14], is still the preferred technique for extracting sensible spectral representations of audio data, and its use in machine learning has been fruitful for tackling hard AI tasks, such as speech recognition, music genre recognition, noise reduction, and audio similarity estimation. Computing the MFCC representation involves the following steps:

- **(i)** the raw audio is divided into (often overlapping) chunks of small size (e.g. $25ms$), and a *Discrete Fourier Transform (DFT)* is applied to each of the chunks, to produce a spectrogram of the sound at each chunk, that is, a continuous distribution of sound density across the frequency spectrum;
- **(ii)** the frequency spectrum is then warped according to a logarithmic function, which causes the frequency space to better reflect human ear perception of frequencies;
- **(iii)** a set of triangular band-pass filters is convolved across the frequency spectrum, discretizing it into a finite number of frequencies; finally,
- **(iv)** a *Discrete Cosine Transform (DCT)* is applied to the logarithm of the discretized spectrogram along the frequency axis, which compresses the spectral information at each point in time into a fixed number of coefficients.

This transformation does not modify the temporal ordering of the events; nevertheless, the classical approach at this point is to feed data to off-the-shelf classification methods which do not make use of such ordering (except, for example, recurrent neural networks). Moreover, the transformation does not preserve the spectral component, and the description of each time point is not directly related to sound frequencies. We applied MFCC up to step (*iii*), ultimately obtaining that each audio sample is represented as a time series with attributes describing the volume of different sound frequencies (called, here, *features*), and fed the resulting data to our temporal decision tree and random forest learning methods which are designed to learn natively from time series.

Test setting. We compare a temporal decision tree model (*TDT*), and two temporal random forest models with 50 and 100 trees, respectively. As for the random forest models, different choices for sampling the random subspace of decisions were explored:

- **(i)** considering the decisional space in its entirety;
- **(ii)** subsampling a low number of randomly-chosen features;
- **(iii)** subsampling a low number of randomly-chosen relations;
- **(iv)** subsampling a low number of randomly-chosen of features and relations.

After an initial pre-screening, we concluded that for this particular problem the best results are obtained by using either all relations and all features, or all relations and half of the features. While single decision trees experiments have been run with the standard pre-pruning setting (minimum entropy gain of 0.01 for a split to be meaningful, and a maximum entropy at each leaf of 0.6), random forest grow full trees.

As for the data, the chunk size and overlap for the DFT were fixed to the standard values of $15ms$ and $25ms$, respectively, and, since the processed series in this form present many points (100 for each second of recording), a moving average filter was applied, and the resulting series were capped at a maximum of 30 time points. Ultimately, different parametrizations were investigated by varying the number (f) of filters (frequencies), the size (w) and step (s) of the moving average, and by performing/not performing a *peak normalization* step prior to the MFCC phase. After the pre-screening phase, the moving average size and step have been fixed to 75 and 50 for the cough version, and to 45 and 30 for the breath version; moreover, $f \in \{40, 60\}$ for cough and $f \in \{20, 40\}$ for breath. In all cases we let $\bowtie$ be in $\{\leq, \geq\}$, $\sim$ be $\geq$, and $\gamma = 0.8$. Both versions count 439 audio samples, of which, as recalled above, 141 are positives and 298 are negatives; to minimize the bias, the dataset is, first, balanced by downsampling the majority class (thus deriving a subset of 282 samples), and, second, split into two (balanced) sets for training (80%) and test (20%). Since these two steps are randomized, this process is repeated 5 times (with seeds from 1 to 5), and we show the results of each repetition plus their average and standard deviation. Moreover, for a fixed training and test set, TRF is run 5 times with different seeds (again, with seeds from 1 to 5), and only their average is shown; so, for example, the first line in Tab. 2, is itself the result of averaging five runs on the same training/test splitting.

Results. The following questions are interesting: Is temporal random forest a suitable method to solve this problem? Which combination of parameters gives the best results? Are our best results comparable with the results obtained by standard techniques, especially in [11]? How our results can be interpreted?

As much as the suitability of our method is concerned, let us focus on Tab. 2, first, in which we show the detailed results for the case of temporal random forests; performances are all expressed in percentage. Each group of results is characterized by some parameters, and in particular different groups correspond to different numbers of features (20,40, or 60), the fact that cough or breath samples were used, and the fact that peak normalization was used (N), or not. The first general observation is that the average accuracy obtained by describing the subjects with their cough sample is 72.58 with a standard deviation of 0.72, while it is 69.40 with a standard deviation of 2.16 when subjects are described by their breath sample; this means that cough samples have a clear distinguishing power which is captured by temporal random forests. The standard *t-test* run on the populations of accuracies (averaged over the five runs) results in the two cases gives a *p-value* < 0.001, indicating that cough is clearly more informative than breath, for this particular task. In terms of sensitivity versus specificity, the best performance is obtained by a forest, learned on cough samples, with 100 trees and 60 features, using half of them in each tree: 71.57 and 74.86, respectively. This means that this model is able to correctly identify a real COVID-19-positive subject from his/her cough sample almost than 3 out of 4 times, with less than 28% of false negative cases. This particular configuration showed also a peak in performance during the first run (sensitivity: 81.43), possibly indicating that some samples are more informative than others. Comparing our results with [11] is not immediate: we could not use instances described by both cough and breath samples (and this combination resulted, in [11], more performing than using cough or breath only), we used a moving window technique for data preparation, and, more importantly, we limited our samples to the first 30 points of each sample, which corresponds to a length between 5 and 15 seconds (while the original samples have a length of around 30 seconds in most cases). Taking all this into account, however, we perform better than [11]: our best model improves the best model from [11] for the same data (task 1) by 2% in both precision and sensitivity.

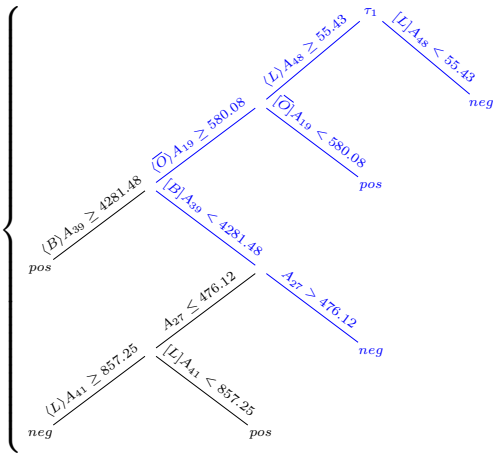
Table 2 Detailed results: random forests.

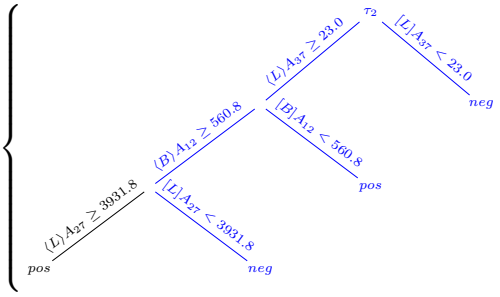
TRF		50 trees, all features				50 trees, half features				100 trees, all features				100 trees, half features			
		sens	spec	prec	acc	sens	spec	prec	acc	sens	spec	prec	acc	sens	spec	prec	acc
cough,40,N	1	80.0	85.0	84.53	82.5	78.57	83.57	82.93	81.07	79.29	88.57	87.77	83.93	80.0	85.0	84.3	82.5
	2	75.71	61.43	66.4	68.57	75.0	57.86	64.04	66.43	73.57	63.57	66.89	68.57	74.29	59.29	64.65	66.79
	3	58.57	78.57	73.44	68.57	55.0	81.43	74.65	68.21	57.86	79.29	73.83	68.57	57.86	80.71	75.15	69.29
	4	66.43	75.0	72.69	70.71	70.0	75.0	73.72	72.5	71.43	74.29	73.58	72.86	70.71	75.71	74.51	73.21
	5	65.0	68.57	67.42	66.79	70.71	77.14	75.69	73.93	68.57	69.29	69.08	68.93	69.29	76.43	74.65	72.86
	avg	**69.14**	**73.71**	**72.9**	**71.43**	**69.86**	**75.0**	**74.21**	**72.43**	**70.14**	**75.0**	**74.23**	**72.57**	**70.43**	**75.43**	**74.65**	**72.93**
	std	8.62	9.08	7.21	6.34	9.0	10.16	6.75	5.72	7.91	9.57	8.13	6.6	8.15	9.76	6.95	5.97
cough,60,N	1	79.29	83.57	82.94	81.43	80.71	83.57	83.19	82.14	80.71	84.29	83.69	82.5	78.57	83.57	82.9	81.07
	2	79.29	57.86	65.39	68.57	74.29	62.14	66.21	68.21	77.14	60.71	66.33	68.93	78.57	63.57	68.35	71.07
	3	57.86	78.57	73.18	68.21	55.0	77.86	71.42	66.43	57.14	80.71	74.76	68.93	58.57	80.71	75.36	69.64
	4	66.43	75.0	72.87	70.71	68.57	74.29	72.74	71.43	70.0	75.0	73.96	72.5	67.86	75.71	73.63	71.79
	5	67.86	70.0	69.36	68.93	69.29	73.57	72.49	71.43	66.43	68.57	67.89	67.5	69.29	74.29	73.02	71.79
	avg	**70.15**	**73.0**	**72.75**	**71.57**	**69.57**	**74.29**	**73.21**	**71.93**	**70.28**	**73.86**	**73.33**	**72.07**	**70.57**	**75.57**	**74.65**	**73.07**
	std	9.18	9.81	6.51	5.6	9.48	7.86	6.18	6.1	9.27	9.46	6.86	6.12	8.38	7.68	5.29	4.56
cough,40	1	76.43	80.0	79.89	78.21	73.57	74.29	74.15	73.93	75.0	80.71	79.67	77.86	77.14	80.0	79.49	78.57
	2	70.0	66.43	67.73	68.21	70.0	63.57	65.81	66.79	69.29	65.0	66.59	67.14	70.71	62.86	65.56	66.79
	3	59.29	84.29	79.15	71.79	57.14	82.86	77.36	70.0	57.14	84.29	78.55	70.71	58.57	87.14	82.14	72.86
	4	74.29	72.86	73.29	73.57	73.57	71.43	71.98	72.5	74.29	75.71	75.38	75.0	74.29	75.0	74.83	74.64
	5	70.71	81.43	79.27	76.07	72.14	78.57	77.25	75.36	71.43	77.14	75.87	74.29	72.14	80.0	78.33	76.07
	avg	**70.14**	**77.0**	**75.87**	**73.57**	**69.28**	**74.14**	**73.31**	**71.72**	**69.43**	**76.57**	**75.21**	**73.0**	**70.57**	**77.0**	**76.07**	**73.79**
	std	6.61	7.26	5.28	3.86	6.94	7.33	4.76	3.39	7.24	7.27	5.14	4.15	7.13	9.01	6.44	4.43
cough,60	1	75.71	78.57	77.94	77.14	75.0	76.43	76.16	75.71	76.43	77.86	77.65	77.14	81.43	76.43	77.63	78.93
	2	70.0	63.57	65.96	66.79	74.29	63.57	67.15	68.93	75.0	60.0	65.21	67.5	72.14	60.71	64.73	66.43
	3	54.29	81.43	74.56	67.86	57.86	83.57	78.08	70.71	56.43	80.71	74.52	68.57	59.29	80.0	74.95	69.64
	4	72.14	72.86	72.74	72.5	73.57	77.14	76.48	75.36	74.29	73.57	73.78	73.93	72.14	75.71	74.88	73.93
	5	72.14	79.29	77.85	75.71	70.71	80.0	77.88	75.36	72.86	80.71	79.16	76.79	72.86	81.43	79.7	77.14
	avg	**68.86**	**75.14**	**73.81**	**72.0**	**70.29**	**76.14**	**75.15**	**73.21**	**71.0**	**74.57**	**74.06**	**72.79**	**71.57**	**74.86**	**74.38**	**73.21**
	std	8.4	7.2	4.92	4.6	7.13	7.57	4.55	3.16	8.25	8.65	5.42	4.53	7.91	8.26	5.76	5.18
breath,20,N	1	77.14	68.57	71.65	72.86	77.86	69.29	71.92	73.57	75.71	71.43	72.66	73.57	77.14	68.57	71.22	72.86
	2	60.71	58.57	59.33	59.64	62.86	54.29	57.88	58.57	60.71	52.86	56.34	56.79	59.29	54.29	56.49	56.79
	3	60.0	75.0	70.95	67.5	63.57	76.43	73.68	70.0	65.0	70.0	68.5	67.5	62.86	76.43	73.07	69.64
	4	64.29	65.71	65.25	65.0	62.86	68.57	66.7	65.71	64.29	67.14	66.17	65.71	64.29	69.29	67.68	66.79
	5	70.71	80.71	78.99	75.71	67.14	75.71	73.57	71.43	72.14	77.86	76.98	75.0	72.14	76.43	75.52	74.29
	avg	66.57	69.71	69.23	68.14	66.86	68.86	68.75	67.86	67.57	67.86	68.13	67.71	67.14	69.0	68.8	68.07
	std	7.27	8.52	7.38	6.37	6.4	8.9	6.71	5.93	6.16	9.26	7.78	7.26	7.3	9.04	7.45	6.95
breath,40,N	1	75.71	58.57	64.75	67.14	72.86	64.29	67.19	68.57	77.86	60.71	66.54	69.29	74.29	62.14	66.28	68.21
	2	66.43	55.71	60.02	61.07	62.86	52.14	56.83	57.5	64.29	55.71	59.33	60.0	60.0	53.57	56.39	56.79
	3	62.86	73.57	70.46	68.21	60.0	73.57	70.07	66.79	63.57	70.71	68.5	67.14	62.14	70.71	67.99	66.43
	4	62.14	68.57	66.54	65.36	62.14	65.0	64.02	63.57	60.71	69.29	66.46	65.0	64.29	66.43	65.81	65.36
	5	68.57	81.43	78.65	75.0	69.29	80.0	77.73	74.64	70.0	80.71	78.75	75.36	70.0	81.43	79.2	75.71
	avg	67.14	67.57	68.08	67.36	65.43	67.0	67.17	66.21	67.29	67.43	67.92	67.36	66.14	66.86	67.13	66.5
	std	5.46	10.62	7.0	5.07	5.41	10.54	7.69	6.32	6.8	9.66	6.99	5.65	5.88	10.32	8.13	6.77
breath,20	1	72.14	77.86	76.59	75.0	70.71	75.71	74.54	73.21	72.86	77.86	76.72	75.36	72.14	76.43	75.45	74.29
	2	65.0	72.14	70.0	68.57	70.71	66.43	67.61	68.57	65.71	70.71	69.21	68.21	73.57	66.43	68.61	70.0
	3	65.0	70.0	68.48	67.5	62.86	70.71	68.35	66.79	68.57	70.71	70.18	69.64	68.57	68.57	68.56	68.57
	4	62.14	78.57	74.65	70.36	64.29	77.86	74.92	71.07	65.0	80.0	76.5	72.5	65.71	77.14	74.38	71.43
	5	62.86	83.57	79.52	73.21	67.86	82.86	80.26	75.36	62.86	85.71	81.63	74.29	66.43	86.43	83.32	76.43
	avg	65.43	76.43	73.85	70.93	67.29	74.71	73.14	71.0	67.0	77.0	74.85	72.0	**69.28**	**75.0**	**74.06**	**72.14**
	std	3.96	5.42	4.58	3.14	3.62	6.36	5.23	3.45	3.86	6.42	5.14	3.03	3.46	7.94	6.08	3.2
breath,40	1	69.29	80.0	77.73	74.64	71.43	79.29	77.79	75.36	71.43	77.86	76.47	74.64	72.14	77.86	76.54	75.0
	2	61.43	72.86	69.34	67.14	66.43	70.71	69.39	68.57	62.86	70.71	68.14	66.79	67.14	68.57	68.05	67.86
	3	65.0	71.43	69.35	68.21	65.0	69.29	67.86	67.14	67.86	68.57	68.32	68.21	67.86	67.86	67.85	67.86
	4	62.86	80.0	76.05	71.43	62.14	78.57	74.54	70.36	62.14	80.0	75.65	71.07	62.86	80.0	75.89	71.43
	5	62.14	87.86	84.06	75.0	64.29	85.0	81.23	74.64	64.29	88.57	85.43	76.43	66.43	87.86	84.83	77.14
	avg	64.14	78.43	75.31	71.28	65.86	76.57	74.16	71.21	65.72	77.14	74.8	71.43	67.29	76.43	74.63	71.86
	std	3.17	6.59	6.21	3.6	3.48	6.52	5.61	3.65	3.88	7.97	7.12	4.1	3.33	8.38	7.05	4.18

Single decision trees, whose results are shown in Tab. 3, are not as performing as forests of trees: the maximum averaged (only on different training/test splits: single tree learning is a deterministic process) accuracy reached with a single tree is 66%, which is less than the minimum accuracy reached with forests of trees. But single trees can be interpreted. In Tab. 3, right-hand side, each displayed tree has been learned in the conditions indicated by the arrow (so, for example, the topmost one has been learned by cough samples, with 60 features, no normalization, and with the subsampling obtained by seed 1), but in full training mode; in full training mode, the difference between the five runs of each configuration is limited to the downsampling phase. On each of the trees, we selected specific leaves with high *confidence* and *support* (in Tab. 3, right-hand side, these correspond to highlighted branches).

Table 3 Detailed results: single temporal decision trees.

TDT		sens	spec	prec	acc
cough,40,N	1	64.29	71.43	69.23	67.86
	2	64.29	35.71	50.0	50.0
	3	57.14	67.86	64.0	62.5
	4	67.86	50.0	57.58	58.93
	5	64.29	64.29	64.29	64.29
	avg	63.57	57.86	61.02	60.72
	std	3.91	14.81	7.42	6.8
cough,60,N	1	85.71	75.0	77.42	80.36
	2	71.43	39.29	54.05	55.36
	3	57.14	82.14	76.19	69.64
	4	57.14	67.86	64.0	62.5
	5	64.29	60.71	62.07	62.5
	avg	67.14	65.0	66.75	66.07
	std	11.95	16.44	9.92	9.45
cough,40	1	64.29	75.0	72.0	69.64
	2	71.43	50.0	58.82	60.71
	3	35.71	89.29	76.92	62.5
	4	60.71	50.0	54.84	55.36
	5	71.43	64.29	66.67	67.86
	avg	60.71	65.72	65.85	63.21
	std	14.73	16.87	9.11	5.73
cough,60	1	67.86	75.0	73.08	71.43
	2	71.43	46.43	57.14	58.93
	3	50.0	78.57	70.0	64.29
	4	60.71	53.57	56.67	57.14
	5	67.86	60.71	63.33	64.29
	avg	63.57	62.86	64.04	63.22
	std	8.53	13.74	7.41	5.59
breath,20,N	1	60.71	71.43	68.0	66.07
	2	60.71	50.0	54.84	55.36
	3	46.43	75.0	65.0	60.71
	4	53.57	53.57	53.57	53.57
	5	57.14	75.0	69.57	66.07
	avg	55.71	65.0	62.2	60.36
	std	5.97	12.22	7.49	5.84
breath,40,N	1	64.29	67.86	66.67	66.07
	2	64.29	39.29	51.43	51.79
	3	53.57	71.43	65.22	62.5
	4	64.29	57.14	60.0	60.71
	5	57.14	78.57	72.73	67.86
	avg	60.72	62.86	63.21	61.79
	std	5.05	15.28	8.0	6.26
breath,20	1	71.43	39.29	54.05	55.36
	2	78.57	39.29	56.41	58.93
	3	71.43	60.71	64.52	66.07
	4	64.29	75.0	72.0	69.64
	5	60.71	78.57	73.91	69.64
	avg	69.29	58.57	64.18	63.93
	std	6.96	18.83	8.93	6.49
breath,40	1	75.0	53.57	61.76	64.29
	2	78.57	35.71	55.0	57.14
	3	57.14	50.0	53.33	53.57
	4	71.43	71.43	71.43	71.43
	5	67.86	75.0	73.08	71.43
	avg	70.0	57.14	62.92	63.57
	std	8.22	16.17	9.11	8.15





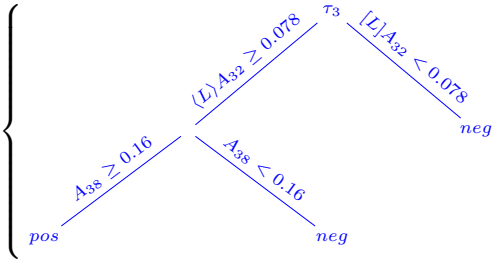


Table 4 Examples of extracted rules; all constants are scaled by a factor 10^3, and the symbols $\geq$ and $\leq$ (resp., their duals $<$ and $>$) denote $\geq_{\geq 0.8}$ and $\leq_{\geq 0.8}$ (resp., $<_{\geq 0.2}$ and $>_{\geq 0.2}$).

tree	rule	conf.	sup.
tree τ_1	$[L]A_{48} < 55.43 \Rightarrow neg$	0.74	0.37
	$\langle L\rangle A_{48} \geq 55.43 \wedge [L](A_{48} \geq 55.43 \rightarrow \overline{[O]}A_{19} < 580.08) \Rightarrow pos$	0.86	0.31
	$\langle L\rangle(A_{48} \geq 55.43 \wedge \overline{\langle O\rangle}A_{19} > 580.08 \wedge \overline{[O]}(A_{19} > 580.08 \rightarrow [B](A_{39} < 4281.48 \wedge A_{27} > 476.12))) \Rightarrow neg$	0.89	0.16
tree τ_2	$[L]A_{37} < 23.0 \Rightarrow neg$	0.73	0.40
	$\langle L\rangle A_{37} \geq 23.0 \wedge [L](A_{37} \geq 23.0 \rightarrow [B]A_{12} < 560.8) \Rightarrow pos$	0.83	0.39
	$\langle L\rangle(A_{37} \geq 23.0 \wedge \langle B\rangle(A_{12} \geq 560.8 \wedge [B](A_{12} \geq 560.8 \rightarrow [L]A_{27} < 3931.8))) \Rightarrow neg$	0.83	0.17
tree τ_3	$[L]A_{32} < 0.078 \Rightarrow neg$	0.77	0.36
	$\langle L\rangle A_{32} \geq 0.078 \wedge [L](A_{32} \geq 0.078 \rightarrow A_{38} < 0.16) \Rightarrow neg$	0.78	0.14
	$\langle L\rangle(A_{32} \geq 0.078 \wedge A_{38} \geq 0.016) \Rightarrow pos$	0.77	0.5

Each of these leaves can be interpreted as a classification rule of the type $\varphi \Rightarrow pos/neg$, where φ if the path-formula that can be extracted from the branch as explained in Section 3. The result of such an interpretation is in Tab. 4, in which every rule has been made explicit and its confidence and support displayed. Rules of this type can be visualized by synthesizing their model (in logical terms), and even converted into audible sound, which corresponds to *what should be heard (in a cough sample) to suspect a COVID-19 infection*. Obviously, implementing a tool for real-time screening based on rules as simple as these ones is much easier and direct than performing complex higher-dimensional matrix computations.

5 Conclusions

The ability of explaining the underlying theory that is extracted with machine learning methods is of uttermost importance, especially in medicine applications. Interpretability and explainability in learning are often synonym of a symbolic approach, which, in turn, should be based on logics that are able to capture the complexity of the phenomena. Modal symbolic learning offers classical learning tools enhanced with modal propositional logics that allow one to extract complex information from data; temporal symbolic learning is the specialization of modal symbolic learning to the case of temporal data and temporal logics. In the recent literature temporal decision trees, based on Halpen and Shoham's interval temporal logic HS have been proposed for learning from multivariate time series. In this paper we proposed a generalization of temporal decision trees to temporal random forest, following the path traced in the propositional case. In order to test our method, we applied it to the case of recognizing COVID-19-positive subjects from negative ones using a recording of cough/breath sample, interpreted as a multivariate time series. Not only this approach is completely innovative, but our performances are superior to those of classical methodologies applied to the same data, while allowing the interpretation of the results, and enabling the visualization, and even the transformation in audible sounds of the models that represent the distinguishing characteristics of a cough/breath sample of a positive subject. In abstract terms, such an ability could be useful to train medical personnel to recognize positive subjects, but also to develop automatic procedures that perform a (rough) screening, for example as a smartphone application.

This work is part of a larger project that aims to generalize symbolic learning methods with modal logics in a systematic way. As we have seen, not only this presents new challenges at the mathematical level, but also at the implementation one. Open problems at this moment include studying better interpretation techniques for temporal, and, in general, modal random forests, that do not require resorting to single trees, studying more efficient data structures that require less computational power, both in terms of space and time,

for symbolic learning, and completing the generalization of the whole range of symbolic methods, from decision trees to rule based classifiers, with deterministic and randomized learning methods, and with underlying logics both crisp and fuzzy. Exploring how to learn from *multi-frame* dimensional data is also on the agenda; this would allow us to improve our performances in the classification between COVID/non-COVID subjects, as we would be able to describe them using both their cough and breath samples. In the end, we want to test our methodologies on all three tasks from [11], to establish if there is an improvement in all cases, and if the rules that we are able to extract are indeed clinically useful in the fight against the virus.

References

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

2 S. Balakrishnan and D. Madigan. Decision trees for functional variables. In *Proc. of the 6th International Conference on Data Mining*, pages 798–802, 2006.

3 E. Bartocci, L. Bortolussi, and G. Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Proc. of the 12th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 8711 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 2014.

4 W.A. Belson. A technique for studying the effects of television broadcast. *Journal of the Royal Statistical Society*, 5(3):195–202, 1956.

5 J. Bezanson, A. Edelman, S. Karpinski, and V.B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

6 J. Bezanson, A. Edelman, S. Karpinski, and V.B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

7 L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

8 L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth Publishing Company, 1984.

9 D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Decidability and complexity of the fragments of the modal logic of Allen's relations over the rationals. *Information and Computation*, 266:97–125, 2019.

10 D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theoretical Computers Science*, 560:269–291, 2014.

11 C. Brown, J. Chauhan, A. Grammenos, J. Han, A. Hasthanasombat, D. Spathis, T. Xia, P. Cicuta, and C. Mascolo. Exploring automatic diagnosis of COVID-19 from crowdsourced respiratory sound data. In *Proc. of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3474–3484, 2020.

12 A. Brunello, E. Marzano, A. Montanari, and G. Sciavicco. J48SS: A novel decision tree approach for the handling of sequential and time series data. *Computers*, 8(1):21, 2019.

13 A. Brunello, G. Sciavicco, and I.E. Stan. Interval temporal logic decision tree learning. In *Proc. of the 16th European Conference on Logics in Artificial Intelligences*, volume 11468 of *Lecture Notes in Computer Science*, pages 778–793. Springer, 2019.

14 S.B. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, 1980.

15 J.J. Rodríguez Diez, C. Alonso González, and H. Boström. Boosting interval based literals. *Intelligent Data Analysis*, 5(3):245–262, 2001.

16 H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.A. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

17 J.H. Friedman and B.E. Popescu. Predictive learning via rule esambles. *The Annals of Applied Statistics*, 2(3), 2008.

18 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.

19 J.Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.

20 J. Han, K. Qian, M. Song, Z. Yang, Z. Ren, S. Liu, J. Liu, H. Zheng, W. Ji, T. Koike, X. Li, Z. Zhang, Y. Yamamoto, and B. Schuller. An early study on intelligent analysis of speech under covid-19: Severity, sleep quality, fatigue, and anxiety. In *Proc. of the Conference INTERSPEECH*, pages 1–5, 2020.

21 T.K. Ho. Random decision forests. In *Proc. of the 3rd International Conference on Document Analysis and Recognition*, pages 278–282, 1995.

22 L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

23 A. Imran, I. Posokhova, H.N. Qureshi, U. Masood, M. Sajid Riaz, K. Ali, C.N. John, I. Hussain, and M. Nabeel. AI4COVID-19: AI enabled preliminary diagnosis for COVID-19 from cough samples via an app. *Informatics in Medicine Unlocked*, 20:1–14, 2020.

24 Y. Kakizawa, R.H. Shumway, and M. Taniguchi. Discrimination and clustering for multivariate time series. *Journal of the American Statistical Association*, 93(441):328–340, 1998.

25 M. Kudo, J. Toyama, and M. Shimbo. Multidimensional curve classification using Passing-Through regions. *Pattern Recognition Letters*, 20(11):1103–1111, 1999.

26 J. Laguarta, F. Hueto, and B. Subirana. COVID-19 artificial intelligence diagnosis using only cough recordings. *IEEE Open Journal of Engineering in Medicine and Biology*, 1:275–281, 2020.

27 A. Liaw and M. Wiener. Classification and regression by RandomForest. *R News*, 2(3):18–22, 2002.

28 J. Lines and A.J. Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592, 2015.

29 E. Lucena-Sánchez and and I.E. Stan G. Sciavicco. Feature and language selection in temporal symbolic regression for interpretable air quality modelling. *Algorithms*, 14(3):1–17, 2021.

30 P. Malhotra, V. TV, L. Vig, P. Agarwal, and G. M. Shroff. Timenet: Pre-trained deep recurrent neural network for time series classification. In *Proc. of the 25th European Symposium on Artificial Neural Networks*, pages 607–612, 2017.

31 N. Meinshausen. Node harvest. *The Annals of Applied Statistics*, 4(4), 2010.

32 R. Messenger and L. Mandell. A modal search technique for predictive nominal scale multivariate analysis. *Journal of the American Statistical Association*, 67(340):768–772, 1972.

33 J.N. Morgan and J.A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of American Statistical Association*, 58(302):415–434, 1963.

34 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

35 R.X.A. Pramono, S. Bowyer, and E. Rodriguez-Villegas. Automatic adventitious respiratory sound analysis: A systematic review. *Plos One*, 12(5):1–43, 2017.

36 J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

37 J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

38 G. Sciavicco and I.E. Stan. Knowledge Extraction with Interval Temporal Logic Decision Trees. In *Proc. of the 27th International Symposium on Temporal Representation and Reasoning*, volume 178 of *Leibniz International Proceedings in Informatics*, pages 9:1–9:16, 2020.

39 I. Sutskever, O. Vinyals, and Q.V. Le. Sequence to sequence learning with neural networks. In *Proc. of the 28th Conference on Neural Information Processing Systems*, pages 3104–3112, 2014.

40 I.H. Witten, E. Frank, and M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edition, 2017.

41 Y. Yamada, E. Suzuki, H. Yokoi, and K. Takabayashi. Decision-tree induction from time-series data based on a standard-example split test. In *Proc. of the 12th International Conference on Machine Learning*, page 840–847. AAAI Press, 2003.

Past Matters: Supporting LTL+Past in the BLACK Satisfiability Checker

Luca Geatti
University of Udine, Italy
Fondazione Bruno Kessler, Trento, Italy

Nicola Gigante
Free University of Bozen-Bolzano, Italy

Angelo Montanari
University of Udine, Italy

Gabriele Venturato
University of Udine, Italy
KU Leuven, Belgium

Abstract

LTL+Past is the extension of Linear Temporal Logic (LTL) supporting *past* temporal operators. The addition of the past does not add expressive power, but does increase the usability of the language both in formal verification and in artificial intelligence, e.g., in the context of multi-agent systems. In this paper, we add the support of past operators to BLACK, a satisfiability checker for LTL based on a SAT encoding of a tree-shaped tableau system. We implement two ways of supporting the past in the tool. The first one is an equisatisfiable translation that removes the past operators, obtaining a future-only formula that can be solved with the original LTL engine. The second one extends the SAT encoding of the underlying tableau to directly support the tableau rules that deal with past operators. We describe both approaches and experimentally compare the two between themselves and with the nuXmv model checker, obtaining promising results.

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

Keywords and phrases SAT, LTL, LTL+Past, Tableaux

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.8

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria. Nicola Gigante acknowledges the partial support of the *TOTA* project "*Temporal Ontologies and Tableaux Algorithms*". Gabriele Venturato acknowledges the partial support of the KU Leuven Research Fund (C14/18/062).

1 Introduction

Linear Temporal Logic (LTL) [20] is the de-facto standard temporal specification language in many areas including *formal verification* [8] and *artificial intelligence* [11]. Satisfiability checking, that is, deciding whether a given formula admits a model, is a particularly important problem because of its wide range of applications, and one of the first that have been studied [23, 26]. Many techniques and tools have been developed to solve it, ranging from tableau systems [1, 17, 22, 26] to reduction to model checking [5], from temporal resolution [9, 10, 14] to automata-theoretic techniques [16].

The B*ounded* L*TL* s*A*t*isfiability* C*hec*K*er*, BLACK for short, is a recently developed tool [12] that solves the satisfiability checking problem for LTL by providing a SAT encoding of the one-pass and tree-shaped tableau method for LTL proposed by Reynolds [22]. In an iterative procedure, the tree-shaped tableau is symbolically explored in a breadth-first way through a SAT encoding of its branches of depth at most k, for increasing values of k. The tool proved to be competitive with other state-of-the-art approaches [12].

In this paper, we extend BLACK to support LTL+Past, which enriches LTL with *past operators*, i.e. temporal modalities that talk about the past of the current time point. Although past operators do not add expressive power to the language, interestingly LTL+Past is exponentially more succinct than LTL, and it is thus able to express some useful properties in a more compact and natural way [19]. LTL+Past has been investigated from both theoretical and algorithmic viewpoints [13, 17, 18, 21], in the areas of formal verification [7] and artificial intelligence, e.g., in the context of multi-agent systems (see [3] and references therein).

We present and compare two different methods to support LTL+Past in BLACK.

The first one is a Tseitin-style [25] translation procedure that, given an LTL+Past formula, generates an *equisatisfiable* LTL formula that can be solved by the original LTL engine of BLACK, or, in principle, by any other tool for LTL satisfiability checking. Since the resulting formula is equisatisfiable, and not equivalent, to the original one, it can avoid the exponential blowup (it causes only a linear size increase). The core idea behind the translation is folklore, but, to the best of our knowledge, this is the first time it is explicitly worked out for LTL, implemented in a tool, and experimentally compared with other approaches.

The second method extends the SAT encoding [12] of the tableau system for LTL described in [13,22] with the ability of directly handling *past* temporal operators. The resulting encoding successfully supports the claim given in [12, 13] that the one-pass and tree-shaped tableau system for LTL, together with its SAT encoding, can be easily extended to other temporal logics. Last but not least, our encoding for the support of past operators is much simpler than similar methods, like, for instance, the one based on *virtual unrollings* by Latvala et al. [15].

We make a comparison of the two methods, showing that the direct encoding outperforms the translation, although both methods show comparable performance. This makes the direct encoding the preferred way of handling the past in BLACK, but shows that the translation can be a useful and effective preprocessing step to deal with past operators in tools that do not support them natively. Since there is not a standardized set of benchmarks involving past operators in the literature, we introduce some novel sets of formulas for the sake of this comparison.

Finally, we compare both solutions with the nuXmv model checker, which is the only widely available tool, as far as we know, that directly supports past operators. The results are promising, showing BLACK to be competitive.

The paper is organized as follows. Section 2 introduces LTL+Past and Reynolds' tableau [22]. Then, Section 3 and Section 4 describe the translation and the direct encoding, respectively. Finally, Section 5 describes the results of the experimental comparison, and Section 6 concludes the paper.

2 Preliminaries

Let Σ be an alphabet of proposition letters. The syntax of an LTL+Past formula ϕ over Σ can be defined as follows:

$$\begin{aligned} \phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid & \qquad \textit{Boolean} \text{ connectives} \\ \mathsf{X}\phi \mid \phi_1 \,\mathcal{U}\, \phi_2 \mid \phi_1 \,\mathcal{R}\, \phi_2 \mid & \qquad \textit{future} \text{ temporal operators} \\ \mathsf{Y}\phi \mid \mathsf{Z}\phi \mid \phi_1 \,\mathcal{S}\, \phi_2 \mid \phi_1 \,\mathcal{T}\, \phi_2 & \qquad \textit{past} \text{ temporal operators} \end{aligned}$$

where $p \in \Sigma$ and ϕ, ϕ_1, and ϕ_2 are LTL+Past formulas. LTL is the fragment that only uses Boolean connectives and future operators. We denote by LTL[Σ] and LTL+Past[Σ], respectively, the sets of LTL and LTL+Past formulas built over the alphabet Σ. Standard shorthands and derived operators are also available, such as $\top \equiv p \vee \neg p$, for some $p \in \Sigma$, $\bot \equiv \neg\top$, $\mathsf{F}\phi \equiv \top \,\mathcal{U}\, \phi$, $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$, $\mathsf{O}\phi \equiv \top \,\mathcal{S}\, \phi$, $\mathsf{H}\phi \equiv \neg\mathsf{O}\neg\phi$.

LTL+Past is interpreted over infinite *state sequences* $\overline{\sigma} \in (2^{\Sigma})^{\omega}$. Given a state sequence $\overline{\sigma} \in (2^{\Sigma})^{\omega}$, the *satisfaction* of a formula ϕ by $\overline{\sigma}$ at a time point $i \geq 0$, denoted as $\overline{\sigma}, i \models \phi$, is defined as follows:

1. $\overline{\sigma}, i \models p$ iff $p \in \sigma_i$;
2. $\overline{\sigma}, i \models \neg\phi$ iff $\overline{\sigma}, i \not\models \phi$;
3. $\overline{\sigma}, i \models \phi_1 \vee \phi_2$ iff $\overline{\sigma}, i \models \phi_1$ or $\overline{\sigma}, i \models \phi_2$;
4. $\overline{\sigma}, i \models \phi_1 \wedge \phi_2$ iff $\overline{\sigma}, i \models \phi_1$ and $\overline{\sigma}, i \models \phi_2$;
5. $\overline{\sigma}, i \models \mathsf{X}\phi$ iff $\overline{\sigma}, i+1 \models \phi$;
6. $\overline{\sigma}, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\overline{\sigma}, i-1 \models \phi$;
7. $\overline{\sigma}, i \models \mathsf{Z}\phi$ iff either $i = 0$ or $\overline{\sigma}, i-1 \models \phi$;
8. $\overline{\sigma}, i \models \phi_1 \,\mathcal{U}\, \phi_2$ iff there exists $j \geq i$ such that $\overline{\sigma}, j \models \phi_2$, and $\overline{\sigma}, k \models \phi_1$ for all k, with $i \leq k < j$;
9. $\overline{\sigma}, i \models \phi_1 \,\mathcal{S}\, \phi_2$ iff there exists $j \leq i$ such that $\overline{\sigma}, j \models \phi_2$, and $\overline{\sigma}, k \models \phi_1$ for all k, with $j < k \leq i$;
10. $\overline{\sigma}, i \models \phi_1 \,\mathcal{R}\, \phi_2$ iff either $\overline{\sigma}, j \models \phi_2$ for all $j \geq i$, or there exists $k \geq i$ such that $\overline{\sigma}, k \models \phi_1$ and $\overline{\sigma}, j \models \phi_2$ for all $i \leq j \leq k$;
11. $\overline{\sigma}, i \models \phi_1 \,\mathcal{T}\, \phi_2$ iff either $\overline{\sigma}, j \models \phi_2$ for all $0 \leq j \leq i$, or there exists $k \leq i$ such that $\overline{\sigma}, k \models \phi_1$ and $\overline{\sigma}, j \models \phi_2$ for all $i \geq j \geq k$

We say that a state sequence $\overline{\sigma}$ satisfies ϕ, written $\overline{\sigma} \models \phi$, if $\overline{\sigma}, 0 \models \phi$. Observe that the $\wedge$ connective, the *release* operator ($\phi_1 \,\mathcal{R}\, \phi_2$), the *triggered* operator ($\phi_1 \,\mathcal{T}\, \phi_2$), and the *weak yesterday* operator ($\mathsf{Z}\phi$) can be defined in terms of the $\vee$ connective, the *until* operator ($\phi_1 \,\mathcal{U}\, \phi_2$), the *since* operator ($\phi_1 \,\mathcal{S}\, \phi_2$), and the *yesterday* operator ($\mathsf{Y}\phi$), respectively, but here we consider them as primitive operators since this allows us to put any formula into *negation normal form* (NNF), which will be useful later. Moreover, note that state sequences have a definite starting point, hence *the past is bounded*, and we need to distinguish between the *yesterday* operator ($\mathsf{Y}\phi$, ϕ holds at the previous state) and the *weak yesterday* operator ($\mathsf{Z}\phi$, ϕ holds at the previous state, if it exists) as opposed to a single *tomorrow* operator ($\mathsf{X}\phi$, ϕ holds at the next state).

The notion of *closure* of a formula will be useful later.

▶ **Definition 1** (Closure of an LTL+Past formula). *Let ψ be an* LTL+Past *formula built over Σ. The* closure *of ψ is the smallest set of formulas $\mathcal{C}(\psi)$ satisfying the following properties:*

1. $\psi \in \mathcal{C}(\psi)$;
2. *for each sub-formula ψ' of ψ, $\psi' \in \mathcal{C}(\psi)$;*
3. *for each $p \in \Sigma$, $p \in \mathcal{C}(\psi)$ if and only if $\neg p \in \mathcal{C}(\psi)$;*
4. *if $\psi_1 \,\mathcal{U}\, \phi_2 \in \mathcal{C}(\psi)$, then $\mathsf{X}(\psi_1 \,\mathcal{U}\, \psi_2) \in \mathcal{C}(\psi)$;*
5. *if $\phi_1 \,\mathcal{R}\, \psi_2 \in \mathcal{C}(\psi)$, then $\mathsf{X}(\psi_1 \,\mathcal{R}\, \phi_2) \in \mathcal{C}(\psi)$;*
6. *if $\psi_1 \,\mathcal{S}\, \psi_2 \in \mathcal{C}(\psi)$, then $\mathsf{Y}(\psi_1 \,\mathcal{S}\, \psi_2) \in \mathcal{C}(\psi)$;*
7. *if $\psi_1 \,\mathcal{T}\, \psi_2 \in \mathcal{C}(\psi)$, then $\mathsf{Z}(\psi_1 \,\mathcal{T}\, \psi_2) \in \mathcal{C}(\psi)$.*

It is worth pointing out that item 3 of Definition 1 only applies to proposition letters because formulas are assumed to be in NNF.

The one-pass and tree-shaped tableau for LTL+Past

Let us now briefly describe the tableau system for LTL+Past introduced by Geatti et al. [13], which will be used as the basis for the direct encoding discussed in Section 4. It extends the tableau system for LTL by Reynolds [22]. The latter has the distinctive features of being *tree-shaped*, as opposed to standard graph-shaped LTL tableaux, e.g., [17], and *one-pass*, since a single pass is sufficient to either accept or reject a given branch.

Table 1 Tableau expansion rules. When a formula ϕ of one of the types shown in the table is found in the label Γ of a node u, one or two children u' and u'' are created with the same label as u, but replacing ϕ by the formulas from $\Gamma_1(\phi)$ and $\Gamma_2(\phi)$, respectively.

Rule	$\phi \in \Gamma$	$\Gamma_1(\phi)$	$\Gamma_2(\phi)$
DISJUNCTION	$\alpha \vee \beta$	$\{\alpha\}$	$\{\beta\}$
CONJUNCTION	$\alpha \wedge \beta$	$\{\alpha, \beta\}$	
UNTIL	$\alpha \,\mathcal{U}\, \beta$	$\{\beta\}$	$\{\alpha, \mathsf{X}(\alpha \,\mathcal{U}\, \beta)\}$
SINCE	$\alpha \,\mathcal{S}\, \beta$	$\{\beta\}$	$\{\alpha, \mathsf{Y}(\alpha \,\mathcal{S}\, \beta)\}$
RELEASE	$\alpha \,\mathcal{R}\, \beta$	$\{\alpha, \beta\}$	$\{\beta, \mathsf{X}(\alpha \,\mathcal{R}\, \beta)\}$
TRIGGERED	$\alpha \,\mathcal{T}\, \beta$	$\{\alpha, \beta\}$	$\{\beta, \mathsf{Z}(\alpha \,\mathcal{T}\, \beta)\}$

For ease of exposition, *w.l.o.g.* we assume formulas to be in NNF. A tableau for a formula ϕ is a tree where each node u is labeled by a set of formulas $\Gamma(u)$, with the root u_0 labeled with $\Gamma(u_0) = \{\phi\}$. At each step, a set of rules is applied to a leaf, until all branches have been either *accepted* or *rejected*. Each rule either adds one or more children to the current leaf or either accept or reject the current branch. Given a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, the sequence of nodes $\langle u_i, \ldots, u_j \rangle$, for some $0 \leq i \leq j \leq n$, is denoted by $\overline{u}_{[i,j]}$.

At each step, the selected node is subject to a number of *expansion rules*, that select a formula of the label and expand it according to its semantics, as reported in Table 1. Each expansion rule creates one or two children depending on the selected formula. After repeated applications of the expansion rules, a node that only contains *elementary* formulas, that is, propositions, *tomorrow*, *yesterday*, or *weak yesterday* formulas, is obtained (*poised* node). Elementary formulas of the form $\mathsf{X}(\phi_1 \,\mathcal{U}\, \phi_2)$ are called X-*eventualities*. An X-eventuality is a formula that, intuitively, requests something to be fulfilled later. Given an X-eventuality $\phi \equiv \mathsf{X}(\phi_1 \,\mathcal{U}\, \phi_2)$, ϕ is said to be *fulfilled* in a node u if $\phi_2 \in \Gamma(u)$.

The tableau advances through time by making *temporal steps*. To do that, the following rules are applied to poised nodes.

STEP A child u_{n+1} is added to u_n, with:

$$\Gamma(u_{n+1}) = \{\alpha \mid \mathsf{X}\alpha \in \Gamma(u_n)\}$$

FORECAST Let

$$G_n = \left\{ \alpha \in \mathcal{C}(\phi) \;\middle|\; \begin{array}{l} \mathsf{Y}\alpha \in \mathcal{C}(\psi) \text{ or} \\ \mathsf{Z}\alpha \in \mathcal{C}(\psi) \text{ for some } \psi \in \Gamma(u_n) \end{array} \right\}$$

For each subset $G'_n \subseteq G_n$ (including $\varnothing$), a child u'_n is added to u_n such that $\Gamma(u'_n) = \Gamma(u_n) \cup G'_n$. This is done once and only once before every application of the STEP rule.

The STEP rule advances the construction of the current branch to the subsequent temporal state. The FORECAST is essential to the well-functioning of the rule dealing with *past*, as it adds a number of branches that nondeterministically guess formulas that may be needed to fulfill past requests coming from future states. For details on the FORECAST rule, we refer the reader to Geatti et al. [13].

Since the STEP rule is not applied to all the poised nodes (to some of which the FORECAST rule is applied instead), we need the following definition.

▶ **Definition 2** (Step node). *In a complete tableau for an* LTL+Past *formula, a poised node u_n is a* step node *if it is either a poised leaf or a poised node to which the STEP rule was applied.*

Given a node u, we define u^* as the closest ancestor of u that is child of a step node, if any. $\Gamma^*(u)$ is the union of the labels of the nodes from u to u^* or to the root, if u^* does not exist.

Before applying the STEP rule though, poised nodes are subject to the application of a few *termination rules*, that is, rules that decide whether the construction has to continue or the current branch has to be either rejected or accepted. Given a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, with u_n a step node, the termination rules are the following.

CONTRADICTION If $\{p, \neg p\} \subseteq \Gamma(u_n)$, for some $p \in \Sigma$, then $\overline{u}$ is *rejected*.

EMPTY If $\Gamma(u_n) = \varnothing$, then $\overline{u}$ is *accepted*.

YESTERDAY If $\mathsf{Y}\alpha \in \Gamma(u_n)$, then the branch $\overline{u}$ is *rejected* if either u_n^* does not exist or $Y_n \not\subseteq \Gamma^*(u_n^*)$, where $Y_n = \{\psi \mid \mathsf{Y}\psi \in \Gamma(u_n)\}$.

W-YESTERDAY If $\mathsf{Z}\alpha \in \Gamma(u_n)$, then $\overline{u}$ is *rejected* if u_n^* exists and $Z_n \not\subseteq \Gamma^*(u_n^*)$, where $Z_n = \{\psi \mid \mathsf{Z}\psi \in \Gamma(u_n)\}$.

LOOP If there exists a position $i < n$ such that $\Gamma(u_i) = \Gamma(u_n)$ and all the X-eventualities requested in u_i are fulfilled in $\overline{u}_{[i+1 \ldots n]}$, then $\overline{u}$ is *accepted*.

PRUNE If there exist two positions i and j such that $i < j \leq n$, $\Gamma(u_i) = \Gamma(u_j) = \Gamma(u_n)$, and all the X-eventualities requested in these nodes which are fulfilled in $\overline{u}_{[j+1 \ldots n]}$ are also fulfilled in $\overline{u}_{[i+1 \ldots j]}$, then $\overline{u}$ is *rejected*.

Intuitively, the CONTRADICTION, YESTERDAY, and W-YESTERDAY rules reject branches that contain some contradiction, either a propositional one or because of some unfulfilled past request. The EMPTY rule accepts a branch devoid of contradictions where there is nothing left to do, while the LOOP one accepts a looping branch where all the X-eventualities are proposed again and fulfilled at every repetition of the loop. Finally, the PRUNE rule, which was the main novelty of the system when introduced by Reynolds [22], rejects a branch that, otherwise, is going to be infinitely unrolled because of an X-eventuality impossible to fulfill.

3 The translation

In this section, we define a procedure which takes as input an LTL+Past formula, and returns as output an *equisatisfiable* LTL formula. The increase in size is only linear, thus avoiding the exponential blowup of the worst-case complexity of the translation of an LTL+Past formula into an equivalent (and not simply an equisatisfiable) LTL one [19]. The idea behind the translation is simple, but this is the first time, as far as we know, that it has been actually implemented and experimentally compared with other approaches to support past operators.

The key idea is to replace past subformulas by fresh proposition letters that are forced to replicate the semantics of past operators with ad-hoc axioms. Even though the produced formula is not equivalent to the original one, the proposed translation procedure allows us to easily recover a model of the original formula, if it is satisfiable in the first place, by simply discarding the additional proposition letters. Note that we will define the translation *without* assuming formulas to be in NNF.

To begin, we define two functions, τ and θ, which are the building blocks of the translation. The function τ replaces past formulas with the corresponding placeholder proposition letters, while θ enriches the formula with the axioms to force those letters to behave correctly.

Let Σ be the alphabet of proposition letters of the input formula ϕ. The alphabet of the output formula is $\Sigma_+ = \Sigma \cup \Sigma_{past}$, where Σ_{past} is the set of fresh proposition letters introduced by τ. We recursively define the function $\tau : \mathsf{LTL{+}Past}[\Sigma] \to \mathsf{LTL}[\Sigma_+]$ as follows:

$$
\begin{aligned}
\tau(p) &= p && \text{where } p \in \Sigma \\
\tau(\neg\phi) &= \neg\tau(\phi) \\
\tau(\mathsf{X}\phi) &= \mathsf{X}\tau(\phi) \\
\tau(\phi_1 \otimes \phi_2) &= \tau(\phi_1) \otimes \tau(\phi_2) && \text{where } \otimes \in \{\wedge, \vee, \mathcal{U}, \mathcal{R}\} \\
\tau(\mathsf{Y}\phi) &= p_{\mathsf{Y}\tau(\phi)} \\
\tau(\mathsf{Z}\phi) &= p_{\mathsf{Z}\tau(\phi)} \\
\tau(\phi_1 \,\mathcal{S}\, \phi_2) &= p_{\tau(\phi_1)\mathcal{S}\tau(\phi_2)} \\
\tau(\phi_1 \,\mathcal{T}\, \phi_2) &= \tau(\neg(\neg\phi_1 \,\mathcal{S}\, \neg\phi_2))
\end{aligned}
$$

The function $\Theta : \mathsf{LTL}[\Sigma_+] \to 2^{\mathsf{LTL}[\Sigma_{++}]}$, where $\Sigma_{++} = \Sigma_+ \cup \{p_{\mathsf{Y}p_\psi} \mid p_\psi \in \Sigma_{past}, \psi \equiv \phi_1 \,\mathcal{S}\, \phi_2\}$, produces a set of formulas which gives the appropriate semantics to the proposition letters in Σ_{past}. It is defined as follows:

$$
\begin{aligned}
\Theta(p) &= \varnothing && \text{where } p \in \Sigma \\
\Theta(\otimes\ \phi) &= \Theta(\phi) && \text{where } \otimes \in \{\neg, \mathsf{X}\} \\
\Theta(\phi_1 \otimes \phi_2) &= \Theta(\phi_1) \cup \Theta(\phi_2) && \text{where } \otimes \in \{\wedge, \vee, \mathcal{U}, \mathcal{R}\} \\
\Theta(p_{\mathsf{Y}\phi}) &= \{Y_{p_{\mathsf{Y}\phi}}\} \cup \Theta(\phi) && \text{where } Y_{p_{\mathsf{Y}\phi}} \equiv \neg p_{\mathsf{Y}\phi} \wedge \mathsf{G}(\mathsf{X}p_{\mathsf{Y}\phi} \leftrightarrow \phi) \\
\Theta(p_{\mathsf{Z}\phi}) &= \{Z_{p_{\mathsf{Z}\phi}}\} \cup \Theta(\phi) && \text{where } Z_{p_{\mathsf{Z}\phi}} \equiv \ \ p_{\mathsf{Z}\phi} \wedge \mathsf{G}(\mathsf{X}p_{\mathsf{Z}\phi} \leftrightarrow \phi) \\
\Theta(p_\psi) &= \{S_{p_\psi}, Y_{p_{\mathsf{Y}p_\psi}}\} \cup \Theta(\phi_1) \cup \Theta(\phi_2)
\end{aligned}
$$

where $\psi \equiv \phi_1 \,\mathcal{S}\, \phi_2$ and

$$S_{p_{(\phi_1 \mathcal{S} \phi_2)}} \equiv \mathsf{G}\Big(p_{(\phi_1 \mathcal{S} \phi_2)} \leftrightarrow \big(\phi_2 \vee (\phi_1 \wedge p_{\mathsf{Y}p_{(\phi_1 \mathcal{S} \phi_2)}})\big)\Big)$$

The first three cases are pretty straightforward. The last three, which are the core of the whole translation, are, instead, more involved.

As for the *yesterday* operator, we state that if the argument ϕ of the yesterday operator is true in a certain state, then in the next state $\mathsf{Y}\phi$ is true. Note that we force the proposition letter $p_{\mathsf{Y}\phi}$ to be *false* in the initial state, because $\mathsf{Y}\phi$ cannot be true in that state. The *weak yesterday* operator behaves almost the same. However, according to its semantics, $\mathsf{Z}\phi$ is always true at the initial state, and thus we constrain the proposition letter $p_{\mathsf{Z}\phi}$ to hold at the initial state.

Let us consider now the *since* operator. By exploiting its semantics and the corresponding *expansion rule* defined for the tableau in Table 1, we say – in the scope of the *always* operator – that $\phi_1 \,\mathcal{S}\, \phi_2$ is true at a certain state if and only if, at that state, either ϕ_2 is true, or ϕ_1 is true and $\phi_1 \,\mathcal{S}\, \phi_2$ was true at the previous state. However, this is not enough to capture the intended semantics, because we have introduced a new symbol, that is, $p_{\mathsf{Y}p_\psi}$, and we need to force the (yesterday) semantics also for it. This can be done exactly as before.

The function $\theta : \mathsf{LTL}[\Sigma_+] \to \mathsf{LTL}[\Sigma_{++}]$ wraps a formula with the semantics of the additional proposition letters. It is formally defined as follows:

$$\theta(\phi) = \phi \wedge \bigwedge_{\psi \,\in\, \Theta(\phi)} \psi$$

The proposed translation procedure is simply the *function composition* of θ and τ.

▶ **Definition 3** (RemovePast). *The function* RemovePast : $\mathsf{LTL{+}Past}[\Sigma] \to \mathsf{LTL}[\Sigma_{++}]$ *is defined as follows:* RemovePast $= \theta \circ \tau$.

It is possible to prove that the above translation results in an *equisatisfiable* formula.

▶ **Theorem 4.** *Let ϕ be an* LTL+Past *formula. Then,* RemovePast(ϕ) *is an* LTL *formula* equisatisfiable *with ϕ.*

Proof. Since the input formula ϕ is finite, the procedure RemovePast(ϕ) always terminates. We have to show that RemovePast(ϕ) is satisfiable if and only if ϕ is satisfiable.

($\leftarrow$) Let $\overline{\sigma}$ be the model which satisfies ϕ, i.e. $\overline{\sigma} \models \phi$. Let us show that there exists $\overline{\sigma}'$ such that $\overline{\sigma}' \models$ RemovePast(ϕ), where, for each $i \geq 0$ and each sub-formula ψ of ϕ, $\overline{\sigma}'$ is defined as follows.

1. $p \in \sigma'_i$ iff $\overline{\sigma} \models_i p$ for $p \in \Sigma$
2. $\tau(\mathsf{Y}\psi) \in \sigma'_i$ iff $i > 0$ and $\overline{\sigma}' \models_{i-1} \tau(\psi)$
3. $\tau(\mathsf{Z}\psi) \in \sigma'_i$ iff either $i = 0$ or $\overline{\sigma}' \models_{i-1} \tau(\psi)$
4. $\tau(\psi_1 \mathcal{S} \psi_2) \in \sigma'_i$ iff there exists $0 \leq j \leq i$ such that $\overline{\sigma}' \models_j \tau(\psi_2)$, and $\overline{\sigma}' \models_k \tau(\psi_1)$ for all k, with $j < k \leq i$

By the definition above, we can prove by induction on the structure of ϕ, that $\overline{\sigma}' \models \tau(\phi)$. Moreover, by Item 2 we have that, for each *yesterday* sub-formula ψ of ϕ, $\overline{\sigma}' \models Y_{\tau(\psi)}$, because with Table 2 we force a step-wise consistency between a *yesterday* formula and its request at the previous state, in $\overline{\sigma}'$, which is exactly what is stated by $Y_{\tau(\psi)}$. Similarly, by Item 2 and Item 2, we also have that $\overline{\sigma}' \models Z_{\tau(\psi)}$ and $\overline{\sigma}' \models S_{\tau(\psi)}$ for, respectively, each *weak yesterday* and each *since* sub-formula ψ of ϕ. This means that $\overline{\sigma}'$ satisfies the conjunction of the three formulas, hence $\overline{\sigma}' \models \bigwedge_{\psi \in \Theta(\tau(\phi))} \psi$. Thus, $\overline{\sigma}' \models \theta(\tau(\phi))$. This allows us to conclude that $\overline{\sigma}' \models$ RemovePast(ϕ).

($\rightarrow$) Given a model $\overline{\sigma}'$ such that $\overline{\sigma}' \models$ RemovePast(ϕ), we can easily build $\overline{\sigma}$ for ϕ by setting that $p \in \sigma_i$ iff $\overline{\sigma}' \models_i p$ for all $p \in \Sigma$. By induction on the structure of ϕ, using the semantics of past operators stated by $Y_{\tau(\psi)}$, $Z_{\tau(\psi)}$, $S_{\tau(\psi)}$, we can prove that $\overline{\sigma} \models \phi$. ◀

4 The direct encoding

The BLACK satisfiability checker is based on an iterative procedure that symbolically explores the tableau tree breadth-first by means of a SAT encoding of the tableau branches up to a given depth k, for increasing values of k. The satisfiability checking procedure employed by BLACK is reported in Algorithm 1 [12]. The three formulas $[\![\phi]\!]^k$, $|\phi|^k$, and $|\phi|^k_T$ encode different rules of the tableau. This section shows how to extend them to support past operators by encoding the tableau rules recalled in Section 2.

Let us start with some notation. Let ϕ be an LTL+Past formula in NNF over the alphabet Σ. We define the following sets of formulas:

$$\mathsf{XR} = \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{tomorrow} \text{ formula}\}$$
$$\mathsf{YR} = \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{yesterday} \text{ formula}\}$$
$$\mathsf{ZR} = \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is a } \textit{weak yesterday} \text{ formula}\}$$
$$\mathsf{XEV} = \{\psi \in \mathcal{C}(\phi) \mid \psi \text{ is an } \mathsf{X}\textit{-eventuality}\}$$

The three encoding formulas are defined over an extended alphabet $\overline{\Sigma}$, which includes:

1. any proposition letter from the original alphabet Σ;
2. the set $\{p_\psi \mid \psi \in \mathsf{XR}, \mathsf{YR}, \mathsf{ZR}\}$, that is, the set of all the *grounded* X-, Y-, and Z-*requests*;
3. a *stepped* version p^k of all the proposition letters defined in items 1 and 2, with $k \in \mathbb{N}$ and p^0 identified as p.

Algorithm 1 BLACK's main procedure [12].

```
1: procedure BLACK(φ)
2:     k ← 0
3:     while True do
4:         if [[φ]]^k is UNSAT then
5:             return φ is UNSAT
6:         end if
7:         if |φ|^k is SAT then
8:             return φ is SAT
9:         end if
10:        if |φ|_T^k is UNSAT then
11:            return φ is UNSAT
12:        end if
13:        k ← k + 1
14:    end while
15: end procedure
```

Intuitively, different stepped versions of the same proposition letter p are used to represent the value of p at different states. Thus, when p^i holds, it means that p holds at i-th step node of the branch, i.e. the i-th state of the model.

Moreover, given $\psi \in \mathcal{C}(\phi)$, we denote by ψ_G the formula in which all the X-, Y-, and Z-*requests* are replaced by their grounded version. Similarly, given $\psi \in \mathcal{C}(\phi)$, we denote by ψ^k the formula in which all proposition letters are replaced by their k stepped version. We write ψ_G^k to denote $(\psi_G)^k$.

The formula $[\![\phi]\!]^k$ is called the *k-unraveling* of ϕ, and encodes the expansion of the tableau tree. To define it, we need to encode the expansion rules of Table 1.

▶ **Definition 5** (Stepped Normal Form). *Given an* LTL+Past *formula* ϕ *in NNF, its* stepped normal form*, denoted by* $\mathrm{snf}(\phi)$*, is defined as follows:*

$$
\begin{aligned}
\mathrm{snf}(\ell) &= \ell && \text{where } \ell \in \{p, \neg p\}, \text{ for } p \in \Sigma \\
\mathrm{snf}(\otimes\, \phi_1) &= \otimes\, \phi_1 && \text{where } \otimes \in \{\mathsf{X}, \mathsf{Y}, \mathsf{Z}\} \\
\mathrm{snf}(\phi_1 \otimes \phi_2) &= \mathrm{snf}(\phi_1) \otimes \mathrm{snf}(\phi_2) && \text{where } \otimes \in \{\wedge, \vee\} \\
\mathrm{snf}(\phi_1\, \mathcal{U}\, \phi_2) &= \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge \mathsf{X}(\phi_1\, \mathcal{U}\, \phi_2)) \\
\mathrm{snf}(\phi_1\, \mathcal{R}\, \phi_2) &= \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee \mathsf{X}(\phi_1\, \mathcal{R}\, \phi_2)) \\
\mathrm{snf}(\phi_1\, \mathcal{S}\, \phi_2) &= \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge \mathsf{Y}(\phi_1\, \mathcal{S}\, \phi_2)) \\
\mathrm{snf}(\phi_1\, \mathcal{T}\, \phi_2) &= \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee \mathsf{Z}(\phi_1\, \mathcal{T}\, \phi_2))
\end{aligned}
$$

The stepped normal form is the extension to past operators of the *next normal form* used by Geatti et al. [13]. It can be noted how it follows the expansion rules of each operator in Table 1. We can now define the k-unraveling of ϕ recursively as follows:

$$
\begin{aligned}
[\![\phi]\!]^0 &= \mathrm{snf}(\phi)_G \wedge \bigwedge_{\psi \in \mathsf{YR}} \neg\psi_G \wedge \bigwedge_{\psi \in \mathsf{ZR}} \psi_G \\
[\![\phi]\!]^{k+1} &= [\![\phi]\!]^k \wedge S_k \wedge Y_k \wedge Z_k
\end{aligned}
$$

$$
S_k \equiv \bigwedge_{\mathsf{X}\alpha \in \mathsf{XR}} \left((\mathsf{X}\alpha)_G^k \leftrightarrow \mathrm{snf}(\alpha)_G^{k+1} \right),
$$

$$Y_k \equiv \bigwedge_{\mathsf{Y}\alpha \in \mathsf{YR}} \left((\mathsf{Y}\alpha)_G^{k+1} \leftrightarrow \mathrm{snf}(\alpha)_G^k \right), \qquad Z_k \equiv \bigwedge_{\mathsf{Z}\alpha \in \mathsf{ZR}} \left((\mathsf{Z}\alpha)_G^{k+1} \leftrightarrow \mathrm{snf}(\alpha)_G^k \right)$$

The S_k, Y_k and Z_k formulas encode, respectively, the STEP, YESTERDAY, and W-YESTERDAY rules of the tableau, while the base case of the 0-unraveling ensures that *yesterday* formulas are false and *weak yesterday* formulas are true at the first state. The CONTRADICTION rule of the tableau is implicitly encoded in the fact that only satisfying assignments of the formula are considered. Note that the FORECAST rule as well does not need to be explicitly encoded: the intrinsic nondeterminism of the SAT solving process accounts for the nondeterministic choices implemented by the rule.

Intuitively, if $[\![\phi]\!]^k$ is unsatisfiable, all the branches of the tableau for ϕ are rejected before $k+1$ steps.

▶ **Lemma 6.** *Let ϕ be an* LTL+Past *formula. Then, $[\![\phi]\!]^k$ is unsatisfiable if and only if all the branches of the complete tableau for ϕ are crossed by the CONTRADICTION or (W-)YESTERDAY rules and contain at most $k+1$ step nodes.*

Proof. We prove the contrapositive, i.e. that $[\![\phi]\!]^k$ is satisfiable if and only if the complete tableau for ϕ has at least a branch that is either accepted, crossed by PRUNE, or longer than $k+1$ step nodes. To do that we establish a connection between truth assignments of $[\![\phi]\!]^k$ and suitable branches of the tableau.

From branches to assignments. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be a branch that is either accepted, crossed by PRUNE, or longer than $k+1$ step nodes. Let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. We define a truth assignment ν for $[\![\phi]\!]^k$ as follows. Note that $[\![\phi]\!]^k$ contains stepped propositions from p^0 until p^k for any given p, so we need at most $k+1$ step nodes from $\overline{u}$, which however can be shorter if it is accepted or crossed by the PRUNE rule. Hence, let us define $\ell = \min\{m, k\}$. Moreover, let us define p_U to be p if $p \in \Sigma$, and to be ψ if $p = \psi_G$ for some X-, Y-, or Z-request ψ, i.e. $(\cdot)_U$ is the inverse of the $(\cdot)_G$ operation. Then, for $0 \leq i \leq \ell$, we set $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$. Then, we complete the assignments for positions $m < j \leq k+1$ (if any) as follows:

1. if the branch has been accepted by the EMPTY rule, all the other positions $j > m$ can be filled arbitrarily;
2. if the branch has been accepted by the LOOP or crossed by the PRUNE rule, then there is a position w such that $\Gamma(\pi_w) = \Gamma(\pi_m)$. Then we continue filling the truth assignment considering the successor of π_w as a successor of π_m.

It can be verified that the truth assignment so constructed satisfies $[\![\phi]\!]^k$.

From assignments to branches. Let ν be a truth assignment for $[\![\phi]\!]^k$. We use ν as a guide to navigate the tableau tree to find a suitable branch which is either accepted, crossed by PRUNE, or has more than $k+1$ step nodes. To do that we build a sequence of branch prefixes $\overline{u}_i = \langle u_0, \ldots, u_i \rangle$ where at each step we obtain $\overline{u}_{i+1}$ by choosing u_{i+1} among the children of u_i, until we find a leaf or we reach $k+1$ step nodes. During the descent, we build a partial function $J : \mathbb{N} \to \mathbb{N}$ that maps positions j in $\overline{u}_i$ to indexes $J(j)$ such that for all ψ it holds that $\psi \in \Gamma(u_j)$ if and only if $\nu \models \mathrm{snf}(\psi)_G^{J(j)}$, i.e. we build a relationship between positions in the branch and steps in ν. As the base case, we put $\overline{u}_0 = \langle u_0 \rangle$ and $J(0) = 0$ so that the invariant holds since $\Gamma(u_0) = \{\phi\}$ and $\nu \models \mathrm{snf}(\phi)_G^0$ by the definition of $[\![\phi]\!]^k$. Then, depending on the rule that was applied to u_i, we choose u_{i+1} among its children as follows:

1. if the STEP rule has been applied to u_i, then there is a unique child that we choose as u_{i+1}, and we define $J(i+1) = J(i) + 1$. Now, for all $\mathsf{X}\alpha \in \Gamma(u_i)$, we have $\alpha \in \Gamma(u_{i+1})$ by construction of the tableau. Note that $\mathrm{snf}(\mathsf{X}\alpha) = \mathsf{X}\alpha$, hence we know by construction that $\nu \models (\mathsf{X}\alpha)_{\mathsf{G}}^{J(j)}$. Then, by definition of $[\![\phi]\!]^k$, we know that $\nu \models \mathrm{snf}(\alpha)_G^{J(j)+1}$, i.e.

$\nu \models \mathrm{snf}(\alpha)_G^{J(i+1)}$. On the other direction, if $\nu \models \mathrm{snf}(\alpha)_G^{J(i+1)}$, then by definition of $[\![\phi]\!]^k$ we have $\nu \models (\mathsf{X}\alpha)_G^{J(i)}$, hence $\nu \models \mathrm{snf}(\mathsf{X}\alpha)_G^{J(i)}$, hence $\mathsf{X}\alpha \in \Gamma(u_i)$, so by construction of the tableau we have $\alpha \in \Gamma(u_{i+1})$. Hence the invariant holds.

2. if the FORECAST rule has been applied to u_i, then there are n children $\{u_i^1, \ldots, u_i^n\}$ such that $\Gamma(u_i) \subseteq \Gamma(u_i^m)$ for all $1 \leq m \leq n$. Now, we set $J(i+1) = J(i)$ and we choose u_{i+1} as a child u_i^m with a label $\Gamma(u_i^m)$ such that for any ψ we have $\psi \in \Gamma(u_{i+1})$ if and only if $\nu \models \mathrm{snf}(\psi)_G^{J(i+1)}$. Note that at least one such child exists, because at least one child has the same label as u_i. Thus the invariant holds by construction.
3. if an *expansion rule* has been applied to u_i, then there are one or two children. In both cases, we set $J(i+1) = J(i)$. Then:
   a. if there is one child, then it is chosen as u_{i+1}. In this case, the rule is the CONJUNCTION rule and has been applied to a formula $\psi \equiv \psi_1 \wedge \psi_2$, hence $\psi_1, \psi_2 \in \Gamma(u_{i+1})$. By construction we know that $\nu \models \mathrm{snf}(\psi)_G^{J(i)}$, hence $\nu \models \mathrm{snf}(\psi)_G^{J(i+1)}$. Now, note that $\mathrm{snf}(\psi_1 \wedge \psi_2) = \mathrm{snf}(\psi_1) \wedge \mathrm{snf}(\psi_2)$, so it holds that $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ and $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$. On the other direction, if $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ and $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$ we know that $\nu \models \mathrm{snf}(\psi_1 \wedge \psi_2)_G^{J(i+1)}$ hence $\nu \models \mathrm{snf}(\psi_1 \wedge \psi_2)_G^{J(i)}$, hence by construction we have $\psi_1 \wedge \psi_2 \in \Gamma(u_i)$ and so we have $\psi_1, \psi_2 \in \Gamma(u_i)$, hence the invariant holds.
   b. if there are two children u_i' and u_i'', then let us suppose the rule applied is the DISJUNCTION rule. Similar arguments will hold for the other rules. In this case, the rule has been applied to a formula $\psi \equiv \psi_1 \vee \psi_2$, hence $\psi_1 \in \Gamma(u_i')$ and $\psi_2 \in \Gamma(u_i'')$. We know $\nu \models \mathrm{snf}(\psi)_G^{J(i)}$, hence $\nu \models \mathrm{snf}(\psi)_G^{J(i+1)}$. Since $\mathrm{snf}(\psi_1 \vee \psi_2) = \mathrm{snf}(\psi_1) \vee \mathrm{snf}(\psi_2)$, it holds that either $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ or $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$. Now, we choose u_{i+1} accordingly, so to respect the invariant. Note that if both nodes are eligible, which one is chosen does not matter. The other direction of the invariant also holds, since if either $\nu \models \mathrm{snf}(\psi_1)_G^{J(i+1)}$ or $\nu \models \mathrm{snf}(\psi_2)_G^{J(i+1)}$, then $\nu \models \mathrm{snf}(\psi_1)_G^{J(i)}$ or $\nu \models \mathrm{snf}(\psi_2)_G^{J(i)}$, so $\nu \models \mathrm{snf}(\psi_1 \vee \psi_2)_G^{J(i)}$, hence $\psi_1 \vee \psi_2 \in \Gamma(u_i)$, hence either $\psi_1 \in \Gamma(u_{i+1})$ or $\psi_2 \in \Gamma(u_{i+1})$.

Let $\overline{u} = \langle u_0, \ldots, u_i \rangle$ be the branch prefix constructed as above, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_n \rangle$ be the sequence of its step nodes. As mentioned, the descent stops when π_n is a leaf or when $n = k+1$. Note in any case that $u_i = \pi_n$. In case we find a leaf, note that it cannot have been crossed by the CONTRADICTION rule. Otherwise, we would have $\{p, \neg p\} \subseteq \Gamma(u_i)$, which would mean $\nu \models p^{J(i)}$ and $\nu \models \neg p^{J(i)}$, which is not possible. Moreover, it cannot have been crossed by the YESTERDAY rule, since that would mean there is some $\mathsf{Y}\alpha \in \Gamma(\pi_n)$ with $\alpha \notin \Gamma^*(\pi_{n-1})$. But, we know that $\nu \models \mathrm{snf}(\mathsf{Y}\alpha)_G^{J(i)}$, hence $\nu \models (\mathsf{Y}\alpha)_G^{J(i)}$ since $\mathrm{snf}(\mathsf{Y}\alpha) = \mathsf{Y}\alpha$. Then, by definition of $[\![\phi]\!]^k$, we know that $\nu \models \mathrm{snf}(\alpha)_G^{J(i)-1}$. Since $u_i = \pi_n$ is a step node, $J(i) - 1 = J(j)$ for some j such that $u_j = \pi_{n-1}$, hence $\nu \models \mathrm{snf}(\alpha)_G^{J(j)}$, and by construction we know that $\alpha \in \Gamma(u_j)$, which conflicts with the hypothesis that the YESTERDAY rule crossed the branch. With a similar argument, we can see that it cannot have been crossed by the W-YESTERDAY rule neither. Hence we found a branch which is either longer than $k+1$ step nodes, or have been accepted, or have been crossed by the PRUNE rule. ◀

The formula $|\phi|^k$ is called the *base encoding* of ϕ and, in addition to the k-unraveling, includes the encoding of the EMPTY and LOOP rules, i.e. the rules that can accept branches. The formula is defined as:

$$|\phi|^k \equiv [\![\phi]\!]^k \wedge (E_k \vee L_k)$$

where the E_k formula encodes the EMPTY rule and is defined as follows:

$$E_k \equiv \bigwedge_{\psi \in \mathsf{XR}} \neg \psi_G^k$$

and the L_k formula encodes the LOOP rule and is defined as:

$$L_k \equiv \bigvee_{l=0}^{k-1} ({}_lR_k \wedge {}_lF_k)$$

where

$${}_lR_k \equiv \bigwedge_{\psi \in \mathsf{XR} \cup \mathsf{YR} \cup \mathsf{ZR}} \left(\psi_G^l \leftrightarrow \psi_G^k \right), \text{ and}$$

$${}_lF_k \equiv \bigwedge_{\substack{\psi \in \mathsf{XEV} \\ \psi \equiv \mathsf{X}(\psi_1 \mathcal{U} \psi_2)}} \left(\psi_G^k \rightarrow \bigvee_{i=l+1}^{k} \mathrm{snf}(\psi_2)_G^i \right) .$$

Intuitively, ${}_lR_k$ encodes the presence of two nodes whose labels contain the same requests for the next and the previous nodes, while ${}_lF_k$ checks that all the X-eventualities are fulfilled between those nodes. It can be proved that $|\phi|^k$ correctly encodes tableau trees where at least one branch is accepted in $k+1$ steps.

▶ **Lemma 7.** *Let ϕ be an* LTL+Past *formula. If the complete tableau for ϕ contains an accepted branch of $k+1$ step nodes, then $|\phi|^k$ is satisfiable.*

Proof. Suppose that the complete tableau for ϕ contains an accepted branch of $k+1$ step nodes, so let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be such a branch, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_k \rangle$ be the sequence of its step nodes. Then, by Lemma 6, $[\![\phi]\!]^k$ is satisfiable. We can then build a truth assignment ν in the same way as in the proof of Lemma 6, such that $\nu \models [\![\phi]\!]^k$. Remember that this means we set $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$ for all $0 \leq i \leq k$. So now we have to prove that ν satisfies either E_k or L_k. We will need an auxiliary fact, that is, that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \mathrm{snf}(\psi)_G^i$. That can be done by induction on the structure of ψ, exploiting the definition of the expansion rules of the tableau.

Now, we distinguish two cases depending on which rule accepted the branch:

1. if the branch was accepted by the EMPTY rule, then $\Gamma(\pi_k) = \varnothing$, hence, in particular $\Gamma(\pi_k)$ does not contain any X-request. Hence by definition of ν we have that $\nu \models \neg\psi_G^k$ for any $\psi \in \mathsf{XR}$, so E_k is satisfied;
2. if the branch was accepted by the LOOP rule, then we have a node π_l such that $\Gamma(\pi_l) = \Gamma(\pi_k)$, hence by definition of ν we have $\nu \models \psi_G^l$ if and only if $\nu \models \psi_G^k$ for any $\psi \in \mathsf{XR} \cup \mathsf{YR} \cup \mathsf{ZR}$, so ${}_lR_k$ is satisfied. Moreover, we know that for any X-eventuality $\psi \equiv \mathsf{X}(\psi_1 \,\mathcal{U}\, \psi_2)$ requested in $\Gamma(\pi_k)$, ψ has been fulfilled between π_l and π_k, i.e. there is a $l < j \leq k$ such that $\psi_2 \in \Gamma^*(\pi_j)$. Hence we know that $\nu \models \mathrm{snf}(\psi_2)_G^j$, hence ${}_lF_k$ is satisfied. Then, ${}_lR_k \wedge_l F_k$ is satisfied for at least one l, so L_k is satisfied. ◀

▶ **Lemma 8.** *Let ϕ be an* LTL+Past *formula. If $|\phi|^k$ is satisfiable then the complete tableau for ϕ contains an accepted branch.*

Proof. Suppose that $|\phi|^k$ is satisfiable, hence we have a truth assignment ν such that $\nu \models |\phi|^k$. Then, $[\![\phi]\!]^k$ is satisfiable, and we know from Lemma 6 that the complete tableau for ϕ has a branch that is either accepted, crossed by PRUNE, or longer than $k+1$ step nodes. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be the branch prefix found as shown in the proof of Lemma 6, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. By construction we have a function $J : \mathbb{N} \to \mathbb{N}$ fulfilling the invariant that $\psi \in \Gamma(u_i)$ if and only if $\nu \models \mathrm{snf}(\psi)_G^{J(i)}$. We now show that indeed $\overline{u}$ is accepted or is the prefix of an accepted branch. Since $|\phi|^k$ is satisfiable, either E_k or L_k are satisfiable as well:

1. if E_k is satisfiable, we know that $\nu \models \neg\psi_G^k$ for each $\psi \in \mathsf{XR}$. Since ψ is an X-request, $\mathrm{snf}(\psi) \equiv \psi$, so $\nu \not\models \mathrm{snf}(\psi)_G^k$. Here, $k = J(j)$ for some j, and from the invariant we know that $\psi \notin \Gamma(u_j)$. Hence, u_j does not contain any X-request, so its successor u_{j+1} has an empty label, triggering the EMPTY rule that accepts the branch.
2. if L_k is satisfiable, so are ${}_lR_k$ and ${}_lF_k$ for some $0 \leq l < k$. Hence from ${}_lR_k$ we know that $\nu \models \psi_G^l$ if and only if $\nu \models \psi_G^k$ for all $\psi \in \mathsf{XR} \cup \mathsf{YR} \cup \mathsf{ZR}$, that is $\nu \models \mathrm{snf}(\psi)_G^l$ if and only if $\nu \models \mathrm{snf}(\psi)_G^k$ because ψ is an X-, Y-, or Z-request. Here, $l = J(i)$ and $k = J(j)$ for some i and some j. Since the value of the function J increments at each step node, we can assume *w.l.o.g.* that u_i and u_j are step nodes, and by the invariant we know $\psi \in \Gamma(u_i)$ if and only if $\psi \in \Gamma(u_j)$, i.e. u_i and u_j have the same X-, Y-, and Z-request. Similarly, the fact that $\nu \models {}_lF_k$ tells us that all the X-eventualities requested in u_i are fulfilled between u_{i+1} and u_j. The LOOP rule requires two identical labels in order to trigger, but u_i and u_j only have the same requests. However, since they have the same X-requests, we know that $\Gamma(u_{i+1}) = \Gamma(u_{j+1})$. Then, there is a step node $u_{j'}$, grandchild of u_j, such that $\Gamma(u_j) = \Gamma(u_{j'})$ and the segment of the branch between u_{i+1} and u_j is equal to the segment between u_{j+1} and $u_{j'}$, hence all the X-eventualities requested in u_i and u_j, fulfilled between u_{i+1} and u_j, are fulfilled between u_{j+1} and $u_{j'}$ as well, and the LOOP rule can apply to $u_{j'}$, accepting the branch. ◀

Lastly, the formula $|\phi|_T^k$, called the *termination encoding*, encodes the PRUNE rule. The formula is defined as follows:

$$|\phi|_T^k \equiv [\![\phi]\!]^k \wedge \bigwedge_{i=0}^{k} \neg P^i$$

where

$$P^k \equiv \bigvee_{l=0}^{k-2} \bigvee_{j=l+1}^{k-1} \left({}_lR_j \wedge {}_jR_k \wedge {}_lP_j^k \right)$$

$${}_lP_j^k \equiv \bigwedge_{\substack{\psi \in \mathsf{XEV} \\ \psi \equiv \mathsf{X}(\psi_1 \mathcal{U} \psi_2)}} \left(\psi_G^k \wedge \bigvee_{i=j+1}^{k} \mathrm{snf}(\psi_2)_G^i \rightarrow \bigvee_{i=l+1}^{j} \mathrm{snf}(\psi_2)_G^i \right)$$

It can be proved that $|\phi|_T^k$ is unsatisfiable if the tableau for ϕ contains only rejected branches.

▶ **Lemma 9.** *Let ϕ be an* LTL+Past *formula. If $|\phi|_T^k$ is unsatisfiable, then the complete tableau for ϕ contains only rejected branches.*

Proof. We prove the contrapositive, i.e. that if the complete tableau for ϕ contains an accepted branch, then $|\phi|_T^k$ is satisfiable. Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be such a branch, and let $\overline{\pi} = \langle \pi_0, \ldots, \pi_m \rangle$ be the sequence of its step nodes. By Lemma 6, we know $[\![\phi]\!]^k$ is satisfiable, thus we can obtain a truth assignment ν such that $\nu \models [\![\phi]\!]^k$. We can build ν as in the proof of Lemma 6, i.e. such that $\nu(p^i) = \top$ if and only if $p_U \in \Gamma(\pi_i)$ for all $0 \leq i \leq k$. Similarly to the proof of Lemma 7, we highlight the fact that $\psi \in \Gamma^*(\pi_i)$ if and only if $\nu \models \mathrm{snf}(\psi)_G^i$. Now, since the branch is accepted, the PRUNE rule cannot be applied to it. This means that either a) there are no three nodes π_u, π_v, π_w such that $\Gamma(\pi_u) = \Gamma(\pi_v) = \Gamma(\pi_w)$, or b) these three nodes exist but there is an X-eventuality ψ requested in $\Gamma(\pi_w)$ that is fulfilled between π_u and π_v and not between π_v and π_w. In case a) this means ${}_uR_v \wedge {}_vR_w$ does not hold for any u and v. In case b), ${}_uR_v \wedge {}_vR_w$ holds but ${}_uP_v^w$ does not. In any case, it follows that $\neg P^i$ holds for any $0 \leq i \leq k$, hence $|\phi|_T^k$ is satisfied. ◀

Together with the soundness and completeness results for the underlying tableau [13], the above Lemmata allow us to prove the soundness and completeness of the procedure of Algorithm 1.

▶ **Theorem 10** (Soundness and completeness). *Let ϕ be an* LTL+Past *formula. The* BLACK *algorithm answers* satisfiable *on ϕ if and only if ϕ is satisfiable.*

Proof. ($\rightarrow$) Suppose the BLACK algorithm answers *satisfiable* on the formula ϕ. Then, it means there is a $k \geq 0$ such that $|\phi|^k$ is satisfiable. By Lemma 8, the complete tableau for ϕ has an accepting branch. By the soundness of the tableau, then ϕ is satisfiable.

($\leftarrow$) Now suppose the formula ϕ is satisfiable. By the completeness of the tableau, the complete tableau for ϕ has an accepting branch. Let us suppose such a branch has $k+1$ step nodes for some $k \geq 0$. Then, we want to show that the BLACK algorithm eventually answers *satisfiable*. Let $i < k$ be any earlier iteration of the main loop of the algorithm. We have that by Lemma 6, $[\![\phi]\!]^i$ is satisfiable because there is a branch longer than $i+1$ step nodes. Similarly, by Lemma 9, $|\phi|^i_T$ is satisfiable because not all the branches of the tableau are rejected. Hence, the algorithm does not answer *unsatisfiable* at step i. Arrived at step k, $|\phi|^k$ is satisfiable by Lemma 7 because the tableau has an accepted branch of $k+1$ step nodes, hence the algorithm answers *satisfiable*. ◀

Despite the final encoding may seem trivially simple, this simplicity makes the approach interesting for two reasons. First of all, it was not *a priori* clear that such a simple encoding would be possible, given the presence of the FORECAST rule: a rule that kills the performance of the explicit construction of the tableau turns out to be a non-problem in the symbolic exploration of the tree. Secondly, this simplicity is what drives the experimental success of the encoding. Comparing it for example with Biere et al. [2] which is, as far as we know, the approach implemented by nuXmv: all the *virtual unrollings* machinery that they need to support past operators is much more complex than our seemingly trivial encoding which, however, performs better even though it does not generate CNF formulas of linear size. Hence, showing that the approach by Geatti et al. [12] can be successfully extended so easily can be considered one of the main contributions of this work.

5 Experimental results

We have implemented the above two approaches in version 0.3.0 of the BLACK tool[1]: the translation as an optional module that can be activated upon user request, and the direct encoding as an expansion of the core procedure.

Since the literature lacks significant LTL+Past family of formulas for benchmarks, we have devised two novel sets of formulas. As for the first family, we chose a set of *random formulas*, generated with an algorithm adapted from [24], in order to verify how the tool scales in general. The second family, that we called `crscounter`, is inspired by and adapted from Cimatti et al. [7], where a Kripke structure called $Counter(N)$, where N is a power of two, is introduced. $Counter(N)$ works as follows: it starts at $c = 0$, counts up to $c = N$, jumps back to $c = N/2$, and then loops, counting up to $c = N$ and jumping back to $c = N/2$, forever. Afterwards, they evaluated, on top of that Kripke structure, some parametric properties of the form:

$$P(i) \equiv \neg \mathsf{F}\big(\mathsf{O}((c = \tfrac{N}{2}) \wedge \mathsf{O}((c = \tfrac{N}{2} + 1) \wedge \ldots \wedge \mathsf{O}(c = \tfrac{N}{2} + i)\ldots))\big).$$

[1] The tool can be found at `https://github.com/black-sat/black`. Packages for macOS and common Linux distributions are provided, together with all the necessary scripts to reproduce the tests performed.

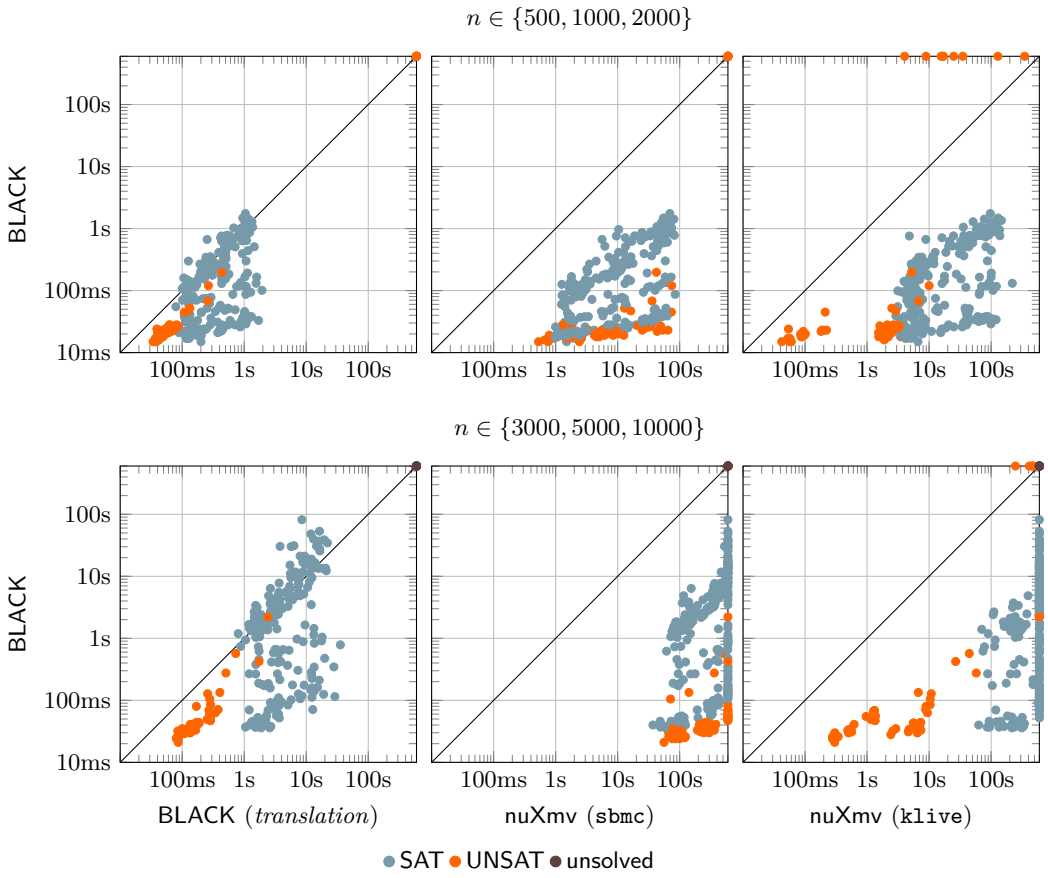


Figure 1 Experimental results of BLACK against the other tested tools and modalities, for random formulas of size $n \in \{500, 1000, 2000, 3000, 5000, 10000\}$. Times are in *seconds*. Note that only instances that could not be solved by *any* tool are marked as unsolved.

The value i identifies the number of nested *once* operators, while the structure of such properties requires that the loop of length $N/2$ in the model is traversed backwards several times in order to reach a counterexample.

The `crscounter` benchmarks were introduced in the context of model checking. Thus, we made a reduction from the model checking problem to the satisfiability checking one for LTL+Past: we built the LTL+Past formulas $\phi_{Counter(N)}$ and $\phi_{P(i)}$ encoding the above elements. In this way, $\neg(\phi_{Counter(N)} \rightarrow \phi_{P(i)})$ is UNSAT if and only if $Counter(N) \models P(i)$. With this framework, we were able to obtain both SAT ($i \leq \frac{N}{2}$) and UNSAT ($i > \frac{N}{2}$) instances. Moreover, this family of formulas stresses the ability to process past operators and find short counterexamples, and thus, it specifically challenges our contribution.

For each formula, we executed both an *internal comparison* between the two proposed techniques – with BLACK over MathSAT [4], which is the best performing solver among those supported by BLACK so far –, and an *external comparison* with nuXmv [5], which, as far as we know, is the only state-of-the-art tool directly supporting past operators. Specifically, we tested BLACK against nuXmv in both `sbmc` and `klive` modalities. The former stands for Simple Bounded Model Checking [2], and it is the closest to our approach between the two. The latter has been proposed more recently, and it is based on K-Liveness [6].

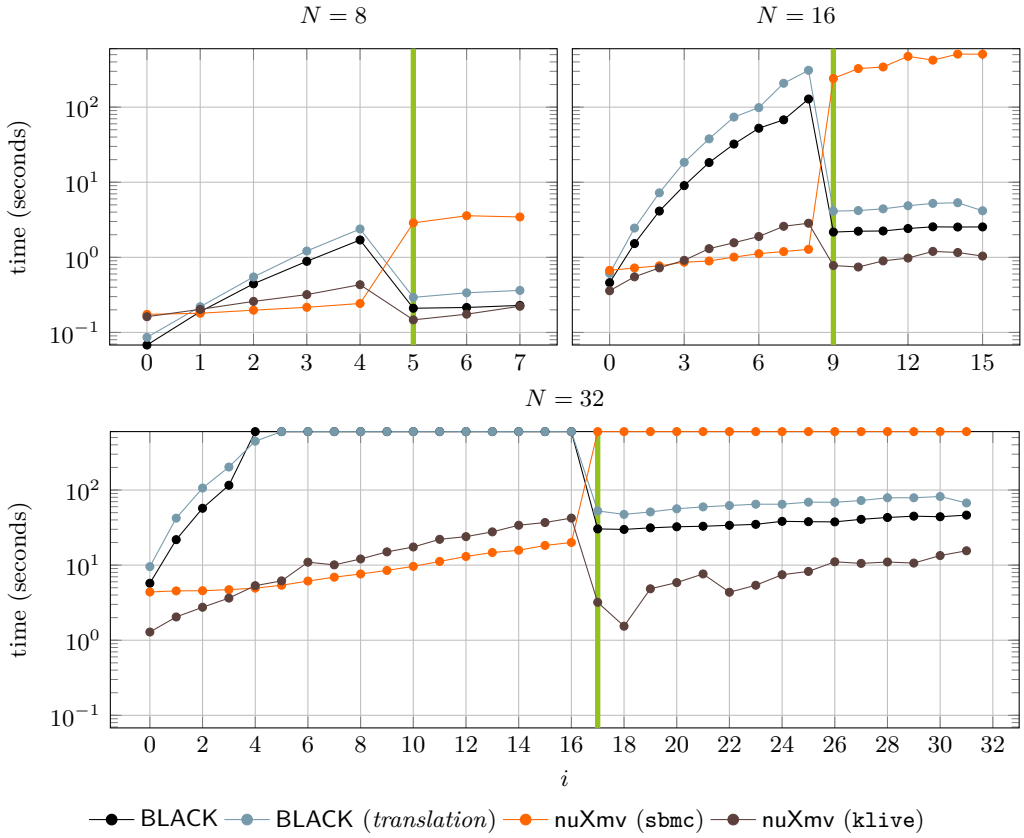


Figure 2 Experimental results for `crscounter` formulas of size $N \in \{8, 16, 32\}$. Green vertical bars indicate where formulas start to be UNSAT.

All the experiments have been performed on a Dell PowerEdge rack server equipped with a 16-cores AMD EPYC™ 7281 CPU (2.7GHz) and 64GB of RAM (DDR4 2400 MT/s). Moreover, experiments have been run in parallel, each on a single core, with a *memory limit* of 3GB of RAM per core, and a 10 minutes *timeout*. Experimental results plots can be found in Figure 1 and Figure 2.

Regarding the random formulas, what immediately catches the eye is that BLACK– in the direct approach – performs overall significantly better, in particular if compared with nuXmv. Indeed, the latter starts to be in difficulty already at size $n = 3000$, and gets almost always stuck at size $n = 10000$. It can also be noticed that BLACK solves UNSAT instances quicker than the SAT ones. This could be a bit surprising, considering that the former is a universal property, while the latter is an existential one. However, this has a twofold explanation. Firstly, the Boolean encoding acts like a breadth-first search, which is in some sense an exhaustive search, up to depth k. Secondly, in all UNSAT random formulas the tableaux are closed by contradiction, and never by the PRUNE rule. This is because of the nature of the random formulas generator: it is less likely to generate a formula with such a peculiar structure as to trigger the PRUNE rule. Also, it is more likely to produce contradictions that can be spotted in the first depth levels. Nevertheless, it is interesting to note that BLACK is able to manage those UNSAT formulas better than nuXmv, overall.

Looking instead at the `crscounter` plots, there are two interesting aspects to point out. First, BLACK performs slightly better with the direct past encoding than with the translation approach. Second, while BLACK performs worse with SAT formulas, the situation overturns

when the formulas become UNSAT. This can be explained by the PRUNE rule: it may cause an overhead in the encoding of SAT formulas, but it has the advantage of guaranteeing a quicker termination, i.e. closure of the tableau, in case of UNSAT ones. Indeed, in this family of formulas, all UNSAT ones have a tableau which is closed by the PRUNE rule.

6 Conclusions

This work contributes to the state of the art of the satisfiability problem for LTL+Past in different directions. First of all, it provides a translation procedure for LTL+Past, that has been implemented into the BLACK tool, but, in principle, can be used also as a preprocessing step for any LTL satisfiability checker in order to let them support the past. Then, it extends the SAT-based encoding of a one-pass and tree-shaped tableau for LTL to LTL+Past, and shows that the resulting tool is very effective and efficient compared to the state-of-the-art ones. It is also worth noting that the encoding is quite simple compared to the *virtual unrollings* techniques used to support past operators by Latvala et al. [15], and it offers an exponential advantage over the explicit construction of the tableau since the FORECAST rule is not required to be encoded. Finally, it introduces two new sets of formulas which aim at starting to build a larger set of standardized formulas for benchmarks in the field.

Results showed that the direct encoding is the preferred way to manage the past, but also that the translation is a viable alternative, as it adds only a linear overhead.

Future work should head at reducing the overhead introduced by the PRUNE rule, possibly by triggering it not at each step, but with some predetermined heuristics, in order to reduce the gap in performance between SAT and UNSAT properties. Moreover, since the framework has been shown to be efficient and modular, it could be interesting to investigate its extension to other logics. Orthogonally, some in-depth theoretical comparison with other state-of-the-art techniques could suggest new ways of improvement.

References

1 Matteo Bertello, Nicola Gigante, Angelo Montanari, and Mark Reynolds. Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In *Proc. of the 25th International Joint Conference on Artificial Intelligence*, pages 950–956. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/139.

2 Armin Biere, Keijo Heljanko, Tommi A. Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Log. Methods Comput. Sci.*, 2(5), 2006. doi:10.2168/LMCS-2(5:5)2006.

3 Laura Bozzelli, Aniello Murano, and Loredana Sorrentino. Alternating-time temporal logics with linear past. *Theor. Comput. Sci.*, 813:199–217, 2020. doi:10.1016/j.tcs.2019.11.028.

4 Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT Solver. In *Computer Aided Verification*, LNCS, pages 299–303. Springer, 2008. doi:10.1007/978-3-540-70545-1_28.

5 Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv Symbolic Model Checker. In *Computer Aided Verification*, pages 334–342. Springer, 2014. doi:10.1007/978-3-319-08867-9_22.

6 Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Verifying LTL Properties of Hybrid Systems with K-Liveness. In *Computer Aided Verification*, LNCS, pages 424–440. Springer, 2014. doi:10.1007/978-3-319-08867-9_28.

7 Alessandro Cimatti, Marco Roveri, and Daniel Sheridan. Bounded Verification of Past LTL. In *Formal Methods in Computer-Aided Design*, LNCS, pages 245–259. Springer, 2004. doi:10.1007/978-3-540-30494-4_18.

8 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8.

9 Michael Fisher. A resolution method for temporal logic. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 99–104. Morgan Kaufmann, 1991.

10 Michael Fisher. A normal form for temporal logics and its applications in theorem-proving and execution. *J. Log. Comput.*, 7(4):429–456, 1997. doi:10.1093/logcom/7.4.429.

11 Michael Fisher, Dov M. Gabbay, and Lluís Vila, editors. *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*. Elsevier, 2005. URL: http://cgi.csc.liv.ac.uk/%7Emichael/handbook.html.

12 Luca Geatti, Nicola Gigante, and Angelo Montanari. A SAT-Based Encoding of the One-Pass and Tree-Shaped Tableau System for LTL. In *Proc. of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 11714 of *LNCS*, pages 3–20. Springer, 2019. doi:10.1007/978-3-030-29026-9_1.

13 Luca Geatti, Nicola Gigante, Angelo Montanari, and Mark Reynolds. One-pass and tree-shaped tableau systems for TPTL and TPTL_b+Past. *Information and Computation*, 2021. in press. doi:10.1016/j.ic.2020.104599.

14 Ullrich Hustadt and Boris Konev. TRP++2.0: A Temporal Resolution Prover. In *Proc. of the 19th International Conference on Automated Deduction*, pages 274–278, 2003.

15 Timo Latvala, Armin Biere, Keijo Heljanko, and Tommi A. Junttila. Simple is better: Efficient bounded model checking for past LTL. In *Proc. of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 3385 of *LNCS*, pages 380–395. Springer, 2005. doi:10.1007/978-3-540-30579-8_25.

16 Jianwen Li, Yinbo Yao, Geguang Pu, Lijun Zhang, and Jifeng He. Aalta: an LTL satisfiability checker over infinite/finite traces. In *Proc. of the 22nd ACM International Symposium on Foundations of Software Engineering*, pages 731–734. ACM, 2014. doi:10.1145/2635868.2661669.

17 Orna Lichtenstein and Amir Pnueli. Propositional Temporal Logics: Decidability and Completeness. *Logic Journal of the IGPL*, 8(1):55–85, 2000. doi:10.1093/jigpal/8.1.55.

18 Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The Glory of the Past. In *Proc. of the 1st Conference on Logics of Programs*, volume 193 of *LNCS*, pages 196–218. Springer, 1985. doi:10.1007/3-540-15648-8_16.

19 Nicolas Markey. Temporal logic with past is exponentially more succinct. *Bulletin of the EATCS*, 79:122–128, 2003.

20 Amir Pnueli. The Temporal Logic of Programs. In *Proc. of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

21 Mark Reynolds. More Past Glories. In *Proc. of the 15th Annual IEEE Symposium on Logic in Computer Science*, pages 229–240. IEEE Computer Society, 2000. doi:10.1109/LICS.2000.855772.

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

23 A. Prasad Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

24 Heikki Tauriainen and Keijo Heljanko. Testing LTL formula translation into Büchi automata. *International Journal on Software Tools for Technology Transfer*, 4(1):57–70, 2002. doi:10.1007/s100090200070.

25 G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, Symbolic Computation, pages 466–483. Springer, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_28.

26 Pierre Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56(1/2):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.

PSPACE-Completeness of the Temporal Logic of Sub-Intervals and Suffixes

Laura Bozzelli
University of Napoli "Federico II", Italy

Angelo Montanari
University of Udine, Italy

Adriano Peron
University of Napoli "Federico II", Italy

Pietro Sala
University of Verona, Italy

Abstract

In this paper, we establish PSPACE-completeness of the finite satisfiability and model checking problems for the fragment of Halpern and Shoham interval logic with modality $\langle E \rangle$, for the "suffix" relation on pairs of intervals, and modality $\langle D \rangle$, for the "sub-interval" relation, under the homogeneity assumption. The result significantly improves the EXPSPACE upper bound recently established for the same fragment, and proves the rather surprising fact that the complexity of the considered problems does not change when we add either the modality for suffixes ($\langle E \rangle$) or, symmetrically, the modality for prefixes ($\langle B \rangle$) to the logic of sub-intervals (featuring only $\langle D \rangle$).

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Interval temporal logic, Satisfiability, Model checking

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.9

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

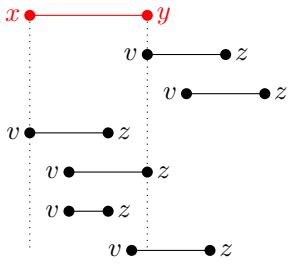
1 Introduction

For a long time, in computer science, interval temporal logics (ITLs) have been considered an attractive, but impractical, alternative to standard point-based ones. On the one hand, they are a natural choice as a specification/representation language in a number of domains; on the other hand, the high undecidability of the satisfiability problem for the most well-known ITLs [8, 14, 16, 20, 30], such as Halpern and Shoham's modal logic of time intervals (HS for short) [14] and Venema's CDT [30], discouraged their extensive use (in fact, some restricted variants of them have been applied in formal verification and AI over the years [17, 24, 26]).

The present work finds its place in the framework of the logic HS, which features one modality for each of the 13 Allen's relations [1], apart from equality. In Table 1, we depict 6 Allen's relations for ordered pairs of intervals, together with the corresponding HS (existential) modalities; the other 7 relations are their inverses and the equality relation. The recent discovery of a significant number of expressive and computationally well-behaved fragments of HS changed the landscape of ITL research [11, 23]. Meaningful examples are the logic $A\overline{A}$ of the temporal neighborhood [10] (the HS fragment with modalities for the *meets* relation and its inverse) and the logic D of (temporal) sub-intervals [9] (the HS fragment with modality $\langle D \rangle$ for the *contains* relation only) over dense orderings.

Model checking (MC) of (finite) Kripke structures against HS and its fragments has been investigated in a series of papers [3, 5, 17, 18, 19, 21, 22] and shown to be decidable. In this setting, each finite path of a Kripke structure is interpreted as an interval whose

Table 1 Allen's relations and corresponding HS modalities.

Allen relation	HS	Definition w.r.t. interval structures	Example
MEETS	$\langle A \rangle$	$[x,y]\mathcal{R}_A[v,z] \iff y = v$	
BEFORE	$\langle L \rangle$	$[x,y]\mathcal{R}_L[v,z] \iff y < v$	
STARTED-BY	$\langle B \rangle$	$[x,y]\mathcal{R}_B[v,z] \iff x = v \wedge z < y$	
FINISHED-BY	$\langle E \rangle$	$[x,y]\mathcal{R}_E[v,z] \iff y = z \wedge x < v$	
CONTAINS	$\langle D \rangle$	$[x,y]\mathcal{R}_D[v,z] \iff [v,z] \subset [x,y]$	
OVERLAPS	$\langle O \rangle$	$[x,y]\mathcal{R}_O[v,z] \iff x < v < y < z$	

labeling satisfies the *homogeneity assumption* [27]: a proposition letter holds over an interval if and only if it holds over all its constituent points (states). MC against full HS is at least EXPSPACE-hard [5] and the only known upper bound is non-elementary [21, 6].[1] The known complexity bounds of MC for full HS coincide with those of MC for the linear-time fragment BE of HS which features modalities ⟨B⟩ and ⟨E⟩ for prefixes and suffixes. These complexity bounds easily transfer to finite satisfiability, that is, satisfiability over finite linear orders, of BE under the homogeneity assumption. Whether or not these problems can be solved elementarily is a difficult open question. On the other hand, MC and finite satisfiability under the homogeneity assumption for all the fragments of BE are known to be elementarily decidable [2, 3, 7]. In particular, for the fragment D of BE (note that the *contains* relation $\mathcal{R}_D$ can be expressed as $\mathcal{R}_B \cup \mathcal{R}_E \cup \mathcal{R}_B \cdot \mathcal{R}_E$), these problems are known to be PSPACE-complete [2].

In a recent contribution [7], we investigated finite satisfiability under the homogeneity assumption for the maximal fragment BD of BE that features modalities ⟨B⟩ and ⟨D⟩ (the other maximal fragment DE of BE with modalities ⟨D⟩ and ⟨E⟩ is completely symmetric, and thus all results for BD immediately transfer to it, and vice versa). The addition of modality ⟨B⟩ makes satisfiability checking for BD more complex than the one for D, as the two relations/modalities may interact in a non-trivial way. We proved EXPSPACE membership of the problem [7] by means of a purely model-theoretic argument, leaving the question of its exact complexity open. In this paper, we answer the question proving that, surprisingly, PSPACE-completeness of D is not affected by the addition of either ⟨B⟩ or ⟨E⟩, and the MC problem for DE (and symmetrically BD) is PSPACE-complete as well. In Figure 1, we add these new MC results to the picture of known MC complexities, showing that they enrich the set of HS "tractable" fragments with two meaningful members. We propose an automata-theoretic approach for solving MC and finite satisfiability under the homogeneity assumption of DE and BD which non-trivially generalizes the one for D [2] and the well-known one for standard LTL [29]. In particular, some important aspects that were not well understood in [2] are generalized by an elegant algebraic framework, which allows us to solve in an asymptotically optimal way the considered problems for DE and BD. In addition, we prove that, over finite linear orders and under the homogeneity assumption, D is less expressive than BD and DE, which in turn are less expressive than BE (in [4], we show that, under the homogeneity assumption, BE and LTL over finite words have the same expressive power).

[1] An expressive comparison of MC for HS and standard point-based temporal logics LTL [25], CTL, and CTL* [13] can be found in [4].

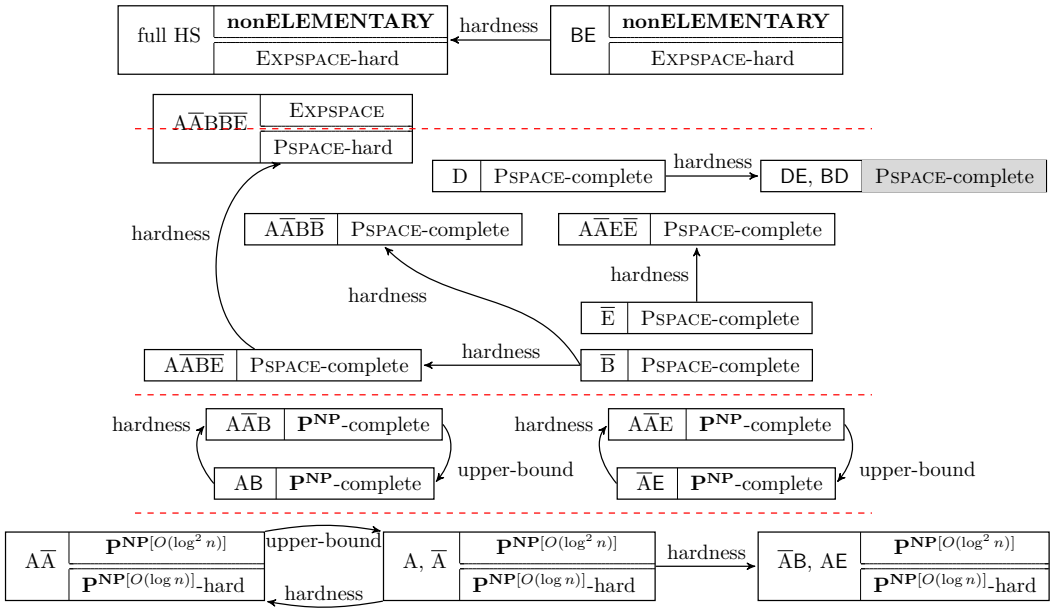


Figure 1 Complexity of the MC problem for HS and its fragments.

We conclude the introduction by recalling an interesting connection between the finite satisfiability problem for BE and its fragments, under the homogeneity assumption, and the non-emptiness problem for generalized $*$-free regular expressions [7]. The latter problem has been shown to be non-elementarily decidable by Stockmeyer in [28], and it can be easily proved to be equivalent to finite satisfiability for the interval temporal logic C of the chop modality [15, 24, 30] under the homogeneity assumption (the chop modality allows one to split the current interval in two parts and to state what is true over the first part and what over the second one). It can be shown that over finite linear orders and under the homogeneity assumption, BE (resp., its proper fragments BD and DE) is equivalent to the weakening of generalized $*$-free regular expressions where the concatenation operator is replaced by the weaker *prefix* and *suffix* ones (resp., *prefix* and *infix*, and *infix* and *suffix*) [7]. Note that the infix operator can be expressed in terms of the combination of the prefix and suffix operators. An immediate by-product of the results given in this paper is that the non-emptiness problem for $*$-free generalized regular expressions turns out to be elementarily decidable and, precisely, PSPACE-complete if one makes use of the *suffix* (resp., *prefix*) operator and the *infix* operator instead of the concatenation operator in the expressions. As for the fragment with both the *prefix* and the *suffix* operators, we only know that its non-emptiness problem is (non-elementarily decidable and) EXPSPACE-hard [5].

2 Preliminaries

We fix the following notation. For a finite word (or sequence) w over some finite alphabet Σ, we denote by $|w|$ the length of w. Moreover, for all $0 \leq i < |w|$, $w[i]$ denotes the $(i+1)^{th}$ letter of w. Given two non-empty finite words w, w' over Σ, we denote by $w \cdot w'$ the concatenation of w and w'. Moreover, if the last letter of w coincides with the first letter of w', we denote by $w \star w'$ the word $w \cdot w'[1] \dots w'[n-1]$, where $n = |w|$ (i.e. the word obtained by concatenating w with the word obtained from w' by erasing the first letter). In particular, when $|w'| = 1$, then $w \star w' = w$.

Finite automata over finite words. A nondeterministic finite automaton (NFA) is a tuple $\mathcal{N} = \langle \Sigma, Q, Q_0, \delta, F \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is the set of *initial* states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of *accepting* states. Given a finite word w over Σ, with $|w| = n$, a run of $\mathcal{N}$ over w is a finite sequence of states $q_0, \ldots, q_n$ such that $q_0 \in Q_0$, and for all $i \in [0, n-1]$, $q_{i+1} \in \delta(q_i, w[i])$. The language $\mathcal{L}(\mathcal{N})$ accepted by $\mathcal{N}$ consists of the finite words w over Σ such that there is a run over w ending in some accepting state. A deterministic finite automaton (DFA) is an NFA $\mathcal{D} = \langle \Sigma, Q, \{q_0\}, \delta, F \rangle$ such that for all $(q, \sigma) \in Q \times \Sigma$, $\delta(q, \sigma)$ is a singleton.

Finite Kripke structures. We fix a finite set $\mathcal{AP}$ of proposition letters which represent predicates over the states of the given system. A *finite Kripke structure* over $\mathcal{AP}$ is a tuple $\mathcal{K} = \langle W, s_0, E, \mu \rangle$, where W is a finite set of states, $s_0 \in W$ is the initial state, $E \subseteq W \times W$ is a left-total relation between states, and $\mu : W \to 2^{\mathcal{AP}}$ is a labelling function assigning to each state the set of propositions that hold at it.

A *path of* $\mathcal{K}$ is a non-empty finite sequence of states $\rho = s_1 \cdots s_n$ such that (i) the first state s_1 coincides with the initial state s_0 of $\mathcal{K}$, and (ii) $(s_i, s_{i+1}) \in E$ for $i = 1, \ldots, n-1$. We extend the labeling μ to paths of $\mathcal{K}$ in the usual way: for a path $\rho = s_1 \ldots s_n$, $\mu(\rho)$ denotes the word over $2^{\mathcal{AP}}$ of length n given by $\mu(s_1) \ldots \mu(s_n)$. A *trace* of $\mathcal{K}$ is a non-empty finite word over $2^{\mathcal{AP}}$ of the form $\mu(\rho)$ for some path ρ of $\mathcal{K}$.

3 The logics DE and BD under the homogeneity assumption

In this section, we recall the logic BE of prefix and suffixes corresponding to the linear-time fragment of HS, and we focus our attention on the fragments DE and BD of BE interpreted over finite linear orders under the homogeneity assumption.

Let $\mathbb{S} = \langle S, < \rangle$ be a linear order over the nonempty set $S \neq \emptyset$, and $\leq$ be the reflexive closure of $<$. Given $x, y \in S$ such that $x \leq y$, we denote by $[x, y]$ the (closed) *interval* over S given by the set of elements $z \in S$ such that $x \leq z$ and $z \leq y$. We denote the set of all intervals over $\mathbb{S}$ by $\mathbb{I}(\mathbb{S})$. We focus our attention on three Allen's relations over intervals:

1. the *proper prefix (or started-by)* relation $\mathcal{R}_B$ defined as follows: $[x, y] \mathcal{R}_B [x', y']$ if $x = x'$ and $y' < y$.
2. the *proper sub-interval (or contains)* relation $\mathcal{R}_D$ defined as follows: $[x, y] \mathcal{R}_D [x', y']$ if $x' \geq x$, $y' \leq y$, and $[x, y] \neq [x', y']$ (the proper subset relation over intervals), and
3. the *proper suffix (or finished-by)* relation $\mathcal{R}_E$ defined as follows: $[x, y] \mathcal{R}_E [x', y']$ if $x < x'$ and $y' = y$.

BE formulas φ are defined by the following syntax:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle \mathrm{B} \rangle \varphi \mid \langle \mathrm{D} \rangle \varphi \mid \langle \mathrm{E} \rangle \varphi$$

where $p \in \mathcal{AP}$, and $\langle \mathrm{B} \rangle$ (resp, $\langle \mathrm{D} \rangle$, resp., $\langle \mathrm{E} \rangle$) is the existential temporal modality for the Allen's relation $\mathcal{R}_B$ (resp., $\mathcal{R}_D$, resp., $\mathcal{R}_E$). We also exploit the conjunction connective $\wedge$, and for any temporal modality $\langle \mathrm{X} \rangle$, with $X \in \{B, D, E\}$, the dual universal modality $[\mathrm{X}]$ defined as: $[\mathrm{X}] \psi := \neg \langle \mathrm{X} \rangle \neg\psi$. The size $|\varphi|$ of a formula φ is the number of distinct sub-formulas of φ. We focus on the fragments DE (logic of sub-intervals and suffixes) and BD (logic of sub-intervals and prefixes) of BE obtained by disallowing the temporal modalities for the Allen's relations $\mathcal{R}_B$ and $\mathcal{R}_E$, respectively. We also consider the fragments B, D, and E defined in the obvious way.

The semantics of the logic BE is given in terms of *interval models*. An interval model $\mathcal{M}$ is a pair $\langle \mathbb{I}(\mathbb{S}), \mathcal{V} \rangle$, where $\mathcal{V} : \mathcal{AP} \mapsto 2^{\mathbb{I}(\mathbb{S})}$ is a *valuation function* that assigns to every proposition letter p the set of intervals $\mathcal{V}(p)$ over which p holds. Given an interval model $\mathcal{M} = \langle \mathbb{I}(\mathbb{S}), \mathcal{V} \rangle$, an interval $[x, y] \in \mathbb{I}(\mathbb{S})$, and a formula φ, the *satisfaction relation* $\mathcal{M}, [x, y] \models \varphi$, meaning that φ holds over the interval $[x, y]$ of $\mathcal{M}$, is inductively defined as follows:

- for every proposition letter $p \in \mathcal{AP}$, $\mathcal{M}, [x, y] \models p$ *if* $[x, y] \in \mathcal{V}(p)$;
- $\mathcal{M}, [x, y] \models \neg\varphi$ *if* $\mathcal{M}, [x, y] \not\models \varphi$;
- $\mathcal{M}, [x, y] \models \varphi_1 \vee \varphi_2$ *if* $\mathcal{M}, [x, y] \models \varphi_1$ or $\mathcal{M}, [x, y] \models \varphi_2$;
- $\mathcal{M}, [x, y] \models \langle X \rangle \varphi$ for $X \in \{B, D, E\}$ *if* there is an interval $[x', y'] \in \mathbb{I}(\mathbb{S})$ such that $[x, y] \mathcal{R}_X [x', y']$ and $\mathcal{M}, [x', y'] \models \varphi$.

A BE-formula is *satisfiable* if it holds over some interval of an interval model. In this paper, we restrict our attention to the finite satisfiability problems, that is, satisfiability over the class of finite linear orders, for the fragments BD and DE. The problems are known to be undecidable [20] in the general case, but decidability can be recovered by restricting to the class of *homogeneous* interval models [21]. Formally, an interval model $\mathcal{M} = \langle \mathbb{I}(\mathbb{S}), \mathcal{V} \rangle$ is *homogeneous* if for every interval $[x, y] \in \mathbb{I}(\mathbb{S})$ and every $p \in \mathcal{AP}$, it holds that $[x, y] \in \mathcal{V}(p)$ if and only if $[x', x'] \in \mathcal{V}(p)$ for every $x' \in [x, y]$.

We observe that homogeneous interval models over finite linear orders correspond to non-empty finite words over $2^{\mathcal{AP}}$. In particular, each non-empty finite word w over $2^{\mathcal{AP}}$ induces the homogeneous interval model $\mathcal{M}(w) = \langle \mathbb{I}(\mathbb{S}), \mathcal{V} \rangle$ over the finite linear order induced by w defined as follows:

- $\mathbb{S} = \langle \{0, \ldots, |w| - 1\}, < \rangle$, and
- for every interval $[i, j]$ of $\mathbb{S}$ (note that $0 \leq i \leq j < |w|$) and $p \in \mathcal{AP}$, $[i, j] \in \mathcal{V}(p)$ if and only if $p \in w[h]$ for all $h \in [i, j]$.

Any fragment $\mathcal{F}$ of BE interpreted over homogeneous models is denoted by $\mathcal{F}_{\mathcal{H}om}$. A non-empty finite word w over $2^{\mathcal{AP}}$ *satisfies an* $\mathcal{F}_{\mathcal{H}om}$ *formula* φ, denoted by $w \models \varphi$, if $\mathcal{M}(w), [0, |w| - 1] \models \varphi$. A finite Kripke structure $\mathcal{K}$ over $\mathcal{AP}$ is *a model of* φ, written $\mathcal{K} \models \varphi$, if each trace w of $\mathcal{K}$ satisfies φ. We also consider the *model checking problem* against $\mathsf{DE}_{\mathcal{H}om}$ (resp., $\mathsf{BE}_{\mathcal{H}om}$) that is the problem of deciding for a given finite Kripke structure $\mathcal{K}$ and $\mathsf{DE}_{\mathcal{H}om}$ (resp., $\mathsf{BD}_{\mathcal{H}om}$) formula φ, whether $\mathcal{K} \models \varphi$.

Expressiveness issues. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two logics interpreted over non-empty finite words over $2^{\mathcal{AP}}$. Given $\psi_1 \in \mathcal{F}_1$ and $\psi_2 \in \mathcal{F}_2$, ψ_1 and ψ_2 are *equivalent* if ψ_1 and ψ_2 are satisfied by the same non-empty finite words over $2^{\mathcal{AP}}$. We say that $\mathcal{F}_1$ *is subsumed by* $\mathcal{F}_2$, denoted $\mathcal{F}_1 \preceq_f \mathcal{F}_2$, if for each $\mathcal{F}_1$ formula there is an equivalent $\mathcal{F}_2$ formula. $\mathcal{F}_1$ and $\mathcal{F}_2$ *have the same expressiveness* (resp.,*are expressively incomparable*) if $\mathcal{F}_1 \preceq_f \mathcal{F}_2$ and $\mathcal{F}_2 \preceq_f \mathcal{F}_1$ (resp., $\mathcal{F}_1 \not\preceq_f \mathcal{F}_2$ and $\mathcal{F}_2 \not\preceq_f \mathcal{F}_1$). Finally, $\mathcal{F}_1$ *is less expressive than* $\mathcal{F}_2$, denoted by $\mathcal{F}_1 \prec_f \mathcal{F}_2$, if $\mathcal{F}_1 \preceq_f \mathcal{F}_2$ and $\mathcal{F}_2 \not\preceq_f \mathcal{F}_1$. It is known [4] that $\mathsf{BE}_{\mathcal{H}om}$ has the same expressiveness as standard LTL over finite words. Here, we show that over finite words, the fragment $\mathsf{D}_{\mathcal{H}om}$ is less expressive than the fragments $\mathsf{BD}_{\mathcal{H}om}$ and $\mathsf{DE}_{\mathcal{H}om}$, which in turn are less expressive than $\mathsf{BE}_{\mathcal{H}om}$ or, equivalently, LTL. In particular, the following hold (a proof is in Appendix A).

▶ **Theorem 1.** *There exists a* $\mathsf{B}_{\mathcal{H}om}$ *(resp.,* $\mathsf{E}_{\mathcal{H}om}$*) formula which cannot be expressed in* $\mathsf{DE}_{\mathcal{H}om}$ *(resp.,* $\mathsf{BD}_{\mathcal{H}om}$*) over finite linear orders. Hence,* $\mathsf{D}_{\mathcal{H}om} \prec_f \mathsf{BD}_{\mathcal{H}om} \prec_f \mathsf{BE}_{\mathcal{H}om}$*,* $\mathsf{D}_{\mathcal{H}om} \prec_f \mathsf{DE}_{\mathcal{H}om} \prec_f \mathsf{BE}_{\mathcal{H}om}$*, and* $\mathsf{BD}_{\mathcal{H}om}$ *and* $\mathsf{DE}_{\mathcal{H}om}$ *are expressively incomparable.*

In order to illustrate the succinctness of the logics $\mathsf{D}_{\mathcal{H}om}$, $\mathsf{BD}_{\mathcal{H}om}$, and $\mathsf{DE}_{\mathcal{H}om}$, we consider a combinatorial requirement. For each $n \geq 1$, let $\mathcal{AP}_n = \{p_1, \ldots, p_n, \overline{p}_1, \ldots, \overline{p}_n\}$. The property that "there is a proper infix such for each $i \in [1, n]$, *exclusively* either p_i holds at some position, or $\overline{p}_i$ holds at some position" can be expressed by the following $\mathsf{D}_{\mathcal{H}om}$ formula ψ_n.

$$\psi_n := \langle \mathrm{D} \rangle \bigwedge_{i=1}^{i=n} ((\langle \mathrm{D} \rangle\, p_i \wedge [D] \neg \overline{p}_i) \vee (\langle \mathrm{D} \rangle\, \overline{p}_i \wedge [D] \neg p_i))$$

We conjecture that there is no LTL formula equivalent to ψ_n of size polynomial in n.

Encoding $\mathsf{DE}_{\mathcal{H}om}$ and $\mathsf{BE}_{\mathcal{H}om}$ in fragments of generalized $*$-free regular expressions. In the following, we show how to encode in $\mathsf{DE}_{\mathcal{H}om}$ (resp., $\mathsf{BE}_{\mathcal{H}om}$) over finite linear orders the fragment of generalized $*$-free regular expressions where the concatenation operator is replaced by the *infix* and *suffix* ones (resp., the *infix* and *prefix* ones). Recall that a *generalized $*$-free regular expression* (hereafter, simply called *general expression*) e over a finite alphabet Σ is a term of the form:

$$e ::= \emptyset \mid a \mid \neg e \mid e + e \mid \ e \cdot e, \text{ for any } a \in \Sigma.$$

We exclude the empty word ϵ from the syntax as it makes more direct the correspondence between restricted expressions and $\mathsf{BE}_{\mathcal{H}om}$ fragments (such a simplification is quite common in the literature). A general expression e of the above form defines the language $\mathcal{L}(e) \subseteq \Sigma^+$, which is inductively defined as follows: (i) $\mathcal{L}(\emptyset) = \emptyset$; (ii) $\mathcal{L}(a) = \{a\}$; (iii) $\mathcal{L}(\neg e) = \Sigma^+ \setminus \mathcal{L}(e)$; (iv) $\mathcal{L}(e_1 + e_2) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$; (v) $\mathcal{L}(e_1 \cdot e_2) = \{w_1 \cdot w_2 : w_1 \in \mathcal{L}(e_1), w_2 \in \mathcal{L}(e_2)\}$.

In [28], Stockmeyer proves that the problem of deciding non-emptiness of $\mathcal{L}(e)$, for a given general expression e, is non-elementary hard. Here, we focus our attention on the following class of restricted expressions, called *prefix/suffix expressions*:

$$e ::= \emptyset \mid a \mid \neg e \mid e + e \mid \ \mathrm{Pre}(e) \mid \ \mathrm{Suf}(e) \mid \mathrm{Inf}(e), \text{ for any } a \in \Sigma,$$

where $\mathrm{Pre}(e)$ and $\mathrm{Suf}(e)$ are, respectively, a shorthand for $e \cdot (\neg\emptyset)$ and $(\neg\emptyset) \cdot e$, while $\mathrm{Inf}(e)$ is a shorthand for $\mathrm{Pre}(e) + \mathrm{Suf}(e) + \mathrm{Pre}(\mathrm{Suf}(e))$. An *infix/suffix* (resp., *infix/prefix*) expression is obtained by a prefix/suffix expression by disallowing the prefix operator Pre (resp., the suffix operator Suf). Assuming that $\Sigma = 2^{\mathcal{AP}}$, every suffix/prefix (resp., infix/suffix, resp., infix/prefix) expression e can be mapped into an equivalent formula ψ_e of $\mathsf{BE}_{\mathcal{H}om}$ (resp., $\mathsf{DE}_{\mathcal{H}om}$, resp., $\mathsf{BD}_{\mathcal{H}om}$) by applying the usual constructions for empty language, letters, negation, and union, plus the following three rules: (i) $\psi_{\mathrm{Pre}(e)} = \langle \mathrm{B} \rangle\, \psi_e$, (ii) $\psi_{\mathrm{Suf}(e)} = \langle \mathrm{E} \rangle\, \psi_e$, and (iii) $\psi_{\mathrm{Inf}(e)} = \langle \mathrm{D} \rangle\, \psi_e$. It is well known that LTL over finite words characterizes the class of languages defined by general expressions [12]. Since over finite words, LTL and $\mathsf{BE}_{\mathcal{H}om}$ have the same expressiveness [4], prefix/suffix expressions and general expressions have the same expressiveness as well. On the other hand, by Theorem 1, infix/suffix (resp., infix/prefix) expressions are less expressive than general expressions.

4 Satisfiability and model checking of $\mathsf{DE}_{\mathcal{H}om}$ over finite linear orders

In this section, we provide an automata-theoretic approach for solving satisfiability and model checking for $\mathsf{DE}_{\mathcal{H}om}$-formulas over finite linear orders. The proposed approach generalizes in a non-trivial manner the classical automata construction [29] for standard LTL over finite words based on the notion of Hintikka sequence. Given a $\mathsf{DE}_{\mathcal{H}om}$-formula φ and a non-empty finite word w over $2^{\mathcal{AP}}$, we associate to each interval $[i, j]$ of w a maximal propositionally consistent set of formulas (φ-*atom*) in the syntactical closure $\mathrm{CL}(\varphi)$ of φ which, intuitively, represents the set of formulas in $\mathrm{CL}(\varphi)$ which hold at the interval $[i, j]$. The syntactical definition of φ-atom locally captures the semantics of the Boolean connectives. In order to capture the

semantics of the temporal modalities and the homogeneity assumption, we define syntactical "semi-local" rules which allow (i) to specify in a functional way the atom associated to a non-singleton interval I in terms of the atoms associated to the two proper maximal sub-intervals of I, and (ii) to enforce "termination" conditions on the atoms associated with singleton intervals of w. Next, for each prefix w_p of w, we consider the sequence of φ-atoms, called *row*, associated with the suffixes of w_p ordered for increasing values of the length (note that in the automata-theoretic approach for LTL, the notion of row collapses to the atom associated to the current position of the given finite word). The previous syntactical rules guarantee monotonicity properties on the atoms of a row and the existence of a functional relation that given the row of a proper prefix w_p of w and the uniquely determined atom of the singleton interval associated to position $|w_p|$ of w, provides the row for the prefix of w leading to position $|w_p|$ (see Section 4.1). As a main technical step (see Section 4.2), by exploiting the monotonicity of rows, we deduce for the given $\mathsf{DE}_{\mathcal{H}om}$-formula, the existence of an equivalence relation on the set of rows of exponential-size index satisfying three fundamental properties: (i) the equivalence relation preserves the set of atoms visited by a row and their relative ordering along the row, (ii) each equivalence class has a minimal representative whose length is polynomial in the size of the given formula, and (iii) the functional relation crucially preserves the equivalence between rows. The previous three properties (i)–(iii) lead to the construction in singly exponential time of a DFA having as states the set of *minimal rows* and accepting the non-empty finite words over $2^{\mathcal{AP}}$ which satisfy the given formula (see Section 4.3). We now proceed with the technical details.

Given a DE-formula φ, we define the *closure of* φ, denoted by $\mathrm{CL}(\varphi)$, as the set of all sub-formulas ψ of φ and of their negations $\neg\psi$ (we identify $\neg\neg\psi$ with ψ). A *φ-atom* A is a subset of $\mathrm{CL}(\varphi)$ such that:

- for every $\psi \in \mathrm{CL}(\varphi)$, $\psi \in A$ if and only if $\neg\psi \notin A$, and
- for every $\psi_1 \vee \psi_2 \in \mathrm{CL}(\varphi)$, $\psi_1 \vee \psi_2 \in A$ if and only if $A \cap \{\psi_1, \psi_2\} \neq \emptyset$.

We denote the set of all φ-atoms by $\mathcal{A}_\varphi$; its cardinality is clearly bounded by $2^{|\varphi|}$. We now consider non-empty finite words over $2^{\mathcal{AP}}$ equipped with a mapping assigning to each interval a φ-atom.

▶ **Definition 2** (φ-word structures and fulfilling φ-word structures). *Let φ be a* DE*-formula. A* φ-word structure *$\mathcal{W}$ is a pair $\mathcal{W} = (w, \mathcal{L})$ consisting of a non-empty finite word over $2^{\mathcal{AP}}$ and a mapping $\mathcal{L}$ assigning to each interval of w (i.e., an interval in the homogeneous interval model $\mathcal{M}(w)$) a φ-atom such that for each position $0 \leq i < |w|$, $\mathcal{L}([i,i]) \cap \mathcal{AP} = w[i]$. The φ-word structure $\mathcal{W} = (w, \mathcal{L})$ is* fulfilling *if for each interval I of w (we also say that I is an interval of $\mathcal{W}$) and for each $\psi \in \mathrm{CL}(\varphi)$ it holds that $\psi \in \mathcal{L}(I)$ if and only if $\mathcal{M}(w), I \models \psi$.*

Evidently, for each non-empty finite word w over $2^{\mathcal{AP}}$, there exists a unique fulfilling φ-word structure associated with w. Let $\mathcal{W} = (w, \mathcal{L})$ be a φ-word structure. For each interval $[i, j]$ of $\mathcal{W}$, we write $\mathcal{L}(i, j)$ to mean $\mathcal{L}([i, j])$. For each $0 \leq i < |w|$, the *i-row of $\mathcal{W}$* is the sequence row_i of φ-atoms given by $row_i = \mathcal{L}(i, i) \cdots \mathcal{L}(0, i)$, i.e., the sequence of atoms labeling the suffixes of the prefix of w until position i ordered for increasing values of the length.

4.1 Characterization of fulfilling φ-word structures

In this section, for the given $\mathsf{DE}_{\mathcal{H}om}$-formula φ, we provide a characterization of fulfilling φ-word structures $\mathcal{W}$ in terms of a "syntactical" functional relation between adjacent $\mathcal{W}$-rows. For a φ-atom A and $X \in \{D, E\}$, we consider the following sets:

- $Req_X(A) := \{\psi \in \mathrm{CL}(\varphi) : \langle \mathrm{X} \rangle\, \psi \in A\}$ (*temporal requests* of A);
- $Obs_X(A) := \{\psi \in A : \langle \mathrm{X} \rangle\, \psi \in \mathrm{CL}(\varphi)\}$ (*observables* of A).

The next proposition, stating that, once the proposition letters of A and its temporal requests have been fixed, A gets unambiguously determined, can be easily proved by induction.

▶ **Proposition 3.** *Let φ be a* DE*-formula. Given a set $R_D \subseteq \{\psi \mid \langle \mathrm{D} \rangle \psi \in \mathrm{CL}(\varphi)\}$, a set $R_E \subseteq \{\psi \mid \langle \mathrm{E} \rangle \psi \in \mathrm{CL}(\varphi)\}$, and a set $P \subseteq \mathrm{CL}(\varphi) \cap \mathcal{AP}$, there exists a unique φ-atom A that satisfies $Req_D(A) = R_D$, $Req_E(A) = R_E$, and $A \cap \mathcal{AP} = P$.*

▶ **Definition 4.** *Let A_B and A_E be two φ-atoms. We denote by $succ_\varphi(A_B, A_E)$ the unique φ-atom A whose sets of propositions and (sub-interval and suffix) temporal requests satisfy:*

(i) $A \cap \mathcal{AP} = A_B \cap A_E \cap \mathcal{AP}$,
(ii) $Req_D(A) = Req_D(A_B) \cup Obs_D(A_B) \cup Req_D(A_E) \cup Obs_D(A_E)$, *and*
(iii) $Req_E(A) = Req_E(A_E) \cup Obs_E(A_E)$.

Definition 4 can be exploited to label a fulfilling φ-word structure $\mathcal{W}$, namely, to determine the φ-atoms labeling all the intervals $[i, j]$ of $\mathcal{W}$, starting from the singleton ones. The idea is the following: if two φ-atoms A_B and A_E label respectively the greatest proper prefix $[i, j-1]$ and the greatest proper suffix $[i+1, j]$ of the same non-singleton interval $[i, j]$, then the atom A labeling interval $[i, j]$ is unique, and it is precisely the one given by $succ_\varphi(A_B, A_E)$. The next lemma proves that this is the general rule for labeling fulfilling φ-word structures (a proof of Lemma 5 is in Appendix B).

▶ **Lemma 5.** *Let $\mathcal{W} = (w, \mathcal{L})$ be a φ-word structure. Then $\mathcal{W}$ is fulfilling* if and only if *for each interval $[i, j]$ of $\mathcal{W}$, it holds that (i) $\mathcal{L}(i, j) = succ_\varphi(\mathcal{L}(i, j-1), \mathcal{L}(i+1, j))$, if $i < j$, and (ii) $Req_D(\mathcal{L}(i, j)) = \emptyset$ and $Req_E(\mathcal{L}(i, j)) = \emptyset$, if $i = j$.*

We now introduce the abstract notion of *φ-rows*, finite sequences of φ-atoms satisfying "syntactical" adjacency requirements which capture the behaviour of $\mathcal{W}$-rows in fulfilling φ-word structures $\mathcal{W}$.

▶ **Definition 6.** *A* φ-row *row is a non-empty finite sequence of φ-atoms such that for all $0 \leq i < |row| - 1$: (i) $(row[i] \cap \mathcal{AP}) \supseteq (row[i+1] \cap \mathcal{AP})$, (ii) $Req_D(row[i+1]) \supseteq Req_D(row[i]) \cup Obs_D(row[i])$, and (iii) $Req_E(row[i+1]) = Req_E(row[i]) \cup Obs_E(row[i])$. The φ-row row is* initialized *if $Req_D(row[0]) = \emptyset$ and $Req_E(row[0]) = \emptyset$.*

We denote by $\mathcal{R}ows_\varphi$ the set of all possible φ-rows. We observe that the sequence of atoms along a φ-row $row = A_0 \cdots A_n$ has a monotonic behaviour, and the number of distinct atoms in row is linearly bounded in the size of φ. Indeed, by Definition 6, row induces three monotonic sequences: (i) one concerns the atomic propositions, is decreasing and is given by $(A_0 \cap \mathcal{AP}) \supseteq (A_1 \cap \mathcal{AP}) \supseteq \ldots \supseteq (A_n \cap \mathcal{AP})$, (ii) the second and the third are increasing, concern the temporal requests, and are given by $Req_D(A_0) \subseteq Req_D(A_1) \subseteq \ldots \subseteq Req_D(A_n)$ and $Req_E(A_0) \subseteq Req_E(A_1) \subseteq \ldots \subseteq Req_E(A_n)$. The number of distinct elements in each sequence is bounded by $|\varphi|$ (w.l.o.g, we assume that $|\mathcal{AP}| \leq |\varphi|$, i.e. we can consider only the propositional letters actually occurring in φ). Since a set of requests and a set of proposition letters uniquely determine a φ-atom, any φ-row may feature at most $3|\varphi|$ distinct atoms, i.e., $n \leq 3|\varphi|$. Since in a fulfilling φ-word structure there are no temporal requests in the atoms labeling the singleton intervals, by Lemma 5, we have the following result.

▶ **Lemma 7.**

1. *The number of distinct atoms in a φ-row row is at most $3|\varphi|$. Moreover, for all $0 \leq i < j < |row|$, if $A_i = A_j$, then $A_k = A_i$ for all $k \in [i, j]$.*
2. *Each $\mathcal{W}$-row of a fulfilling φ-word structure $\mathcal{W}$ is an initialized φ-row.*

We now generalize the successor function $succ_\varphi$ to φ-rows: given a φ-row row and a φ-atom A, we consider the φ-row of length $|row|+1$ and first atom A obtained by a component-wise application of $succ_\varphi$ starting from A and the first atom of row.

▶ **Definition 8.** *Given a φ-atom A and a φ-row row with $|row| = n$, the* A-successor of row, *denoted by $succ_\varphi(row, A)$, is the sequence $B_0 \dots B_n$ of φ-atoms defined as follows: $B_0 = A$ and $B_{i+1} = succ_\varphi(row[i], B_i)$ for all $i \in [0, n-1]$.*

By Definitions 4 and 8, we can easily derive the following lemma.

▶ **Lemma 9.** *Let row be a φ-row and A be a φ-atom. Then, $succ_\varphi(row, A)$ is a φ-row. Moreover, if row is of the form $row = row_1 \cdot row_2$, then $succ_\varphi(row, A) = succ_\varphi(row_1, A) \star succ_\varphi(row_2, A_1)$, where A_1 is the last φ-atom of $succ_\varphi(row_1, A)$.*

By Lemma 5, consecutive rows in fulfilling φ-word structures respect the successor function. In particular, by Lemmata 5 and 7, we obtain the following characterization result.

▶ **Corollary 10** (Characterization of fulfilling φ-word structures)**.** *Let $\mathcal{W} = (w, \mathcal{L})$ be a φ-word structure such that for all $0 \le i < |w|$, $Req_D(\mathcal{L}(i,i)) = \emptyset$ and $Req_E(\mathcal{L}(i,i)) = \emptyset$. Then $\mathcal{W}$ is fulfilling* if and only if*, for each $0 \le j < |w| - 1$, $row_{j+1} = succ_\varphi(row_j, row_{j+1}[0])$, where row_i is the i-row of $\mathcal{W}$ for all $0 \le i < |w|$.*

4.2 Finite abstractions of rows

We describe now the core of the proposed automata-theoretic approach to the satisfiability and model checking problems for $\mathsf{DE}_{\mathcal{H}om}$. Given a $\mathsf{DE}_{\mathcal{H}om}$ formula φ, we introduce an equivalence relation $\sim_\varphi$ of finite index over $\mathcal{R}ows_\varphi$, whose number of equivalence classes is singly exponential in the size of φ and such that each equivalence class has a representative whose length is polynomial in the size of φ. As a crucial result we show that the successor function preserves the equivalence between φ-rows. The equivalence relation $\sim_\varphi$ is based on the notion of *uniform factorization of φ-rows* and *rank of φ-atoms*. In the following, we denote by $N_{D,\varphi}$ the number of sub-interval temporal requests in $\mathrm{CL}(\varphi)$ plus one, i.e., $|\{\psi \mid \langle \mathrm{D} \rangle \psi \in \mathrm{CL}(\varphi)\}| + 1$, and by $N_{E,\varphi}$ the number of suffix temporal requests in $\mathrm{CL}(\varphi)$ plus one, i.e., $|\{\psi \mid \langle \mathrm{E} \rangle \psi \in \mathrm{CL}(\varphi)\}| + 1$. Note that $1 \le N_{D,\varphi} \le |\varphi|$ and $1 \le N_{E,\varphi} \le |\varphi|$.

▶ **Definition 11** (Uniform φ-rows)**.** *A φ-row row is* uniform *if for all $0 \le i < |row| - 1$, $(row[i] \cap \mathcal{AP}) = (row[i+1] \cap \mathcal{AP})$ and $Req_D(row[i+1]) = Req_D(row[i])$.*

Thus, in a uniform φ-row row, all the atoms occurring in row have the same propositional letters and the same sub-interval temporal requests. We represent an arbitrary φ-row row in the form $row = row_1 \cdot \ldots \cdot row_k$ (*uniform factorization of row*) where $row_1, \ldots, row_k$ are uniform φ-rows and $row_i \cdot row_{i+1}[0]$ is not uniform for all $1 \le i < k$. By the monotonicity properties of a φ-row row (see Definition 6 and Lemma 7(1)), the number k of uniform segments in the factorization of row is linearly bounded in the size of φ.

▶ **Lemma 12.** *The following statements hold:*

1. *Let row be a φ-row with uniform factorization $row_1 \cdot \ldots \cdot row_k$. Then k is at most $3|\varphi|$.*
2. *Let row be a uniform φ-row such that $|row| > N_{E,\varphi}$. Then row is of the form $row = row' \cdot B^m$ where $|row'| = N_{E,\varphi}$, $m \ge 1$, and B is the last atom of row'.*
3. *Given a φ-atom A and an integer $n \ge 1$, there is at most one uniform φ-row row such that $row[0] = A$ and $|row| = n$.*

Proof. By Lemma 7(1), the number k of uniform segments in the uniform factorization of the φ-row row is at most the number of distinct atoms in row. Hence, k is at most $3|\varphi|$, and Property (1) directly follows. As for Property (2), let row be a uniform φ-row such that $|row| > N_{E,\varphi}$. By definitions of φ-row and uniform φ-row, for all $0 \leq i < |row| - 1$, *either* $Req_E(row[i]) \subset Req_E(row[i+1])$ *or* $row[j] = row[i]$ for all $i \leq j < |row|$. Thus, since for a φ-atom A, $0 \leq |Req_E(A)| < N_{E,\varphi}$, we obtain that row is of the form $row = row' \cdot B^m$ where $|row'| = N_{E,\varphi}$, $m \geq 1$, and B is the last atom of row'.

As for Property (3), it suffices to observe that in a uniform φ-row all the atoms have the same propositional letters and the same sub-interval temporal requests. Thus, since the suffix temporal requests of a non-first atom in a φ-row are completely specified by the previous atom along the row, the result follows. ◀

We now introduce the notion of *rank* of a φ-atom. We first define the *D-rank of a φ-atom* A, written $rank_D(A)$, as $N_{D,\varphi} - |Req_D(A)|$. Clearly, $1 \leq rank_D(A) \leq |\varphi|$. The *rank of a φ-atom* A, written $rank(A)$, is defined as $rank_D(A) \cdot N_{E,\varphi}$ (i.e. the product of the D-rank of A with the increment of the overall number of suffix temporal requests in φ). Clearly, $1 \leq rank(A) \leq |\varphi|^2$. Intuitively, we can see the rank of an atom as the "number of degrees of freedom" that it gives to the atoms that stay "above it". In particular, by Definition 6, for every φ-row $row = A_0 \cdots A_n$, we have $rank_D(A_0) \geq \ldots \geq rank_D(A_n)$ and $rank(A_0) \geq \ldots \geq rank(A_n)$. Moreover, in a uniform φ-row row, all the atoms occurring in row have the same rank, and we write $rank(row)$ for such a rank.

▶ **Definition 13** (Equivalence relation). *Given two* uniform *φ-rows row and row', we say that row and row' are* equivalent*, written $row \sim_\varphi row'$, if the following conditions hold:*

- *$row[0] = row'[0]$ (hence $rank(row) = rank(row')$), and*
- *either $|row| = |row'|$ or both $|row|$ and $|row'|$ are strictly greater than $rank(row)$.*

Two arbitrary φ-rows row and row' with uniform factorizations $row_1 \cdot \ldots row_k$ and $row'_1 \cdot \ldots row'_{k'}$, respectively, are equivalent*, written $row \sim_\varphi row'$, if $k = k'$ and $row_i \sim_\varphi row'_i$ for all $i \in [1, k]$. A* minimal *φ-row is a φ-row whose uniform factorization $row_1 \cdot \ldots row_k$ is such that $|row_i| \in [1, rank(row_i) + 1]$, for each $1 \leq i \leq k$.*

By construction and Lemma 12, the number of minimal φ-rows is finite and each equivalence class of $\sim_\varphi$ contains a unique minimal φ-row. Thus, the equivalence relation $\sim_\varphi$ has finite index coinciding with the number of minimal φ-rows. This number is roughly bounded by the number of all the possible uniform factorizations of the form $row_1 \cdot \ldots \cdot row_k$ where $k \leq 3|\varphi|$ and for all $i \in [1, k]$, $|row_i|$ ranges from 1 to $|\varphi|^2$ and row_i is the unique uniform φ-row of length $|row_i|$ having as first atom $row_i[0]$. Since the number of possible φ-atoms is $2^{|\varphi|}$, the number of distinct equivalence classes of $\sim_\varphi$ is bounded by $(2^{|\varphi|} \cdot |\varphi|^2)^{3|\varphi|} \leq 2^{9|\varphi|^2}$, which is exponential in the length of the input formula φ. Moreover, each minimal φ-row has length at most $3|\varphi|^3$. Hence, we obtain the following result.

▶ **Lemma 14.** *Each equivalent class of $\sim_\varphi$ contains a unique minimal φ-row. The length of a minimal φ-row is at most $3|\varphi|^3$, and the number of minimal φ-rows is at most $2^{9|\varphi|^2}$.*

We observe that if we replace a segment (sub-row) of a φ-row by an equivalent one, we obtain a φ-row which is equivalent to the original one (for a proof, see Appendix C).

▶ **Lemma 15.** *Let $row_1, row'_1, row_2, row'_2$ be φ-rows such that $row_1 \sim_\varphi row'_1$ and $row_2 \sim_\varphi row'_2$. If $row_1 \star row_2$ and $row'_1 \star row'_2$ are defined, then $row_1 \star row_2 \sim_\varphi row'_1 \star row'_2$.*

We now show that the successor function $succ_\varphi$ on φ-rows preserves the equivalence of φ-rows. We first show (Lemma 16) that the result holds for uniform φ-rows (the proof is provided in Appendix D), and then we generalize Lemma 16 to arbitrary φ-rows.

▶ **Lemma 16.** *Let A be a φ-atom. Then the following statements hold:*

1. *let row be a uniform φ-row such that $|row| > rank(row)$. Then the φ-row $succ_\varphi(row, A)$ is of the form $A \cdot row_1 \cdot \ldots \cdot row_k$ for some $k \geq 1$ such that $row_1, \ldots, row_k$ are uniform φ-rows and $|row_k| > rank(row_k)$.*
2. *Let row and row' be two uniform φ-rows such that $row \sim_\varphi row'$. Then $succ_\varphi(row, A) \sim_\varphi succ_\varphi(row', A)$.*

▶ **Lemma 17.** *Let A be a φ-atom and row and row' be two φ-rows such that $row \sim_\varphi row'$. Then for the φ-rows $succ_\varphi(row, A)$ and $succ_\varphi(row', A)$, it holds that $succ_\varphi(row, A) \sim_\varphi succ_\varphi(row', A)$.*

Proof. The proof is by induction on the number $N(row)$ of distinct uniform segments in the uniform factorization of row. Being row and row' equivalent, $N(row') = N(row)$.

Base step: $N(row) = N(row') = 1$, i.e. row and row' are uniform. In this case, the result directly follows from Lemma 16.

Inductive step: $N(row) = N(row') > 1$. Hence, being $row \sim_\varphi row'$, row (resp., row') can be written in the form $row = row_1 \cdot row_2$ (resp., $row' = row'_1 \cdot row'_2$) such that $row_1 \sim_\varphi row'_1$, $row_2 \sim_\varphi row'_2$, $N(row_1) = N(row'_1) < N(row) = N(row')$, and $N(row_2) = N(row'_2) < N(row) = N(row')$. Let A_1 (resp., A'_1) be the last atom in $succ_\varphi(row_1, A)$ (resp., $succ_\varphi(row'_1, A)$). By the induction hypothesis, $succ_\varphi(row_1, A) \sim_\varphi succ_\varphi(row'_1, A)$, $A_1 = A'_1$, and $succ_\varphi(row_2, A_1) \sim_\varphi succ_\varphi(row'_2, A'_1)$ (note that by Lemma 12, two equivalent φ-rows have the same last atom). By Lemma 9, $succ_\varphi(row, A) = succ_\varphi(row_1, A) \star succ_\varphi(row_2, A_1)$ and $succ_\varphi(row', A) = succ_\varphi(row'_1, A) \star succ_\varphi(row'_2, A'_1)$. Thus, by applying Lemma 15, we obtain that $succ_\varphi(row, A) \sim_\varphi succ_\varphi(row', A)$, and we are done. ◀

4.3 Optimal upper bounds for $\mathsf{DE}_{\mathcal{H}om}$ satisfiability and model-checking

In this subsection, by exploiting Corollary 10 and Lemma 17, we derive an asymptotically optimal automata-theoretic approach for satisfiability and model checking of $\mathsf{DE}_{\mathcal{H}om}$ over finite linear orders. Given a $\mathsf{DE}_{\mathcal{H}om}$-formula φ, we show that it is possible to construct a deterministic finite automaton (DFA) $\mathcal{D}_\varphi$ over the alphabet $2^{\mathcal{AP}}$ having as states the initialized minimal φ-rows which accepts the non-empty finite words over $2^{\mathcal{AP}}$ which satisfy formula φ.

▶ **Definition 18.** *Let row be a minimal φ-row and A an atom. We denote by $succ^{min}_\varphi(row, A)$ the unique minimal φ-row in the equivalence class of $\sim_\varphi$ containing $succ_\varphi(row, A)$. Moreover, for a set $P \subseteq \mathcal{AP}$ of proposition letters, we denote by $A(P)$ the unique φ-atom such that $A(P) \cap \mathcal{AP} = P$, $Req_D(A(P)) = \emptyset$, and $Req_E(A(P)) = \emptyset$.*

We associate with the formula φ the DFA $\mathcal{D}_\varphi = \langle 2^{\mathcal{AP}}, \mathcal{R}ows^{min}_\varphi \cup \{q_0\}, \{q_0\}, \delta, F\rangle$, where $\mathcal{R}ows^{min}_\varphi$ is the set of initialized minimal φ-rows, and δ and F are defined as follows:

- *$\delta(q_0, P) = A(P)$ for all $P \in 2^{\mathcal{AP}}$,*
- *$\delta(row, P) = succ^{min}_\varphi(row, A(P))$ for all $P \in 2^{\mathcal{AP}}$ and $row \in \mathcal{R}ows^{min}_\varphi$;*
- *F is the set of φ-rows $row \in \mathcal{R}ows^{min}_\varphi$ such that $\varphi \in row[n-1]$, where $n = |row|$.*

We now establish the main technical result of this paper.

▶ **Theorem 19.** *Given a $\mathsf{DE}_{\mathcal{H}om}$-formula φ, the DFA $\mathcal{D}_\varphi$ accepts all and only the non-empty finite words over $2^{\mathcal{AP}}$ which satisfy φ.*

Proof. Let w be a non-empty finite word over $2^{\mathcal{AP}}$ and $n = |w| - 1$. We need to show that for the homogeneous interval model $\mathcal{M}(w)$, $\mathcal{M}(w), [0, n] \models \varphi$ if and only if $w \in \mathcal{L}(\mathcal{D}_\varphi)$.

(⇒). Assume that $\mathcal{M}(w), [0,n] \models \varphi$. Let $\mathcal{W} = (w, \mathcal{L})$ be the unique fulfilling φ-word structure associated with the word w and for all $i \in [0,n]$, let row_i be the initialized φ-row corresponding to the i-row of $\mathcal{W}$. By hypothesis $\varphi \in row_n[n]$, and by construction $|row_0| = 1$ and $row_i[0] = A(w[i])$ for all $i \in [0,n]$. Moreover, by Corollary 10, $row_{i+1} = succ_\varphi(row_i, row_{i+1}[0])$ for all $i \in [0, n-1]$. For each $i \in [0,n]$, let row_i^{min} be the unique minimal φ-row in the equivalence class $[row_i]_{\sim_\varphi}$. Note that $row_0^{min} = row_0$, the last atom of row_n^{min} contains φ, row_i^{min} is initialized and $row_i^{min}[0] = row_i[0] = A(w[i])$ for all $i \in [0,n]$. By applying Lemma 17, $succ_\varphi(row_i^{min}, row_{i+1}[0])$ is equivalent to $row_{i+1} = succ_\varphi(row_i, row_{i+1}[0])$ for all $i \in [0, n-1]$. Hence, by the definition of $succ_\varphi^{min}$, we obtain that $row_{i+1}^{min} = succ_\varphi^{min}(row_i^{min}, A(w[i+1]))$ for all $i \in [0, n-1]$. By Definition 18, it follows that there is an accepting run of $\mathcal{D}_\varphi$ over w, i.e. $w \in \mathcal{L}(\mathcal{D}_\varphi)$.

(⇐). Let us assume that w is accepted by $\mathcal{D}_\varphi$. By Definition 18, there exist $n+1$ minimal initialized minimal φ-rows $row_0^{min}, \ldots, row_n^{min}$ such that $row_0^{min} = A(w[0])$, φ belongs to the last atom of row_n^{min}, $row_i^{min}[0] = A(w[i])$ for all $i \in [0,n]$, and $row_{i+1}^{min} = succ_\varphi^{min}(row_i^{min}, row_{i+1}^{min}[0])$ for all $i \in [0, n-1]$. Let $row_0, \ldots, row_n$ be the sequence of φ-rows defined as follows: $row_0 = row_0^{min}$ and $row_{i+1} = succ_\varphi(row_i, row_{i+1}^{min}[0])$ for all $i \in [0, n-1]$. By Lemma 17, we have $row_i \sim_\varphi row_i^{min}$ for all $i \in [0,n]$. Hence, row_i is initialized for all $i \in [0,n]$, and $\varphi \in row_n[n]$. Let us define the φ-word structure $\mathcal{W} = (w, \mathcal{L})$ where $\mathcal{L}(i,j) = row_j[j-i]$ for every $0 \leq i \leq j \leq n$. By Corollary 10, $\mathcal{W}$ is fulfilling. Thus, since $\varphi \in \mathcal{L}(0,n)$, we obtain that $\mathcal{M}(w), [0,n] \models \varphi$ and the result follows. ◀

By Theorem 19, satisfiability of a $\mathsf{DE}_{\mathcal{H}om}$-formula φ reduces to checking non-emptiness of the DFA $\mathcal{D}_\varphi$ in Definition 18 whose number of states is singly exponential in the size of φ (Lemma 14). For the model-checking problem, given a finite Kripke structure $\mathcal{K}$, for checking that $\mathcal{K}$ is a model of φ, we apply the standard model-checking approach taking the synchronous product $\mathcal{K} \times \mathcal{D}_{\neg\varphi}$ of $\mathcal{K}$ with the automaton associated with the negation of the formula φ: the NFA $\mathcal{K} \times \mathcal{D}_{\neg\varphi}$ accepts all and only the traces of $\mathcal{K}$ which violate property φ. Hence, $\mathcal{K} \not\models \varphi$ if and only if the language accepted by $\mathcal{K} \times \mathcal{D}_{\neg\varphi}$ of $\mathcal{K}$ is not empty. Note that the number of states in $\mathcal{K} \times \mathcal{D}_{\neg\varphi}$ is linear in the number of $\mathcal{K}$-states and singly exponential in the size of φ, and the automata $\mathcal{D}_\varphi$ and $\mathcal{K} \times \mathcal{D}_{\neg\varphi}$ can be constructed "on the fly". Thus, since non-emptiness of NFA is in NLOGSPACE, the complexity classes NPSPACE =PSPACE and NLOGSPACE are closed under complement, and satisfiability and model checking against the fragment $\mathsf{D}_{\mathcal{H}om}$ are known to be PSPACE-complete [2], we obtain the following result.

▶ **Theorem 20.** *Finite satisfiability and model checking for* $\mathsf{DE}_{\mathcal{H}om}$*-formulas are both* PSPACE*-complete. Moreover, for* $\mathsf{DE}_{\mathcal{H}om}$*-formulas of fixed size, model checking is in* NLOGSPACE*.*

As for the logic $\mathsf{BD}_{\mathcal{H}om}$ over finite linear orders, we obtain results similar to Theorem 20. Let $\mathsf{DE}(\varphi)$ be the $\mathsf{DE}_{\mathcal{H}om}$ formula obtained from a $\mathsf{BD}_{\mathcal{H}om}$ formula φ by replacing each occurrence of modality $\langle \mathrm{B} \rangle$ with $\langle \mathrm{E} \rangle$. For each non-empty finite word w over $2^{\mathcal{AP}}$, $w \models \varphi$ iff $w^R \models DE(\varphi)$ (w^R is the reverse of w). Hence, the automaton $\mathcal{N}_\varphi$ accepting the models w of φ corresponds to the "reverse" of the DFA $\mathcal{D}_{DE(\varphi)}$ of Definition 18 associated with $DE(\varphi)$. Note that $\mathcal{N}_\varphi$ has the same states as $\mathcal{D}_{DE(\varphi)}$ but it is not deterministic. This is an important difference between $\mathsf{BD}_{\mathcal{H}om}$ and $\mathsf{DE}_{\mathcal{H}om}$ in the proposed automata-theoretic approach.

5 Results for $\mathsf{BD}_{\mathcal{H}om}$ and concluding remarks

For the logic $\mathsf{BD}_{\mathcal{H}om}$ over finite linear orders, we obtain results similar to Theorem 20. For a $\mathsf{BD}_{\mathcal{H}om}$ formula φ, let $\mathsf{DE}(\varphi)$ be the $\mathsf{DE}_{\mathcal{H}om}$ formula obtained by replacing each occurrence of modality $\langle \mathrm{B} \rangle$ with $\langle \mathrm{E} \rangle$. Evidently, for each non-empty finite word w over $2^{\mathcal{AP}}$, $w \models \varphi$ iff

$w^R \models DE(\varphi)$ (w^R is the reverse of w). Hence, the automaton $\mathcal{N}_\varphi$ accepting the models w of φ corresponds to the "reverse" of the DFA $\mathcal{D}_{DE(\varphi)}$ of Definition 18 associated with $DE(\varphi)$. Note that $\mathcal{N}_\varphi$ has the same states as $\mathcal{D}_{DE(\varphi)}$ but it is not deterministic. On the other hand, $\mathcal{N}_\varphi$ is deterministic in the *backward-direction*. Thus, for the $\mathsf{DE}_{\mathcal{H}om}$ formulas, the associated automata are deterministic in the *forward-direction* but non-deterministic in the *backward-direction*. Dually, for the $\mathsf{BD}_{\mathcal{H}om}$ formulas, the equivalent automata are deterministic in the *backward-direction* but non-deterministic in the *forward-direction*.

The results obtained for $\mathsf{DE}_{\mathcal{H}om}$ and $\mathsf{BD}_{\mathcal{H}om}$ are particularly interesting when compared with known results for $\mathsf{BE}_{\mathcal{H}om}$, where the latter includes $\mathsf{DE}_{\mathcal{H}om}$ and $\mathsf{BD}_{\mathcal{H}om}$ as proper fragments and, apparently, is quite close to $\mathsf{DE}_{\mathcal{H}om}$ and $\mathsf{BD}_{\mathcal{H}om}$. The complexity of MC for $\mathsf{BE}_{\mathcal{H}om}$ is still unknown: the problem is at least EXPSPACE-hard [5], while the only known upper bound is nonelementary [21]. Whether or not this problem can be solved elementarily is a difficult open question. Being DE and BD the most significant fragments of BE, the proved results provide a better insight into such an open question. The exact complexity of finite satisfiability for $\mathsf{BE}_{\mathcal{H}om}$ is also an open issue: the same upper/lower bounds can be shown to hold by linear-time reductions to/from the MC problem.

References

1 J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Satisfiability and Model Checking for the Logic of Sub-Intervals under the Homogeneity Assumption. In *Proc. 44th ICALP*, volume 80 of *LIPIcs*, pages 120:1–120:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.120.

3 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Model checking for fragments of the interval temporal logic HS at the low levels of the polynomial time hierarchy. *Inf. Comput.*, 262(Part):241–264, 2018. doi:10.1016/j.ic.2018.09.006.

4 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval vs. Point Temporal Logic Model Checking: An Expressiveness Comparison. *ACM Trans. Comput. Log.*, 20(1):4:1–4:31, 2019. URL: https://dl.acm.org/citation.cfm?id=3281028.

5 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Which fragments of the interval temporal logic HS are tractable in model checking? *Theor. Comput. Sci.*, 764:125–144, 2019. doi:10.1016/j.tcs.2018.04.011.

6 L. Bozzelli, A. Montanari, and A. Peron. Complexity analysis of a unifying algorithm for model checking interval temporal logic. In *Proc 26th TIME*, volume 147 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.TIME.2019.18.

7 L. Bozzelli, A. Montanari, A. Peron, and P. Sala. On a temporal logic of prefixes and infixes. In *Proc. 45th MFCS*, volume 170 of *LIPIcs*, pages 21:1–21:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.21.

8 D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. The dark side of interval temporal logic: marking the undecidability border. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):41–83, 2014. doi:10.1007/s10472-013-9376-4.

9 D. Bresolin, V. Goranko, A. Montanari, and P. Sala. Tableaux for Logics of Subinterval Structures over Dense Orderings. *Journal of Logic and Computation*, 20(1):133–166, 2010. doi:10.1093/logcom/exn063.

10 D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood logics: Expressiveness, decidability, and undecidable extensions. *Annals of Pure and Applied Logic*, 161(3):289–304, 2009. doi:10.1016/j.apal.2009.07.003.

11 D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Interval Temporal Logics: a Journey. *Bulletin of the EATCS*, 105:73–99, 2011. URL: http://albcom.lsi.upc.edu/ojs/index.php/beatcs/article/view/98.

12 V. Diekert and P. Gastin. First-order definable languages. In *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

13 E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.

14 J. Y. Halpern and Y. Shoham. A Propositional Modal Logic of Time Intervals. *Journal of the ACM*, 38:279–292, 1991. doi:10.1145/115234.115351.

15 I. Hodkinson, A. Montanari, and G. Sciavicco. Non-finite axiomatizability and undecidability of interval temporal logics with C, D, and T. In *CSL*, volume 5213 of *LNCS*, pages 308–322. Springer, 2008. doi:10.1007/978-3-540-87531-4_23.

16 K. Lodaya. Sharpening the Undecidability of Interval Temporal Logic. In *Proc. 6th ASIAN*, LNCS 1961, pages 290–298. Springer, 2000. doi:10.1007/3-540-44464-5_21.

17 A. Lomuscio and J. Michaliszyn. An Epistemic Halpern-Shoham Logic. In *Proc. 23rd IJCAI*, pages 1010–1016. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6632.

18 A. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *Proc. 21st ECAI*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 543–548. IOS Press, 2014. doi:10.3233/978-1-61499-419-0-543.

19 A. Lomuscio and J. Michaliszyn. Model checking multi-agent systems against epistemic HS specifications with regular expressions. In *Proc. 15th KR*, pages 298–308. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12823.

20 J. Marcinkowski and J. Michaliszyn. The Undecidability of the Logic of Subintervals. *Fundam. Inform.*, 131(2):217–240, 2014. doi:10.3233/FI-2014-1011.

21 A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. *Acta Informatica*, 53(6-8):587–619, 2016. doi:10.1007/s00236-015-0250-1.

22 A. Molinari, A. Montanari, and A. Peron. Model checking for fragments of Halpern and Shoham's interval temporal logic based on track representatives. *Inf. Comput.*, 259(3):412–443, 2018. doi:10.1016/j.ic.2017.08.011.

23 A. Montanari. Interval temporal logics model checking. In *Proc. 23rd TIME*, page 2. IEEE Computer Society, 2016. doi:10.1109/TIME.2016.32.

24 B. Moszkowski. *Reasoning About Digital Circuits*. PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.

25 A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.

26 I. Pratt-Hartmann. Temporal propositions and their logic. *Artificial Intelligence*, 166(1-2):1–36, 2005. doi:10.1016/j.artint.2005.04.003.

27 P. Roeper. Intervals and tenses. *Journal of Philosophical Logic*, 9:451–469, 1980. doi:10.1007/BF00262866.

28 L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

29 M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.

30 Y. Venema. A Modal Logic for Chopping Intervals. *Journal of Logic and Computation*, 1(4):453–476, 1991. doi:10.1093/logcom/1.4.453.

A Proof of Theorem 1

In this section we provide a proof of Theorem 1. We establish the following result. Hence, Theorem 1 directly follows.

▶ **Proposition 21.** *Over finite linear orders, the following holds:*

1. *there exists a* $\mathsf{B}_{\mathcal{H}om}$ *formula which cannot be expressed in* $\mathsf{DE}_{\mathcal{H}om}$*;*
2. *there exists an* $\mathsf{E}_{\mathcal{H}om}$ *formula which cannot be expressed in* $\mathsf{BD}_{\mathcal{H}om}$*.*

We prove Proposition 21(2). The proof of Proposition 21(1) is similar and we omit the details here. Let $\mathcal{AP} = \{p\}$. We consider the $\mathsf{E}_{\mathcal{H}om}$ formula φ_E over $\mathcal{AP}$ defined as follows:

$$\varphi_E := \langle \mathrm{E} \rangle (p \wedge \langle \mathrm{E} \rangle \top)$$

which asserts the existence of a proper suffix of length at least 2 where p holds. In order to prove that no formula in $\mathsf{BD}_{\mathcal{H}om}$ is equivalent to φ_E over finite linear orders, we define two families $(w_n)_{n \geq 1}$ and $(w'_n)_{n \geq 1}$ of non-empty finite words over $2^{\{p\}}$ such that

- φ_E distinguishes between w_n and w'_n for each $n \geq 1$, and
- for every $\mathsf{BD}_{\mathcal{H}om}$ formula ψ, there is $n \geq 1$ such that ψ does not distinguish between w_n and w'_n.

For each $n \geq 1$, the words w_n and w'_n over $2^{\{p\}}$ are defined as follows:

$$w_n = (\emptyset \{p\}^n)^{n+2} \{p\} \text{ and } w'_n = (\emptyset \{p\}^n)^{n+2} \emptyset \{p\}$$

Note that for each $n \geq 1$, the property of having a suffix of length at least 2 where p holds is satisfied by w_n but not by w'_n. Hence, the following holds:

▶ **Lemma 22.** *For each $n \geq 1$, $w_n \models \varphi_E$ and $w'_n \not\models \varphi_E$.*

For a $\mathsf{BD}_{\mathcal{H}om}$ formula ψ, we denote by $d(\psi)$ the joint nesting depth of the temporal modalities in ψ. Proposition 21(2) directly follows from Lemma 22 and the following lemma.

▶ **Lemma 23.** *Let $n \geq 1$. Then, for each $\mathsf{BD}_{\mathcal{H}om}$ formula ψ such that $d(\psi) < n$, it holds that $w_n \models \psi$ if and only if $w'_n \models \psi$*

Proof. Fix $n \geq 1$. In order to prove Lemma 23, we need some preliminary results. The following claim can be proved by a straightforward induction on $k \geq 1$.

▷ Claim 1. Let $k \geq 1$. Then for each $\mathsf{BD}_{\mathcal{H}om}$ formula ξ such that $d(\xi) < k$, it holds that (i) $\{p\}^k \models \xi$ iff $\{p\}^{k+1} \models \xi$ and (ii) $\{p\}^j \emptyset \{p\}^k \models \xi$ iff $\{p\}^j \emptyset \{p\}^{k+1} \models \xi$ for each $j \geq 0$.

▷ Claim 2. Let $i, j \geq 0$, $k \geq 1$, and $\nu(j,i,k) = \{p\}^j (\emptyset \{p\}^n)^i \emptyset \{p\}^k$. Then for each $\mathsf{BD}_{\mathcal{H}om}$ formula ξ such that $d(\xi) < k$, $\nu(j,i,k) \models \xi$ iff $\nu(j,i,k) \cdot \{p\} \models \xi$.

Proof of Claim 2. The proof is by induction on $k \geq 1$. For the base case ($k = 1$), being $d(\xi) < 1$ (hence, ξ does not contain temporal modalities), the result trivially follows. Now, assume that $k > 1$. We proceed by a double induction on $i \geq 0$. If $i = 0$, the result directly follows from Claim 1. Now, let us assume that $i > 0$. By construction, for each proper prefix (resp., proper infix) v of $\nu(j,i,k) \cdot \{p\}$, there is a proper prefix (resp., proper infix) v' of $\nu(j,i,k)$ such that either (i) $v' = v$, or (ii) $v = \{p\}^{k+1}$ and $v' = \{p\}^k$, or (iii) $v = \nu(j',i',k') \cdot \{p\}$ and $v' = \nu(j',i',k')$ for some $j', i' \geq 0$ and $k' \geq 1$ such that either $k' = k$ and $i' < i$, or $k' = k-1$ and $i' \leq i$. Hence, the result easily follows from the induction hypothesis and Claim 1. ◁

▷ Claim 3. Let $h \geq 1$, $i, j \geq 0$, $\nu(j,h) = \{p\}^j (\emptyset \{p\}^n)^h$ and $\nu(j,h,i) = \{p\}^j (\emptyset \{p\}^n)^h \emptyset \{p\}^i$. Then, for each $\mathsf{BD}_{\mathcal{H}om}$ formula ξ such that $d(\xi) < h$, it holds that (i) $\nu(j,h) \models \xi$ iff $\nu(j,h+1) \models \xi$, and (ii) $\nu(j,h,i) \models \xi$ iff $\nu(j,h+1,i) \models \xi$.

Proof of Claim 3. The proof is by induction on $h \geq 1$. The base case ($h = 1$) trivially follows, since being $d(\xi) < 1$, ξ does not contains temporal modalities. Now, assume that $h > 1$. Let $w = \nu(j,h)$ (resp., $w = \nu(j,h,i)$) and $w' = \nu(j,h+1)$ (resp., $w' = \nu(j,h+1,i)$). We need to prove that $w \models \xi$ iff $w' \models \xi$. By construction, the following holds:

- for each proper infix of v (resp., v') of w (resp., w), there exists a proper infix of v' (resp., v) of w' (resp., w) such that v' (resp., v) is a prefix of w' (resp., w) if v (resp., v') is a prefix of w (resp., w') and either (i) $v' = v$, or (ii) $v' = \nu(j', h)$ and $v = \nu(j', h-1)$ for some $j' \geq 0$, or (iii) $v' = \nu(j', h, i')$ and $v = \nu(j', h-1, i')$ for some $i', j' \geq 0$.

Hence, the result easily follows from the induction hypothesis. ◁

By Claim 3, we easily deduce the following result.

▷ Claim 4. Let $j \geq 0$. Then for each $\mathsf{BD}_{\mathcal{H}om}$ formula ξ such that $d(\xi) < n$, $\{p\}^j(\emptyset\{p\}^n)^{n+1} \models \xi$ iff $\{p\}^j(\emptyset\{p\}^n)^{n+1}\emptyset\{p\} \models \xi$.

We now prove Lemma 23. The proof is by induction on n. The base case ($n = 1$) trivially follows, since being $d(\psi) < n$, ψ does not contains temporal modalities. Now, assume that $n > 1$. We need to show that for each $\mathsf{BE}_{\mathcal{H}om}$ formula ξ such that $d(\xi) < n-1$, the following holds:

- for each proper infix (resp., proper prefix) v of w_n, there exists a proper infix (resp., proper prefix) v' of w'_n such that $v \models \xi$ iff $v' \models \xi'$;
- for each proper infix (resp., proper prefix) v' of w'_n, there exists a proper infix (resp., proper prefix) v of w_n such that $v \models \xi$ iff $v' \models \xi'$.

The result easily follows from the definition of w_n and w'_n and Claims 1–4. ◀

B Proof of Lemma 5

▶ **Lemma 5.** *Let $\mathcal{W} = (w, \mathcal{L})$ be a φ-word structure. Then $\mathcal{W}$ is fulfilling* if and only if *for each interval $[i, j]$ of $\mathcal{W}$, it holds that (i) $\mathcal{L}(i, j) = succ_\varphi(\mathcal{L}(i, j-1), \mathcal{L}(i+1, j))$, if $i < j$, and (ii) $Req_D(\mathcal{L}(i, j)) = \emptyset$ and $Req_E(\mathcal{L}(i, j)) = \emptyset$, if $i = j$.*

Proof.

($\Rightarrow$). Assume that $\mathcal{W}$ is fulfilling. Hence, for each interval $[i, j]$ of $\mathcal{W}$, $\mathcal{L}(i, j)$ is the set of formulas $\psi \in \mathrm{CL}(\varphi)$ such that $\mathcal{M}(w), [i, j] \models \psi$ (recall that $\mathcal{M}(w)$ is the homogeneous interval model associated with the word w). Thus, if $i = j$, then $Req_D(\mathcal{L}(i, j)) = \emptyset$ and $Req_E(\mathcal{L}(i, j)) = \emptyset$. Otherwise, $i < j$ and being $\mathcal{M}(w)$ homogeneous, we have that $\mathcal{L}(i, j) \cap \mathcal{AP} = \mathcal{L}(i, j-1) \cap \mathcal{L}(i+1, j) \cap \mathcal{AP}$. Moreover, by the semantics of DE, the following holds:

- for each $\langle \mathrm{D} \rangle \psi \in \mathrm{CL}(\varphi)$, $\langle \mathrm{D} \rangle \psi \in \mathcal{L}(i, j)$ if and only if either $\langle \mathrm{D} \rangle \psi \in \mathcal{L}(i, j-1)$, or $\psi \in \mathcal{L}(i, j-1)$, or $\langle \mathrm{D} \rangle \psi \in \mathcal{L}(i+1, j)$, or $\psi \in \mathcal{L}(i+1, j)$;
- for each $\langle \mathrm{E} \rangle \psi \in \mathrm{CL}(\varphi)$, $\langle \mathrm{E} \rangle \psi \in \mathcal{L}(i, j)$ if and only if either $\langle \mathrm{E} \rangle \psi \in \mathcal{L}(i+1, j)$ or $\psi \in \mathcal{L}(i+1, j)$.

This means that $\mathcal{L}(i, j) = succ_\varphi(\mathcal{L}(i, j-1), \mathcal{L}(i+1, j))$, and the result follows.

($\Leftarrow$). Assume that for every interval $[i, j]$ of $\mathcal{W}$, we have $\mathcal{L}(i, j) = succ_\varphi(\mathcal{L}(i, j-1), \mathcal{L}(i+1, j))$ if $i < j$, and $Req_D(\mathcal{L}(i, j)) = \emptyset$ and $Req_E(\mathcal{L}(i, j)) = \emptyset$ if $i = j$. We have to prove that $\mathcal{W}$ is fulfilling. Let $[i, j]$ be an interval of $\mathcal{W}$ and $\psi \in \mathrm{CL}(\varphi)$. We prove by induction on the structure of ψ that $\psi \in \mathcal{L}(i, j)$ if and only if $\mathcal{M}(w), [i, j] \models \psi$. Hence, the result follows.

- $\psi = p$ with $p \in \mathcal{AP}$: we have to show that $\mathcal{L}(i, j) \cap \mathcal{AP} = \bigcap_{h \in [i,j]} \mathcal{L}(h, h) \cap \mathcal{AP}$. The proof is by a double induction on $j - i \geq 0$. If $i = j$, the property trivially holds. Let us assume now that $j - i > 0$. Since $\mathcal{L}(i, j) = succ_\varphi(\mathcal{L}(i, j-1), \mathcal{L}(i+1, j))$, by Condition (i) of Definition 4 and the induction hypothesis, we obtain that $\mathcal{L}(i, j) \cap \mathcal{AP} = \bigcap_{h \in [i+1,j]} \mathcal{L}(h, h) \cap \bigcap_{h \in [i,j-1]} \mathcal{L}(h, h) \cap \mathcal{AP}$. Hence, the result directly follows.

- $\psi = \neg\psi_1$ or $\psi = \psi_1 \vee \psi_2$: for these cases, the result directly follows from the induction hypothesis and the definition of φ-atom (recall that $\mathcal{L}(i,j)$ is a φ-atom).
- $\psi = \langle \mathrm{D} \rangle \psi_1$ or $\psi = \langle \mathrm{E} \rangle \psi_1$: the proof is by a double induction on $j - i \geq 0$. If $i = j$, then $\mathcal{M}(w), [i,j] \not\models \psi$, $Req_D(\mathcal{L}(i,j)) = \emptyset$, and $Req_E(\mathcal{L}(i,j)) = \emptyset$. Hence, the result follows. Now, assume that $j - i > 0$. First, let us consider the case where $\psi = \langle \mathrm{D} \rangle \psi_1$. Since $\mathcal{L}(i,j) = succ_\varphi(\mathcal{L}(i,j-1), \mathcal{L}(i+1,j))$, by Condition (ii) of Definition 4 and the induction hypothesis, we have that $\langle \mathrm{D} \rangle \psi_1 \in \mathcal{L}(i,j)$ if and only if either $\mathcal{M}(w), [i,j-1] \models \langle \mathrm{D} \rangle \psi_1$, or $\mathcal{M}(w), [i,j-1] \models \psi_1$, or $\mathcal{M}(w), [i+1,j] \models \langle \mathrm{D} \rangle \psi_1$, or $\mathcal{M}(w), [i+1,j] \models \psi_1$ if and only if $\mathcal{M}(w), [i,j] \models \langle \mathrm{D} \rangle \psi_1$.
  Now, let us consider the case where $\psi = \langle \mathrm{E} \rangle \psi_1$. By Condition (iii) of Definition 4 and the induction hypothesis, we have that $\langle \mathrm{E} \rangle \psi_1 \in \mathcal{L}(i,j)$ if and only if either $\mathcal{M}(w), [i+1,j] \models \langle \mathrm{E} \rangle \psi_1$ or $\mathcal{M}(w), [i+1,j] \models \psi_1$ if and only if $\mathcal{M}(w), [i,j] \models \langle \mathrm{E} \rangle \psi_1$, and the result follows.

◀

C Proof of Lemma 15

▶ **Lemma 15.** *Let $row_1, row_1', row_2, row_2'$ be φ-rows such that $row_1 \sim_\varphi row_1'$ and $row_2 \sim_\varphi row_2'$. If $row_1 \star row_2$ and $row_1' \star row_2'$ are defined, then $row_1 \star row_2 \sim_\varphi row_1' \star row_2'$.*

Proof. We consider the case where row_1 and row_2 are uniform, hence, row_1' and row_2' are uniform as well. The general case easily follows from the considered case. By hypothesis $row_1 \star row_2$ and $row_1' \star row_2'$ are defined. This entails that $row_1 \star row_2$ and $row_1' \star row_2'$ are uniform as well. Thus since $row_1 \sim_\varphi row_1'$ and $row_2 \sim_\varphi row_2'$, by Definition 13, we obtain that $row_1 \star row_2$ and $row_1' \star row_2'$ have the same first atom A and indicated by m (resp., m') the length of $row_1 \star row_2$ (resp., $row_1' \star row_2'$), it holds that either $m = m'$, or both $m > rank(A)$ and $m' > rank(A)$. Hence, the result follows. ◀

D Proof of Lemma 16

In order to prove Lemma 16, we need a preliminary technical result (Lemma 5) that considers uniform φ-rows of the form B^m for some φ-atom B.

▶ **Lemma 5.** *Let A and B be two φ-atoms such that $rank_D(succ_\varphi(B,A)) = rank_D(B) - h$ for some $h \geq 0$ (note that $h < rank_D(B)$). Given $m > (rank_D(B) - h) \cdot N_{E,\varphi}$, if B^m is a φ-row than the φ-row $succ_\varphi(B^m, A)$ is of the form $A \cdot row_1 \cdot \ldots \cdot row_k$ for some $k \geq 1$ such that $row_1, \ldots, row_k$ are uniform φ-rows and*

- *$rank_D(row_i) > rank_D(row_{i+1})$ for each $1 \leq i < k$,*
- *$|row_k| > rank(row_k)$.*

Proof. Let $rank_D(succ_\varphi(B,A)) = rank_D(B) - h$ for some $0 \leq h < rank_D(B)$, $m > (rank_D(B) - h) \cdot N_{E,\varphi}$, and row be the φ-row of length $m+1$ given by $succ_\varphi(B^m, A)$. Since $row[0] = A$ and $row[i+1] = succ_\varphi(B, row[i])$ for all $i \in [0, m-1]$, by Definition 4, for all $i \in [1, m]$, the following holds:

- $row[i] = B \cap A \cap \mathcal{AP}$;
- $rank_D(row[i-1]) \geq rank_D(row[i])$ and $Req_E(row[i-1]) \subseteq Req_E(row[i])$;
- if $i < m$ and $row[i] = row[i+1]$, then $row[j] = row[i]$ for all $j \geq i$.

Since by hypothesis $rank_D(row[1]) = rank_D(B) - h$, we easily deduce that $row = succ_\varphi(B^m, A)$ is of the form

$$A \cdot row_1 \cdot \ldots \cdot row_k$$

for some $k \geq 1$ such that $row_1, \ldots, row_k$ are uniform φ-rows and

- $rank_D(row_1) = rank_D(B) - h$,
- $rank_D(row_i) > rank_D(row_{i+1})$ for each $1 \leq i < k$, and
- $|row_i| \leq N_{E,\varphi}$ for each $1 \leq i < k$.

It remains to show that $|row_k| > rank(row_k)$. By the previous points, we have that $rank_D(B) - h = rank_D(row_1) > \ldots > rank_D(row_k)$. Hence, $rank_D(B) - h \geq rank_D(row_k) + k - 1$. Since $m > (rank_D(B) - h) \cdot N_{E,\varphi}$ and $|row_i| \leq N_{E,\varphi}$ for each $1 \leq i < k$, we obtain $|row_k| = m - \sum_{i=1}^{i=k-1} |row_i| \geq m - \sum_{i=1}^{i=k-1} N_{E,\varphi} > (rank_D(B) - h) \cdot N_{E,\varphi} - (k-1)N_{E,\varphi} \geq rank_D(row_k) \cdot N_{E,\varphi} = rank(row_k)$. ◀

By exploiting Lemma 5, we now prove Lemma 16.

▶ **Lemma 16.** *Let A be a φ-atom. Then the following statements hold:*

1. *let row be a uniform φ-row such that $|row| > rank(row)$. Then the φ-row $succ_\varphi(row, A)$ is of the form $A \cdot row_1 \cdot \ldots \cdot row_k$ for some $k \geq 1$ such that $row_1, \ldots, row_k$ are uniform φ-rows and $|row_k| > rank(row_k)$.*
2. *Let row and row' be two uniform φ-rows such that $row \sim_\varphi row'$. Then $succ_\varphi(row, A) \sim_\varphi succ_\varphi(row', A)$.*

Proof.

Proof of Property (1). Let A be a φ-atom and row be a uniform φ-row such that $|row| > rank(row)$. We need to show that the length $|\eta|$ of the last uniform segment η in the uniform factorization of $succ_\varphi(row, A)$ satisfies $|\eta| > rank(\eta)$. Since $|row| > rank(row)$ and $rank(row) \geq N_{E,\varphi}$, by Lemma 12(2), row is of the form $row = row_1 \cdot B^m$ where $m \geq 1$, $|row_1| = N_{E,\varphi}$ and B is the last atom of row_1. Let row' be the φ-row given by $succ_\varphi(row, A)$. Then row' can be written in the form

$$row' = (A \cdot row'_1) \star succ_\varphi(B^m, B')$$

where $A \cdot row'_1 = succ_\varphi(row_1, A)$ and B' is the last atom of row'_1. In particular, $|row'_1| = N_{E,\varphi}$. Let $B'' = succ_\varphi(B, B')$. By Definition 4, we have that $rank_D(B'') \leq rank_D(row'_1[0]) \leq rank_D(row)$. We distinguish two cases:

- $rank_D(B'') = rank_D(row)$. In this case, we have that all the atoms in $row'_1 \cdot B''$ have the same sub-interval temporal requests. Moreover, since row is uniform, by Definition 4, all the atoms in $row'_1 \cdot B''$ have the same propositional letters. Hence, $row'_1 \cdot B''$ is a uniform φ-row. Since $|row'_1| = N_{E,\varphi}$, by Lemma 12(2), B'' coincides with the last atom B' of row'_1. Thus, $B' = succ_\varphi(B, B')$ and $row' = A \cdot row'_1 \cdot (B')^m$ where $row'_1 \cdot (B')^m$ is a uniform φ-row having the same length and the same rank as row. Thus, since $|row| > rank(row)$, the result in this case holds.
- $rank_D(B'') < rank_D(row) = rank_D(B)$. We have that $m = |row| - N_{E,\varphi} > rank(row) - N_{E,\varphi} = (rank_D(B) - 1) \cdot N_{E,\varphi} \geq rank(B'')$. Since $B'' = succ_\varphi(B, B')$, by Lemma 5, the length $|\eta|$ of the last uniform segment η in the uniform factorization of $succ_\varphi(B^m, B')$ satisfies $|\eta| > rank(\eta)$, and the result follows.

Proof of Property (2). Let A be a φ-atom and row and row' be two uniform φ-rows such that $row \sim_\varphi row'$. We need to show that $succ_\varphi(row, A) \sim_\varphi succ_\varphi(row', A)$. By hypothesis and Definition 13, there are two cases:

- $row[0] = row'[0]$ and $|row| = |row'|$. Since row and row' are uniform, by Lemma 12(3), $row = row'$, and the result obviously follows.

- $row[0] = row'[0]$, $|row| \neq |row'|$, $|row| > rank(row)$ and $|row'| > rank(row')$. Assume that $|row| < |row'|$ (the case where $|row'| < |row|$ being similar). Since row and row' are uniform and $row[0] = row'[0]$, it holds that $rank(row) = rank(row')$. Moreover, $|row| > rank(row) \geq N_{E,\varphi}$. Applying Lemma 12(2) and Lemma 12(3), we deduce that row is of the form $row = row_1 \cdot B^2$ and $row' = row_1 \cdot B^{k+2}$ where B is a φ-atom and $k = |row'| - |row|$. By Property (1) of Lemma 16 the last uniform segment η of $succ_\varphi(row, A)$ satisfies $|\eta| > rank(\eta) \geq N_{E,\varphi}$. Thus, by Lemma 12(2), $succ_\varphi(row, A)$ is of the form $row' \cdot (B')^2$ for a φ-atom B' such that $B' = succ_\varphi(B, B')$. Since $succ_\varphi(row', A) = (row' \cdot (B')^2) \star succ_\varphi(B^k, B')$, we obtain that $succ_\varphi(row', A) = row' \cdot (B')^{2+k}$. Thus, since the last uniform segment η in $row' \cdot (B')^2$ satisfies $|\eta| > rank(\eta)$, we deduce that $succ_\varphi(row, A)$ and $succ_\varphi(row', A)$ are equivalent. ◀

Deciding FO-Rewritability of Ontology-Mediated Queries in Linear Temporal Logic

Vladislav Ryzhikov
Department of Computer Science, Birkbeck, University of London, UK

Yury Savateev
Department of Computer Science, Birkbeck, University of London, UK
HSE University, Moscow, Russia

Michael Zakharyaschev
Department of Computer Science, Birkbeck, University of London, UK
HSE University, Moscow, Russia

Abstract

Our concern is the problem of determining the data complexity of answering an ontology-mediated query (OMQ) given in linear temporal logic *LTL* over $(\mathbb{Z}, <)$ and deciding whether it is rewritable to an $\mathsf{FO}(<)$-query, possibly with extra predicates. First, we observe that, in line with the circuit complexity and FO-definability of regular languages, OMQ answering in AC^0, ACC^0 and NC^1 coincides with $\mathsf{FO}(<,\equiv)$-rewritability using unary predicates $x \equiv 0 \,(\mathrm{mod}\, n)$, $\mathsf{FO}(<,\mathsf{MOD})$-rewritability, and $\mathsf{FO}(\mathsf{RPR})$-rewritability using relational primitive recursion, respectively. We then show that deciding $\mathsf{FO}(<)$-, $\mathsf{FO}(<,\equiv)$- and $\mathsf{FO}(<,\mathsf{MOD})$-rewritability of *LTL* OMQs is ExpSpace-complete, and that these problems become PSpace-complete for OMQs with a linear Horn ontology and an atomic query, and also a positive query in the cases of $\mathsf{FO}(<)$- and $\mathsf{FO}(<,\equiv)$-rewritability. Further, we consider $\mathsf{FO}(<)$-rewritability of OMQs with a binary-clause ontology and identify OMQ classes, for which deciding it is PSpace-, Π^p_2- and coNP-complete.

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

Keywords and phrases Linear temporal logic, ontology-mediated query, first-order rewritability

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.10

Funding This work was supported by the UK EPSRC grant EP/S032282.

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Motivation. The problem we consider in this paper originates in the area of ontology-based data access (OBDA) to temporal data. The aim of the OBDA paradigm [38, 51] and systems such as Mastro or Ontop[1] is to facilitate management and integration of possibly incomplete and heterogeneous data by providing the user with a view of the data through the lens of a description logic (DL) ontology. Thus, the user can think of the data as a "virtual knowledge graph" [52], $\mathcal{A}$, whose labels – unary and binary predicates supplied by an ontology, $\mathcal{O}$ – are the only thing to know when formulating queries, $\varkappa$. Ontology-mediated queries (OMQs) $\boldsymbol{q} = (\mathcal{O}, \varkappa)$ are supposed to be answered over $\mathcal{A}$ under the open world semantics (taking account of all models of $\mathcal{O}$ and $\mathcal{A}$), which can be prohibitively complex. So the key to practical OBDA is ensuring first-order rewritability of $\boldsymbol{q}$ (aka boundedness in the datalog literature [1]), which reduces open-world reasoning to evaluating an FO-formula over $\mathcal{A}$. The

[1] `https://www.obdasystems.com`, `https://ontopic.biz`

W3C standard ontology language *OWL 2 QL* for OBDA is based on the *DL-Lite* family of DL [3,17], which uniformly guarantees FO-rewritability of all OMQs with a conjunctive query. Other ontology languages with this feature include various dialects of tgds; see, e.g., [7,16,19]. However, by design such languages are rather inexpressive.

Theory and practice of OBDA have revived the interest to the problem of deciding whether an OMQ given in some expressive language is FO-rewritable, which was thoroughly investigated in the 1980–90s for datalog queries; see, e.g., [2,21,37,44,46]. The data complexity and rewritability of OMQs in various DLs and disjunctive datalog have become an active research area in the past decade [14, 24, 28, 29, 36], lying at the crossroads of logic, database theory, knowledge representation, circuit and descriptive complexity, and CSP.

There have been numerous attempts to extend ontology and query languages with constructors capable of representing events over temporal data; see [5, 35] for surveys and [15,49,50] for more recent developments. However, so far the focus has been on the uniform complexity of reasoning with arbitrary ontologies and queries in a given language rather than on understanding the data complexity and FO-rewritability of individual temporal OMQs. On the other hand, the non-uniform analysis of OMQs in DLs or datalog mentioned above is not applicable to standard temporal logics interpreted over linearly-ordered structures.

In this paper, we take a first step towards understanding the problem of FO-rewritability of OMQs over temporal data by focusing on the temporal dimension and considering OMQs given in linear temporal logic *LTL* interpreted over $(\mathbb{Z}, <)$.

▶ **Example 1.** Let $\mathcal{O}$ be an *LTL* ontology with the following axioms (describing a system's behaviour and) containing the temporal operators $\Box_F/\Box_P$ (always in the future/past), $\Diamond_F/\Diamond_P$ (sometime in the future/past) and $\bigcirc_F/\bigcirc_P$ (the next/previous minute):

$$\Box_P\Box_F\big(\mathit{Malfunction} \to \Diamond_F \mathit{Fixed}\big), \tag{1}$$

$$\Box_P\Box_F\big(\mathit{Fixed} \to \bigcirc_F \mathit{InOperation}\big), \tag{2}$$

$$\Box_P\Box_F\big(\mathit{Malfunction} \land \bigcirc_P \mathit{Malfunction} \land \bigcirc_P^2 \mathit{Malfunction} \to \neg\bigcirc_F \mathit{InOperation}\big). \tag{3}$$

We query temporal data, say

$$\mathcal{A} = \{\mathit{Malfunction}(2), \mathit{Malfunction}(5), \mathit{Malfunction}(6), \mathit{Fixed}(6), \mathit{Malfunction}(7)\}$$

by means of *LTL*-formulas such as

$$\varkappa = \Diamond_P\Diamond_F\big(\mathit{Malfunction} \land \bigvee_{1\le i\le 5} \bigcirc_F^i(\mathit{Fixed} \land \bigvee_{1\le j\le 5} \neg\bigcirc_F^j \mathit{InOperation})\big)$$

asking whether there was a malfunction that was fixed in $\le$ 5m but within the next 5m the equipment went out of operation again. The certain answer to the OMQ $\boldsymbol{q} = (\mathcal{O}, \varkappa)$ over $\mathcal{A}$ is yes because $\varkappa$ is true in all models of $\mathcal{O}$ and $\mathcal{A}$. It is readily seen that the certain answer to $\boldsymbol{q}$ over any given data instance $\mathcal{A}'$ in the signature $\{\mathit{Malfunction}, \mathit{Fixed}\}$ can be computed by evaluating over $\mathcal{A}'$ the following FO(<)-sentence, called an FO(<)-rewriting of $\boldsymbol{q}$:

$$\exists x\,\big[\mathit{Malfunction}(x) \land \bigvee_{1\le i\le 5}\big(\mathit{Fixed}(x+i) \land \bigvee_{1\le j\le 5}\bigwedge_{0\le k\le 2} \mathit{Malfunction}(x+i+j-k)\big)\big].$$

Problem and related work. The problem we are interested in can be formulated in complexity-theoretic terms: given an *LTL* OMQ $\boldsymbol{q}$, determine the data complexity of answering $\boldsymbol{q}$ over any data instance $\mathcal{A}$ in a given signature Ξ. For simplicity's sake, let us assume that $\boldsymbol{q}$ is Boolean (with a yes/no answer). Then the data instances $\mathcal{A}$, over

which the answer to $\boldsymbol{q}$ is yes, form a language $\boldsymbol{L}(\boldsymbol{q})$ over the alphabet 2^{Ξ}. In fact, using the automata-theoretic view of *LTL* [48], one can show that $\boldsymbol{L}(\boldsymbol{q})$ is regular, and so can be decided in NC^1 [8, 10]. The circuit and descriptive complexity of regular languages was investigated in [9, 43], which established an $\mathrm{AC}^0/\mathrm{ACC}^0/\mathrm{NC}^1$ trichotomy, gave algebraic characterisations of languages in these classes (implying that the trichotomy is decidable) and also in terms of extensions of FO. Namely, the languages $\boldsymbol{L}$ in AC^0 are definable by $\mathsf{FO}(<,\equiv)$-sentences with unary predicates $x \equiv 0 \,(\mathrm{mod}\ n)$; those in ACC^0 are definable by $\mathsf{FO}(<,\mathsf{MOD})$-sentences with quantifiers $\exists^n x\, \psi(x)$ checking whether the number of positions satisfying ψ is divisible by n; and all regular languages $\boldsymbol{L}$ are definable in $\mathsf{FO}(\mathsf{RPR})$ with relational primitive recursion [20].

Thus, our problem can be equivalently formulated in logic terms: given an *LTL* OMQ $\boldsymbol{q}$, decide whether $\boldsymbol{L}(\boldsymbol{q})$ is $\mathsf{FO}(<,\equiv)$- or $\mathsf{FO}(<,\mathsf{MOD})$-definable. In the OBDA context, we are also interested in $\mathsf{FO}(<)$-definability (without any extra predicates, quantifiers or recursion), which has been thoroughly investigated in both automata theory and logic; see, e.g., [23] and references therein. In particular, deciding $\mathsf{FO}(<)$-definability of regular languages given by a NFA can be done in PSpace [13, 41], with a matching lower bound established even for languages given by a minimal DFA [18]. These classical results have recently been extended by showing that deciding each of $\mathsf{FO}(<)$-, $\mathsf{FO}(<,\equiv)$-, and $\mathsf{FO}(<,\mathsf{MOD})$-definability of languages given by a two-way NFA can be done in PSpace, and that a matching lower bound holds for languages given by a minimal DFA [33]. Note also that, by Kamp's Theorem [30, 39], $\mathsf{FO}(<)$-rewritability reduces answering *LTL* OMQs to model checking *LTL*-formulas.

$\mathsf{FO}(\mathsf{RPR})$-rewritability of all *LTL* OMQs was proved in [6], which also provided (uniform) rewritability results for various classes of *LTL* OMQs (to be defined below); see Table 2.

Our contribution. Let $\mathcal{L} \in \{\mathsf{FO}(<), \mathsf{FO}(<,\equiv), \mathsf{FO}(<,\mathsf{MOD})\}$. To investigate $\mathcal{L}$-rewritability of *LTL* OMQs $\boldsymbol{q} = (\mathcal{O}, \varkappa)$, we follow the classification of [6], according to which the axioms of every *LTL* ontology $\mathcal{O}$ are given in the clausal form

$$\Box_P \Box_F \big(C_1 \wedge \dots \wedge C_k \ \to\ C_{k+1} \vee \dots \vee C_{k+m}\big), \tag{4}$$

where the C_i are atoms, possibly prefixed by the temporal operators $\bigcirc_F$, $\bigcirc_P$, $\Box_F$, $\Box_P$. Given some $\boldsymbol{o} \in \{\Box, \bigcirc, \Box\bigcirc\}$ and $\boldsymbol{c} \in \{\mathit{bool}, \mathit{horn}, \mathit{krom}, \mathit{core}\}$, we denote by $\mathit{LTL}^{\boldsymbol{o}}_{\boldsymbol{c}}$ the fragment of *LTL* with clauses of the form (4), where the C_i can only use the (future and past) operators indicated in $\boldsymbol{o}$, and $m \leq 1$ if $\boldsymbol{c} = \mathit{horn}$; $k+m \leq 2$ if $\boldsymbol{c} = \mathit{krom}$; $k+m \leq 2$ and $m \leq 1$ if $\boldsymbol{c} = \mathit{core}$; and arbitrary k, m if $\boldsymbol{c} = \mathit{bool}$. If $\boldsymbol{o}$ is omitted, the C_i are atomic. An $\mathit{LTL}^{\boldsymbol{o}}_{\mathit{horn}}$-ontology $\mathcal{O}$ is linear if, in each of its axioms (4), at most one C_i, for $1 \leq i \leq k$, can occur on the right-hand side of an axiom in $\mathcal{O}$ (is an IDB predicate, in datalog parlance). We distinguish between arbitrary $\mathit{LTL}^{\boldsymbol{o}}_{\boldsymbol{c}}$ OMQs $\boldsymbol{q} = (\mathcal{O}, \varkappa)$, where $\mathcal{O}$ is any $\mathit{LTL}^{\boldsymbol{o}}_{\boldsymbol{c}}$ ontology and $\varkappa$ any *LTL*-formula with $\bigcirc$-, $\Box$- and $\Diamond$-operators; positive OMQs (OMPQs), where $\varkappa$ is $\to, \neg$-free; existential OMPQs (OMPEQs) with $\Box$-free $\varkappa$; and atomic OMQs (OMAQs) with atomic $\varkappa$.

The main result of this paper is the tight complexity bounds on deciding $\mathcal{L}$-rewritability (and so data complexity) of *LTL* OMQs in various classes defined above, which are summarised in Table 1. The ExpSpace upper bound in the first stripe is shown using the $\mathcal{L}$-definability criteria recently obtained in [33] and exponential-size NFAs for *LTL* akin to those in [47]; in the proof of the matching lower bound, an exponential-size automaton is encoded in a polynomial-size ontology. If the ontology in an $\mathit{LTL}^{\bigcirc}_{\mathit{horn}}$ OMAQ is linear, we show that its language (yes-data instances) can be captured by a 2NFA with polynomially many states, which allows us to reduce the complexity of deciding $\mathcal{L}$-rewritability to PSpace. However, for linear $\mathit{LTL}^{\bigcirc}_{\mathit{horn}}$ OMPQs (with more expressive queries $\varkappa$), the existence of polynomial-state

Table 1 Complexity of deciding FO-rewritability of *LTL* OMQs.

class of OMQs	FO(<)	FO(<, ≡), AC^0	FO(<, MOD), ACC^0
$LTL^{\bigcirc}_{horn}$ OMAQs LTL_{krom} OMPEQs $LTL^{\Box\bigcirc}_{bool}$ OMQs	ExpSpace	ExpSpace	ExpSpace
linear $LTL^{\bigcirc}_{horn}$ OMAQ linear $LTL^{\bigcirc}_{horn}$ OMPQs	PSpace	PSpace	PSpace ?
$LTL^{\bigcirc}_{krom}$ OMAQs $LTL^{\bigcirc}_{core}$ OMPEQs $LTL^{\bigcirc}_{core}$ OMPQs	coNP Π^p_2 PSpace	all in AC^0 [6]	–

2NFAs remains open; instead, we show how the structure of the canonical (minimal) models for $LTL^{\bigcirc}_{horn}$-ontologies can be utilised to yield a PSpace algorithm. In the third stripe of the table, we deal with binary-clause ontologies. The coNP-completeness of deciding FO-rewritability of $LTL^{\bigcirc}_{krom}$ OMAQs is established using unary NFAs and results from [42]. The Π^p_2-completeness for $LTL^{\bigcirc}_{core}$ OMPEQs (without $\vee$ in ontologies but with $\wedge$, $\vee$, $\Diamond$ in queries) and the PSpace-completeness for $LTL^{\bigcirc}_{core}$ OMPQs (admitting $\Box$ in queries, too) can be explained by the fact that the combined complexity of answering such OMPEQs and OMPQs is NP-hard rather than tractable as in the previous case.

All omitted details and proofs are provided in the full draft of the paper [40].

2 Preliminaries

Temporal ontology-mediated queries. In our setting, the alphabet of *LTL* comprises a set of *atomic concepts* A_i, $i < \omega$. *Basic temporal concepts*, C, are defined by the grammar $C ::= A_i \mid \Box_F C \mid \Box_P C \mid \bigcirc_F C \mid \bigcirc_P C$. A *temporal ontology*, $\mathcal{O}$, is a finite set of *axioms* in normal form (4) with $\Box_P\Box_F$ omitted. An LTL^{o}_{c} *ontology-mediated query* (OMQ) is a pair $\boldsymbol{q} = (\mathcal{O}, \varkappa)$, where $\mathcal{O}$ is an LTL^{o}_{c} ontology (defined above) and $\varkappa$ a *temporal concept* built from atoms A_i using the Booleans and temporal operators $\bigcirc_F$, $\Box_F$, $\Diamond_F$ and their past-time counterparts $\bigcirc_P$, $\Box_P$, $\Diamond_P$. The set of atomic concepts occurring in $\boldsymbol{q}$ is denoted by $\mathsf{sig}(\boldsymbol{q})$.

A *data instance* – *ABox* in description logic parlance – is a finite set $\mathcal{A}$ of atoms $A_i(\ell)$, for $\ell \in \mathbb{Z}$, together with a finite interval $\mathsf{tem}(\mathcal{A}) = [m, n] \subseteq \mathbb{Z}$, the *active domain* of $\mathcal{A}$, such that $m \leq \ell \leq n$, for all $A_i(\ell) \in \mathcal{A}$. If $\mathcal{A} = \emptyset$, then $\mathsf{tem}(\mathcal{A})$ may also be $\emptyset$. Otherwise, we assume (without loss of generality) that $m = 0$. If $\mathsf{tem}(\mathcal{A})$ is not specified explicitly, it is assumed to be either empty or $[0, n]$, where n is the maximal timestamp in $\mathcal{A}$. By a *signature*, Ξ, we mean any finite set of atomic concepts. An ABox $\mathcal{A}$ is a Ξ-*ABox* if $A_i(\ell) \in \mathcal{A}$ implies $A_i \in \Xi$.

A *temporal interpretation* is a structure of the form $\mathcal{I} = (\mathbb{Z}, A^{\mathcal{I}}_0, A^{\mathcal{I}}_1, \dots)$ with $A^{\mathcal{I}}_i \subseteq \mathbb{Z}$, for every $i < \omega$. The *extension* $\varkappa^{\mathcal{I}}$ of a temporal concept $\varkappa$ in $\mathcal{I}$ is defined inductively as usual in *LTL* under the "strict semantics" [22, 27]: $(\bigcirc_F\varkappa)^{\mathcal{I}} = \{\, n \in \mathbb{Z} \mid n+1 \in \varkappa^{\mathcal{I}} \,\}$, $(\Box_F\varkappa)^{\mathcal{I}} = \{\, n \in \mathbb{Z} \mid k \in \varkappa^{\mathcal{I}} \text{ for all } k > n \,\}$, $(\Diamond_F\varkappa)^{\mathcal{I}} = \{\, n \in \mathbb{Z} \mid \text{there is } k > n \text{ with } k \in \varkappa^{\mathcal{I}} \,\}$, and symmetrically for the past-time operators. We say that an axiom (4) is *true* in $\mathcal{I}$ if $C^{\mathcal{I}}_1 \cap \dots \cap C^{\mathcal{I}}_k \subseteq C^{\mathcal{I}}_{k+1} \cup \dots \cup C^{\mathcal{I}}_{k+m}$. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{O}$ if all axioms of $\mathcal{O}$ are true in $\mathcal{I}$; it is a *model* of $\mathcal{A}$ if $A_i(\ell) \in \mathcal{A}$ implies $\ell \in A^{\mathcal{I}}_i$.

We can treat $\boldsymbol{q}$ as a *Boolean* OMQ, which returns yes/no, or as a *specific* OMQ, which returns timestamps from the ABox in question assigned to the free variable, say x, in the standard FO-translation of $\varkappa$. In the latter case, we write $\boldsymbol{q}(x) = (\mathcal{O}, \varkappa(x))$. More precisely, a *certain answer* to a Boolean OMQ $\boldsymbol{q} = (\mathcal{O}, \varkappa)$ over an ABox $\mathcal{A}$ is yes if, for every model

Table 2 Rewritability of *LTL* OMQs [6].

	OMAQs		OMPQs	
$\boldsymbol{c}$	$LTL^{\Box}_{\boldsymbol{c}}$	$LTL^{\bigcirc}_{\boldsymbol{c}}$ and $LTL^{\Box\bigcirc}_{\boldsymbol{c}}$	$LTL^{\Box}_{\boldsymbol{c}}$	$LTL^{\bigcirc}_{\boldsymbol{c}}$ and $LTL^{\Box\bigcirc}_{\boldsymbol{c}}$
bool	FO(<)	FO(RPR)	FO(RPR)	FO(RPR)
krom		FO(<, ≡)		
horn		FO(RPR)	FO(<)	FO(<, ≡)
core		FO(<, ≡)		

$\mathcal{I}$ of $\mathcal{O}$ and $\mathcal{A}$, there is $k \in \mathbb{Z}$ such that $k \in \varkappa^{\mathcal{I}}$, in which case we write $(\mathcal{O}, \mathcal{A}) \models \exists x \varkappa(x)$. We write $(\mathcal{O}, \mathcal{A}) \models \varkappa(k)$, for $k \in \mathbb{Z}$, if $k \in \varkappa^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{O}$ and $\mathcal{A}$. A *certain answer* to a specific OMQ $\boldsymbol{q}(x) = (\mathcal{O}, \varkappa(x))$ over $\mathcal{A}$ is any $k \in \mathsf{tem}(\mathcal{A})$ with $(\mathcal{O}, \mathcal{A}) \models \varkappa(k)$. By the *evaluation* (or *answering*) *problem* for $\boldsymbol{q}$ or $\boldsymbol{q}(x)$ we understand the decision problem "$(\mathcal{O}, \mathcal{A}) \models^{?} \exists x \varkappa(x)$" or "$(\mathcal{O}, \mathcal{A}) \models^{?} \varkappa(k)$" with input $\mathcal{A}$ or, respectively, $\mathcal{A}$ and $k \in \mathsf{tem}(\mathcal{A})$.

▶ **Example 2.**

(i) Suppose $\mathcal{O}_1 = \{A \to \Box_F B,\ \Box_F B \to C\}$ and $\boldsymbol{q}_1 = (\mathcal{O}_1, C \wedge D)$. The certain answer to $\boldsymbol{q}_1$ over $\mathcal{A}_1 = \{D(0), B(1), A(1)\}$ is yes, and no over $\mathcal{A}_2 = \{D(0), A(1)\}$. The only answer to $\boldsymbol{q}_1(x) = \big(\mathcal{O}_1, (C \wedge D)(x)\big)$ over $\mathcal{A}_1$ is 0.

(ii) Let $\mathcal{O}_2 = \{\, \bigcirc_P A \to B,\ \bigcirc_P B \to A,\ A \wedge B \to \bot \,\}$. The certain answer to $\boldsymbol{q}_2 = (\mathcal{O}_2, C)$ over $\mathcal{A}_1 = \{A(0)\}$ is no, and yes over $\mathcal{A}_2 = \{A(0), A(1)\}$. There are no certain answers to $\boldsymbol{q}_2(x) = (\mathcal{O}_1, C(x))$ over $\mathcal{A}_1$, while over $\mathcal{A}_2$ the answers are 0 and 1.

(iii) Consider now the ontology $\mathcal{O}_3 = \{\bigcirc_P B_k \wedge A_0 \to B_k,\ \bigcirc_P B_{1-k} \wedge A_1 \to B_k \mid k = 0, 1\}$. For any word $\boldsymbol{e} = e_1 \dots e_n \in \{0,1\}^n$, let $\mathcal{A}_{\boldsymbol{e}} = \{B_0(0)\} \cup \{A_{e_i}(i) \mid 0 < i \leq n\} \cup \{E(n)\}$. The answer to $\boldsymbol{q}_3 = (\mathcal{O}_3, B_0 \wedge E)$ over the ABox $\mathcal{A}_{\boldsymbol{e}}$ is yes iff the number of 1s in $\boldsymbol{e}$ is even.

▶ Remark 3. As follows from [4, 25], if arbitrary (boxed) *LTL*-formulas are used as axioms of an ontology $\mathcal{O}$, then one can construct an $LTL^{\Box\bigcirc}_{bool}$ ontology $\mathcal{O}'$ that is a model conservative extension of $\mathcal{O}$. For example, let $\mathcal{O}'$ be the result of replacing (1) in $\mathcal{O}$ from Example 1 by $\mathit{Malfunction} \wedge \Box_F X \to \bot$ and $\top \to X \vee \mathit{Fixed}$, for a fresh X. Then $\boldsymbol{q} = (\mathcal{O}, \varkappa)$ is equivalent to $\boldsymbol{q}' = (\mathcal{O}', \varkappa)$ in the sense that $\boldsymbol{q}$ and $\boldsymbol{q}'$ have the same certain answers over any $\mathsf{sig}(\boldsymbol{q})$-ABox.

Let $\mathcal{L} \in \{\mathsf{FO}(<), \mathsf{FO}(<, \equiv), \mathsf{FO}(<, \mathsf{MOD}), \mathsf{FO}(\mathsf{RPR})\}$. A Boolean OMQ $\boldsymbol{q}$ is *$\mathcal{L}$-rewritable over Ξ-ABoxes* if there is an $\mathcal{L}$-sentence $\boldsymbol{Q}$ such that, for any Ξ-ABox $\mathcal{A}$, the certain answer to $\boldsymbol{q}$ over $\mathcal{A}$ is yes iff $\mathfrak{S}_{\mathcal{A}} \models \boldsymbol{Q}$. Here, $\mathfrak{S}_{\mathcal{A}}$ is a structure with domain $\mathsf{tem}(\mathcal{A})$ ordered by $<$, in which $\mathfrak{S}_{\mathcal{A}} \models A_i(\ell)$ iff $A_i(\ell) \in \mathcal{A}$. A specific OMQ $\boldsymbol{q}(x)$ is *$\mathcal{L}$-rewritable over Ξ-ABoxes* if there is an $\mathcal{L}$-formula $\boldsymbol{Q}(x)$ with one free variable x such that, for any Ξ-ABox $\mathcal{A}$, k is a certain answer to $\boldsymbol{q}(x)$ over $\mathcal{A}$ iff $\mathfrak{S}_{\mathcal{A}} \models \boldsymbol{Q}(k)$. The sentence $\boldsymbol{Q}$ and the formula $\boldsymbol{Q}(x)$ are called *$\mathcal{L}$-rewritings* of the OMQs $\boldsymbol{q}$ and $\boldsymbol{q}(x)$, respectively. All $LTL^{\Box\bigcirc}_{bool}$ (Boolean and specific) OMQs are $\mathsf{FO}(\mathsf{RPR})$-rewritable. The *syntactic* classification of *LTL* OMQs by their rewritability type, obtained in [6], is shown in Table 2. It follows, e.g., that all $LTL^{\Box\bigcirc}_{core}$ OMPQs are $\mathsf{FO}(<, \equiv_{\mathbb{N}})$-rewritable, with some of them being not $\mathsf{FO}(<)$-rewritable. It is to be noted that $\mathsf{FO}(<, \mathsf{MOD})$-rewritable OMQs such as $\boldsymbol{q}_3$ in Example 2 and 4 are not captured by these syntactic classes.

▶ **Example 4.**

(i) An $\mathsf{FO}(<)$-rewriting of $\boldsymbol{q}_1(x)$ over arbitrary ABoxes is

$$\boldsymbol{Q}_1(x) = D(x) \land [C(x) \lor \exists y\,(A(y) \land \forall z\,((x < z \leq y) \to B(z)))],$$

$\exists x\,\boldsymbol{Q}_1(x)$ is an $\mathsf{FO}(<)$-rewriting of $\boldsymbol{q}_1$.

(ii) An $\mathsf{FO}(<,\equiv)$-rewriting of $\boldsymbol{q}_2(x)$ is

$$\begin{aligned}\boldsymbol{Q}_2(x) = \;& C(x) \lor \exists x, y\,[(A(x) \land A(y) \land \mathsf{odd}(x,y)) \lor \\ & (B(x) \land B(y) \land \mathsf{odd}(x,y)) \lor (A(x) \land B(y) \land \neg\mathsf{odd}(x,y))],\end{aligned}$$

where $\mathsf{odd}(x,y) = \big(x \equiv 0 \,(\mathrm{mod}\ 2) \leftrightarrow y \not\equiv 0 \,(\mathrm{mod}\ 2)\big)$ implies that $|x-y|$ is odd, and an $\mathsf{FO}(<,\equiv)$-rewriting of $\boldsymbol{q}_2$ is $\exists x\,\boldsymbol{Q}_2(x)$. Recall that odd is not expressible in $\mathsf{FO}(<)$ [34].

(iii) The OMQ $\boldsymbol{q}_3$ is not rewritable to an FO-formula with any numeric predicates as PARITY is not in AC^0 [26]; the following sentence is an $\mathsf{FO}(<,\mathsf{MOD})$-rewriting of $\boldsymbol{q}_3$:

$$\begin{aligned}\boldsymbol{Q}_3 = \exists x, y\,\big[&E(x) \land (y \leq x) \land \forall z\,\big((y < z \leq x) \to A_0(z) \lor A_1(z)\big) \land \\ &\big((B_0(y) \land \exists^2 z\,((y < z \leq x) \land A_1(z))) \lor (B_1(y) \land \neg\exists^2 z\,((y < z \leq x) \land A_1(z)))\big)\big].\end{aligned}$$

In this paper, our aim is to understand how (complex it is) to decide the optimal type of FO-rewritability for a given *LTL* OMQ $\boldsymbol{q}$ over Ξ-ABoxes. Although all of our results hold for both Boolean and specific OMQs, here we only focus on the former; detailed proofs for the latter can be found in the full draft. We begin by observing an intimate connection between $\mathcal{L}$-rewritability of OMQs and $\mathcal{L}$-definability of certain regular languages.

Automata, languages, and OMQs. A *two-way nondeterministic finite automaton* is a quintuple $\mathfrak{A} = (Q, \Sigma, \delta, Q_0, F)$ that consists of an alphabet Σ, a finite set of states Q with a subset $Q_0 \neq \emptyset$ of initial states and a subset F of accepting states, and a transition function $\delta\colon Q \times \Sigma \to 2^{Q\times\{-1,0,1\}}$ indicating the next state and whether the head should move left (-1), right (1), or stay put (0). If $Q_0 = \{q_0\}$ and $|\delta(q,a)| = 1$, for all $q \in Q$ and $a \in \Sigma$, then $\mathfrak{A}$ is *deterministic*, in which case we write $\mathfrak{A} = (Q, \Sigma, \delta, q_0, F)$. If $\delta(q,a) \subseteq Q \times \{1\}$, for all $q \in Q$ and $a \in \Sigma$, then $\mathfrak{A}$ is a *one-way* automaton, and we write $\delta\colon Q \times \Sigma \to 2^Q$. As usual, DFA and NFA refer to one-way deterministic and non-deterministic finite automata, respectively, while 2DFA and 2NFA to the corresponding two-way automata. Given a 2NFA $\mathfrak{A}$, we write $q \to_{a,d} q'$ if $(q',d) \in \delta(q,a)$; given an NFA $\mathfrak{A}$, we write $q \to_a q'$ if $q' \in \delta(q,a)$. A *run* of a 2NFA $\mathfrak{A}$ is a word in $(Q \times \mathbb{N})^*$. A run $(q_0,i_0),\dots,(q_m,i_m)$ is a *run of* $\mathfrak{A}$ *on a word* $w = a_0 \dots a_n \in \Sigma^*$ if $q_0 \in Q_0$, $i_0 = 0$ and there exist $d_0,\dots,d_{m-1} \in \{-1,0,1\}$ such that $q_j \to_{a_j,d_j} q_{j+1}$ and $i_{j+1} = i_j + d_j$ for all j, $0 \leq j < m$. The run is *accepting* if $q_m \in F$, $i_m = n+1$. $\mathfrak{A}$ *accepts* $w \in \Sigma^*$ if there is an accepting run of $\mathfrak{A}$ on w; the language $\boldsymbol{L}(\mathfrak{A})$ of $\mathfrak{A}$ is the set of all words accepted by $\mathfrak{A}$.

Given an NFA $\mathfrak{A}$, states $q, q' \in Q$, and $w = a_0 \dots a_n \in \Sigma^*$, we write $q \to_w q'$ if either $w = \varepsilon$ and $q' = q$ or there is a run of $\mathfrak{A}$ on w that starts with $(q_0, 0)$ and ends with $(q', n+1)$. We say that a state $q \in Q$ is *reachable* if $q' \to_w q$, for some $q' \in Q_0$ and $w \in \Sigma^*$. Given a DFA $\mathfrak{A} = (Q, \Sigma, \delta, q_0, F)$, for any word $w \in \Sigma^*$, we define a function $\delta_w\colon Q \to Q$ by taking $\delta_w(q) = q'$ iff $q \to_w q'$.

A language $\boldsymbol{L}$ over an alphabet Σ is *$\mathcal{L}$-definable* if there is an $\mathcal{L}$-sentence φ in the signature Σ, whose symbols are treated as unary predicates, such that, for any $w \in \Sigma^*$, we have $w = a_0 \dots a_n \in \boldsymbol{L}$ iff $\mathfrak{S}_w \models \varphi$, where $\mathfrak{S}_w$ is a structure with domain $\{0,\dots,n\}$, in which $\mathfrak{S}_w \models a(i)$ iff $a = a_i$, for $i \leq n$.

For any OMQ $\boldsymbol{q}$ and $\Xi \subseteq \mathsf{sig}(\boldsymbol{q})$, we regard $\Sigma_\Xi = 2^\Xi$ as an *alphabet*. Any Ξ-ABox $\mathcal{A}$ can be given as a Σ_Ξ-word $w_\mathcal{A} = a_0 \dots a_n$ with $a_i = \{A \mid A(i) \in \mathcal{A}\}$. Conversely, any Σ_Ξ-word $w = a_0 \dots a_n$ gives the ABox $\mathcal{A}_w$ with $\mathsf{tem}(\mathcal{A}_w) = [0, n]$ and $A(i) \in \mathcal{A}_w$ iff $A \in a_i$. The word $\emptyset$ corresponds to $\mathcal{A}_\emptyset = \emptyset$ with $\mathsf{tem}(\mathcal{A}_\emptyset) = [0, 0]$. The *language* $\boldsymbol{L}_\Xi(\boldsymbol{q})$ is defined to be the set of Σ_Ξ-words $w_\mathcal{A}$ with a yes-answer to $\boldsymbol{q}$ over $\mathcal{A}$.

▶ **Proposition 5.** *The language $\boldsymbol{L}_\Xi(\boldsymbol{q})$ is regular. For $\mathcal{L} \in \{\mathsf{FO}(<), \mathsf{FO}(<, \equiv), \mathsf{FO}(<, \mathsf{MOD})\}$, the OMQ $\boldsymbol{q}$ is $\mathcal{L}$-rewritable over Ξ-ABoxes iff $\boldsymbol{L}_\Xi(\boldsymbol{q})$ is $\mathcal{L}$-definable.*

Proof. Let $\mathsf{sub}_\boldsymbol{q}$ be the set of temporal concepts in $\boldsymbol{q}$ and their negations. A *type* is any maximal subset $\tau \subseteq \mathsf{sub}_\boldsymbol{q}$ consistent with $\mathcal{O}$. Let $\boldsymbol{T}$ be the set of all types. Define an NFA $\mathfrak{A}$ over Σ_Ξ with $\boldsymbol{L}(\mathfrak{A}) = \Sigma_\Xi^* \setminus \boldsymbol{L}_\Xi(\boldsymbol{q})$. Its states are $Q_{\neg\varkappa} = \{\tau \in \boldsymbol{T} \mid \neg\varkappa \in \tau\}$. The transition relation $\to_a$, for $a \in \Sigma_\Xi$, is defined by taking $\tau_1 \to_a \tau_2$ if the following conditions hold: (a) $a \subseteq \tau_2$, (b) $\bigcirc_F C \in \tau_1$ iff $C \in \tau_2$, (c) $\Box_F C \in \tau_1$ iff $C \in \tau_2$ and $\Box_F C \in \tau_2$, (d) $\Diamond_F C \in \tau_1$ iff $C \in \tau_2$ or $\Diamond_F C \in \tau_2$, and symmetrically for $\bigcirc_P, \Box_P, \Diamond_P$. The initial (accepting) states are those $\tau \in Q_{\neg\varkappa}$, for which $\tau \cup \{\Box_P \neg\varkappa\}$ (respectively, $\tau \cup \{\Box_F \neg\varkappa\}$) is consistent with $\mathcal{O}$. Then $w \in \boldsymbol{L}(\mathfrak{A})$ iff $(\mathcal{O}, \mathcal{A}_w) \not\models \exists x\, \varkappa(x)$, for any $w \in \Sigma_\Xi^*$. The number of states in $\mathfrak{A}$ is $2^{O(|\boldsymbol{q}|)}$ and $\mathfrak{A}$ can be constructed using space polynomial in $|\boldsymbol{q}|$ as *LTL*-satisfiability is in PSPACE. ◀

Thus, we can reformulate the evaluation problem for an *LTL* OMQ $\boldsymbol{q}$ over Ξ-ABoxes as the *word problem* for the regular language $\boldsymbol{L}_\Xi(\boldsymbol{q})$.

3 Deciding FO-rewritability of *LTL* OMQs

In this section, we establish the complexity of recognising the rewritability type of an arbitrary $\mathit{LTL}_{\mathit{bool}}^{\Box\bigcirc}$ OMQ.

▶ **Theorem 6.** *For any $\mathcal{L} \in \{\mathsf{FO}(<), \mathsf{FO}(<, \equiv), \mathsf{FO}(<, \mathsf{MOD})\}$, deciding $\mathcal{L}$-rewritability of $\mathit{LTL}_{\mathit{bool}}^{\Box\bigcirc}$ OMQs over Ξ-ABoxes is EXPSPACE-complete; the lower bound holds already for $\mathit{LTL}_{\mathit{horn}}^{\bigcirc}$ OMAQs.*

Proof. The upper bound follows from Proposition 5 and the fact that $\mathcal{L}$-definability of the language of an NFA can be checked in polynomial space [33]. Here, we sketch the proof of the matching lower bound for $\mathit{LTL}_{\mathit{horn}}^{\bigcirc}$ OMAQs, which is inspired by the reductions used for the PSPACE-hardness proofs of $\mathcal{L}$-definability of DFA languages in [18] and [33, Theorem 2].

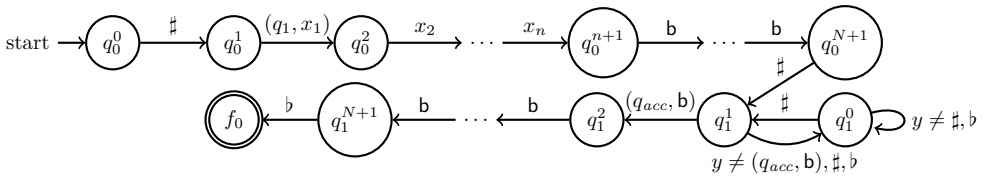
The structure of the proof is as follows: given a Turing machine $\boldsymbol{M}$ that decides a language using at most $N = \exp(n)$ tape cells on any input of size n, for some exponential function exp, we construct (following [33]) automata $\mathfrak{A}_<$, $\mathfrak{A}_\equiv$, and $\mathfrak{A}_\mathsf{MOD}$ of size polynomial in N whose languages $\boldsymbol{L}(\mathfrak{A}_<)$, $\boldsymbol{L}(\mathfrak{A}_\equiv)$, and $\boldsymbol{L}(\mathfrak{A}_\mathsf{MOD})$ are, respectively, $\mathsf{FO}(<)$-, $\mathsf{FO}(<, \equiv)$-, and $\mathsf{FO}(<, \mathsf{MOD})$-definable iff $\boldsymbol{M}$ rejects $\boldsymbol{x}$. Then we construct $\mathit{LTL}_{\mathit{horn}}^{\bigcirc}$ OMAQs $(\mathcal{O}_<, F)$, $(\mathcal{O}_\equiv, F)$, and $(\mathcal{O}_\mathsf{MOD}, F)$ of polynomial size in $|\boldsymbol{x}|$ and $|\boldsymbol{M}|$ that are rewritable into $\mathsf{FO}(<)$, $\mathsf{FO}(<, \equiv)$, and $\mathsf{FO}(<, \mathsf{MOD})$, respectively, iff the corresponding language $\boldsymbol{L}(\mathfrak{A}_\mathcal{L})$ is $\mathcal{L}$-definable.

Suppose $\boldsymbol{M} = (Q, \Sigma, \gamma, \mathsf{b}, q_0, q_{acc})$ with a set Q of states, tape alphabet Σ with b for blank, transition function γ, initial state q_0 and accepting state q_{acc}. Without loss of generality we assume that $\boldsymbol{M}$ erases the tape before accepting and has its head at the left-most cell in an accepting configuration, and if $\boldsymbol{M}$ does not accept the input, it runs forever. Given an input word $\boldsymbol{x} = x_1 \dots x_n$ over Σ, we represent configurations $\mathfrak{c}$ of the computation of $\boldsymbol{M}$ on $\boldsymbol{x}$ by an N-long word written on the tape (with sufficiently many blanks at the end), in which the symbol, y, in the active cell is replaced by the pair (q, y) for the current state q. The accepting computation of $\boldsymbol{M}$ on $\boldsymbol{x}$ is encoded by the word $\sharp\, \mathfrak{c}_1\, \sharp\, \mathfrak{c}_2\, \sharp \dots \sharp\, \mathfrak{c}_{k-1}\, \sharp\, \mathfrak{c}_k \flat$ over the alphabet $\Sigma' = \Sigma \cup (Q \times \Sigma) \cup \{\sharp, \flat\}$, with $\mathfrak{c}_1, \mathfrak{c}_2, \dots, \mathfrak{c}_k$ being the subsequent configurations.

In particular, $\mathfrak{c}_1$ is the initial configuration on $\boldsymbol{x}$ of the form $(q_0, x_1)x_2 \dots x_n \mathsf{b} \dots \mathsf{b}$, and $\mathfrak{c}_k$ is the accepting configuration the form $(q_{acc}, \mathsf{b})\mathsf{b} \dots \mathsf{b}$. As usual for this representation of computations, we may regard γ as a partial function from $\big(\Sigma \cup (Q \times \Sigma)\big)^3$ to $\Sigma \cup (Q \times \Sigma)$.

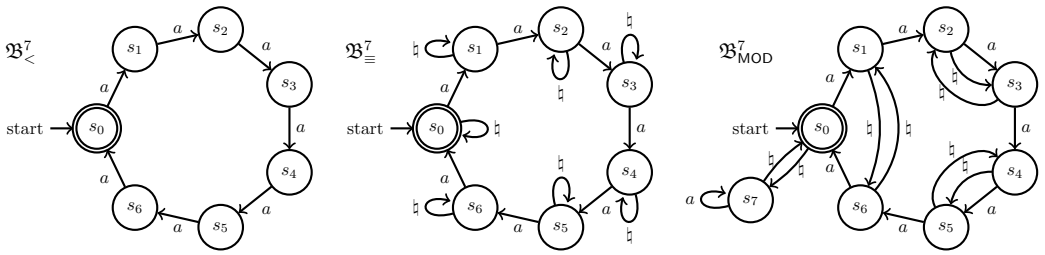
Let p be the first prime such that $p > N+1$ and $p \not\equiv \pm 1 \pmod{10}$. By [12, Corollary 1.6], p is polynomial in N. Our first aim is to construct a $p+1$-long sequence $\mathfrak{A}_i$ of disjoint DFAs over Σ' such that each $\mathfrak{A}_i$ is of size polynomial in N and $|\boldsymbol{M}|$, it checks certain properties of an accepting computation on $\boldsymbol{x}$, and $\boldsymbol{M}$ accepts $\boldsymbol{x}$ iff the intersection of the $\boldsymbol{L}(\mathfrak{A}_i)$ is not empty and consists of the single word encoding the accepting computation on $\boldsymbol{x}$.

The DFA $\mathfrak{A}_0$ checks whether an input word starts with $\sharp\mathfrak{c}_1$ and ends with $\sharp\mathfrak{c}_k\flat$:



If $1 \leq i \leq N$, the DFA $\mathfrak{A}_i$ checks, for all j, whether $\gamma(\sigma_{i-1}^j, \sigma_i^j, \sigma_{i+1}^j) = \sigma_i^{j+1}$, where σ_l^k denotes the lth symbol of $\mathfrak{c}_k$. Finally, if $N+1 \leq i \leq p$, then $\mathfrak{A}_i$ accepts all words with a single occurrence of $\flat$, which is the input's last character. It is not hard to check that the $\mathfrak{A}_i$ are such that $\boldsymbol{M}$ accepts $\boldsymbol{x}$ iff $\bigcap_{i=0}^p \boldsymbol{L}(\mathfrak{A}_i) \neq \emptyset$, in which case this intersection consists of a single word that encodes the accepting computation of $\boldsymbol{M}$ on $\boldsymbol{x}$.

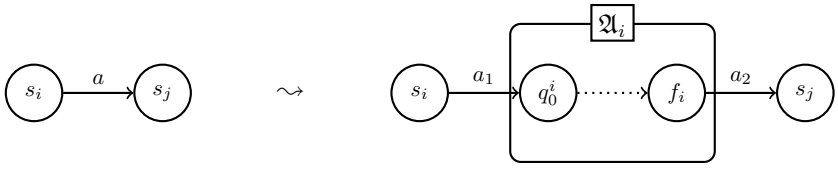
Now we use the $\mathfrak{A}_i$ to define the automata $\mathfrak{A}_<$, $\mathfrak{A}_\equiv$, and $\mathfrak{A}_{\mathsf{MOD}}$. To begin with, we construct DFAs $\mathfrak{B}^p_<$, $\mathfrak{B}^p_\equiv$ and $\mathfrak{B}^p_{\mathsf{MOD}}$, where $p > 5$ is a prime number, following the patterns shown in the picture below for $p = 7$:



In general, $\mathfrak{B}^p_{\mathsf{MOD}} = \big(\{s_i \mid i \leq p\}, \{a, \natural\}, \delta^{\mathfrak{B}^p_{\mathsf{MOD}}}, s_0, \{s_0\}\big)$, where

- $\delta_a^{\mathfrak{B}^p_{\mathsf{MOD}}}(s_p) = s_p$, and $\delta_a^{\mathfrak{B}^p_{\mathsf{MOD}}}(s_i) = s_j$ if $i, j < p$ and $j \equiv i+1 \pmod p$;
- $\delta_\natural^{\mathfrak{B}^p_{\mathsf{MOD}}}(s_0) = s_p$, $\delta_\natural^{\mathfrak{B}^p_{\mathsf{MOD}}}(s_p) = s_0$, and $\delta_\natural^{\mathfrak{B}^p_{\mathsf{MOD}}}(s_i) = s_j$ if $1 \leq i, j < p$ and $i \cdot j \equiv p-1 \pmod p$, that is, $j = -1/i$ in the finite field $\mathbb{F}_p$.

Now take some fresh symbols a_1, a_2. We define the automata $\mathfrak{A}_<$, $\mathfrak{A}_\equiv$, $\mathfrak{A}_{\mathsf{MOD}}$ over the same alphabet $\Sigma_+ = \Sigma' \cup \{a_1, a_2, \natural\}$ by taking, respectively, $\mathfrak{B}^p_<$, $\mathfrak{B}^p_\equiv$, $\mathfrak{B}^p_{\mathsf{MOD}}$ and replacing each transition $s_i \to_a s_j$ in them by a fresh copy of $\mathfrak{A}_i$, for $i \leq p$, as shown in the picture below, where q_0^i is the initial state of $\mathfrak{A}_i$:



We make $\mathfrak{A}_<$, $\mathfrak{A}_\equiv$, $\mathfrak{A}_{\mathsf{MOD}}$ deterministic by adding a trash state tr looping on itself with every $y \in \Sigma_+$, and adding the missing transitions leading to tr. It follows that $\mathfrak{A}_<$, $\mathfrak{A}_\equiv$, and $\mathfrak{A}_{\mathsf{MOD}}$ are minimal DFAs of size polynomial in N and $|\boldsymbol{M}|$. Using the algebraic properties of respective syntactic monoids of these languages (see [33, Theorem 1]), one can prove that the languages $\boldsymbol{L}(\mathfrak{A}_<)$, $\boldsymbol{L}(\mathfrak{A}_\equiv)$, and $\boldsymbol{L}(\mathfrak{A}_{\mathsf{MOD}})$ are $\mathcal{L}$-definable for the respective $\mathcal{L}$ iff $\boldsymbol{M}$ rejects $\boldsymbol{x}$.

Now we define $\mathit{LTL}^{\bigcirc}_{horn}$ ontologies $\mathcal{O}_<$, $\mathcal{O}_\equiv$ and $\mathcal{O}_{\mathsf{MOD}}$ simulating $\mathfrak{A}_<$, $\mathfrak{A}_\equiv$ and $\mathfrak{A}_{\mathsf{MOD}}$ such that the size of each ontology is polynomial in $|\boldsymbol{x}|$ and $|\boldsymbol{M}|$.

While $\mathfrak{A}_0$ is of size exponential in n, it has a rather repetitive structure with many transitions of the "same type": $q_0^l \to_{\mathsf{b}} q_0^{l+1}$, for $n < l < N$. We deal with them with the help of counters, where a *counter* is a set $\mathbb{A} = \{A^i_j \mid i = 0, 1,\ j = 1, \dots, k\}$ of atoms for some k logarithmic in N that is used to store values between 0 and $2^k - 1$, which can be different at different time points. We can define Boolean formulas such as $[\mathbb{A} = c]$, $[\mathbb{A} > c]$, $[\mathbb{A} = \mathbb{B} + 1]$ with self-explanatory names that are true iff the value stored in the counters satisfies the corresponding condition. For example, the formula $[\mathbb{A} = \mathbb{B}] = \bigwedge_{j=1}^{k} \big((B^0_j \to A^0_j) \land (B^1_j \to A^1_j)\big)$ is true at a time point $m \in \mathbb{Z}$ in an interpretation $\mathcal{I}$ iff the values stored in $\mathbb{A}$ and $\mathbb{B}$ are the same. In particular, we can have counters $\mathbb{A}$ and $\mathbb{L}$ with atomic concepts A^i_j and L^i_j, for $i = 0, 1$, $j = 1, \dots, k$, and then the transitions $q_0^l \to_{\mathsf{b}} q_0^{l+1}$ for $n < l < N$ in $\mathfrak{A}_0$ are captured by the formula

$$[\mathbb{A} = 0] \land Q_0 \land [\mathbb{L} > n] \land [\mathbb{L} < N + 1] \land \mathsf{b} \to [(\bigcirc_F \mathbb{A}) = 0] \land \bigcirc_F Q_0 \land [(\bigcirc_F \mathbb{L}) = \mathbb{L} + 1],$$

which is equivalent to polynomially-many $\mathit{LTL}^{\bigcirc}_{horn}$ axioms. Using the same idea, we can encode all of the transitions of the automata $\mathfrak{A}_<$ and $\mathfrak{A}_\equiv$ by $\mathit{LTL}^{\bigcirc}_{horn}$ ontologies $\mathcal{O}_<$ and $\mathcal{O}_\equiv$ of size polynomial in n and $\boldsymbol{M}$.

Given a word $w = a_1 \dots a_k$, we denote by $\mathcal{A}_w$ the ABox constructed by taking $\bigcup\{a_j(j)\}$ and adding to it $X(0)$ to mark the beginning of the word, and $Y(k+1)$ to mark the end. We also add to the ontology axioms to ensure that an atomic concept F is entailed by the counters and the end word marker Y when the values of the counters correspond to the accepting state of $\mathfrak{A}_\mathcal{L}$. This way we have that $\mathfrak{A}_<$ accepts w iff $(\mathcal{O}_<, \mathcal{A}_w) \models \exists x\, F(x)$. Thus $(\mathcal{O}_<, F)$ is $\mathsf{FO}(<)$-rewritable iff $\boldsymbol{M}$ rejects $\boldsymbol{x}$, as required. $\mathcal{O}_\equiv$ is constructed very similarly (see $\mathfrak{B}^7_<$ vs. $\mathfrak{B}^7_\equiv$) and one can show that $(\mathcal{O}_\equiv, F)$ is $\mathsf{FO}(<, \equiv)$-rewritable iff $\boldsymbol{M}$ rejects $\boldsymbol{x}$.

Defining $\mathcal{O}_{\mathsf{MOD}}$ requires some additional tricks. Most importantly, we need to extend $\mathcal{O}_<$ with axioms for handling $\natural$-transitions between certain states of $\mathfrak{A}_{\mathsf{MOD}}$ as follows:

$$[\mathbb{A} = 0] \land S \land \natural \to [(\bigcirc_F \mathbb{A}) = p] \land \bigcirc_F S, \quad [\mathbb{A} = p] \land S \land \natural \to [(\bigcirc_F \mathbb{A}) = 0] \land S,$$
$$[\mathbb{A} > 0] \land [\mathbb{A} < p] \land S \land \natural \to [(\bigcirc_F \mathbb{A}) = \mathbb{J}] \land \bigcirc_F S.$$

Here, $\mathbb{J}$ is a new counter that stores the value $j = -1/i$ in the field $\mathbb{F}_p$, which is required to make sure that, for $i \neq 0, p$, we have $\mathcal{O}_{\mathsf{MOD}} \models [\mathbb{A} = i] \land S \land \natural \to [(\bigcirc_F \mathbb{A}) = j] \land \bigcirc_F S$. We achieve this as follows. To compute modular inverses using the standard algorithm [31, Exercise 4.5.2.39], we need to halve the number in a counter (easy), compare two counters (using an additional counter), add and subtract (using extra counters for carries). All of this can be done by means of $O(k)$ counters (a fixed number of counters per $O(k)$ steps of the algorithm) with polynomially-many additional axioms. So we compute j when required and store it in the counter $\mathbb{J}$. ◀

We also observe that $\mathit{LTL}^{\bigcirc}_{horn}$ ontologies can be encoded by positive existential queries mediated by covering axioms available in LTL_{krom}:

▶ **Theorem 7.** *$\mathcal{L}$-rewritability of LTL_{krom} OMPEQs over Ξ-ABoxes is ExpSpace-complete.*

Proof. Any $LTL^{\bigcirc}_{horn}$ OMAQ $\boldsymbol{q} = (\mathcal{O}, A)$ can be reduced to an $LTL^{\bigcirc}_{krom}$ OMPEQ $\boldsymbol{q}' = (\mathcal{O}', \varkappa)$. For example, we can encode $\mathcal{O} = \{\bigcirc_P A_1 \land A_2 \to A\}$ by $\varkappa = A \lor \Diamond_P \Diamond_F (\bigcirc_P A_1 \land A_2 \land \bar{A})$, for a fresh atom $\bar{A}$, and $\mathcal{O}' = \{A \land \bar{A} \to \bot, \top \to A \lor \bar{A}\}$. ◀

4 Deciding $\mathcal{L}$-rewritability of linear positive $LTL^{\bigcirc}_{horn}$ OMQs

As well known, deciding FO-rewritability of monadic datalog queries is 2ExpTime-complete [11,21], which goes down to PSpace for the important class of linear monadic queries [21,45]. It is not hard to see that any DFA can be simulated by a linear $LTL^{\bigcirc}_{horn}$ OMAQ, which gives a PSpace lower bound for deciding $\mathcal{L}$-rewritability. Also, recall from [6] that, for any $LTL^{\Box\bigcirc}_{horn}$ ontology $\mathcal{O}$ and ABox $\mathcal{A}$ consistent with $\mathcal{O}$, there is a *canonical model* $\mathcal{C}_{\mathcal{O},\mathcal{A}}$ of $\mathcal{O}$ and $\mathcal{A}$ such that $(\mathcal{O}, \mathcal{A}) \models A(k)$ iff $\mathcal{C}_{\mathcal{O},\mathcal{A}} \models A(k)$, for all $k \in \mathbb{Z}$. Given an interpretation $\mathcal{I}$, an OMQ $\boldsymbol{q}$ and $k \in \mathbb{Z}$, we denote by $\tau_{\mathcal{I}}(k)$ the $\boldsymbol{q}$-type of k in $\mathcal{I}$ (see the proof of Proposition 5).

▶ **Theorem 8.**

(i) *For any $\mathcal{L} \in \{\mathsf{FO}(<), \mathsf{FO}(<, \equiv), \mathsf{FO}(<, \mathsf{MOD})\}$, deciding $\mathcal{L}$-rewritability of linear $LTL^{\bigcirc}_{horn}$ OMAQs over Ξ-ABoxes is PSpace-complete.*

(ii) *For any $\mathcal{L} \in \{\mathsf{FO}(<), \mathsf{FO}(<, \equiv)\}$, deciding $\mathcal{L}$-rewritability of linear $LTL^{\bigcirc}_{horn}$ OMPQs over Ξ-ABoxes is PSpace-complete.*

Proof.

(i) We encode an OMAQ $\boldsymbol{q}$ as a polysize 2NFA $\mathfrak{A}^{\Xi}_{\mathcal{O}}$ over the alphabet 2^{Ξ}, having (among others) states q_L for $L \in \mathit{idb}(\mathcal{O}) \cup \{\bot\}$, with $\boldsymbol{L}_{\Xi}(\boldsymbol{q}) = \{\boldsymbol{a} \in \Sigma^*_{\Xi} \mid \emptyset^N \boldsymbol{a} \emptyset^N \in \boldsymbol{L}(\mathfrak{A}^{\Xi}_{\mathcal{O}})\}$, $\mathit{idb}(\mathcal{O})$ comprising the IDB predicates of $\mathcal{O}$ and $N = \mathit{poly}(|\boldsymbol{q}|)$. To illustrate, the following transitions are in $\mathfrak{A}^{\Xi}_{\mathcal{O}}$ for the axiom $\bigcirc^2_P A' \land \bigcirc_P A \to B$ with IDB A: $q_A \to_{a,-1} q'$ for any $a \in 2^{\Xi}$, $q' \to_{q,1} q_h$ if $A' \in a$ and $q' \to_{q,1} q''$ otherwise, and $q'' \to_{a,1} q_B$ for any $a \in 2^{\Xi}$, where q_h is a fixed trash state. Then we transform, in PSpace, the 2NFA $\mathfrak{A}^{\Xi}_{\mathcal{O}}$ to a DFA $\mathfrak{A}'$ with $\boldsymbol{L}_{\Xi}(\boldsymbol{q}) = \boldsymbol{L}(\mathfrak{A}')$ in the same way as in [33, Section 5], but with different initial and accepting states, to reflect the fact that accepted words have $\emptyset^N$ as a prefix and suffix.

(ii) The canonical model property of $LTL^{\Box\bigcirc}_{horn}$ allows us to formulate the following criteria in terms of types of the canonical model and ABoxes (cf. [33, Theorem 1 (*i*), (*ii*)]):

▶ **Lemma 9.** *An $LTL^{\Box\bigcirc}_{horn}$ OMPQ $\boldsymbol{q} = (\mathcal{O}, \varkappa)$ is not $\mathsf{FO}(<)$-rewritable iff there exist ABoxes $\mathcal{A}$, $\mathcal{B}$, $\mathcal{D}$ and $k \geq 2$ such that the following conditions hold:*

- *$(\mathcal{O}, \mathcal{A}\mathcal{B}^k\mathcal{D})$ is consistent, $\neg\varkappa \in \tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^k\mathcal{D}}}(|\mathcal{A}|-1)$, $\tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^k\mathcal{D}}}(|\mathcal{A}|-1) = \tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^k\mathcal{D}}}(|\mathcal{A}\mathcal{B}^k|-1)$;*
- *either $\varkappa \in \tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^{k+1}\mathcal{D}}}(|\mathcal{A}\mathcal{B}| - 1)$ and $\tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^{k+1}\mathcal{D}}}(|\mathcal{A}\mathcal{B}| - 1) = \tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^{k+1}\mathcal{D}}}(|\mathcal{A}\mathcal{B}^{k+1}| - 1)$ or $(\mathcal{O}, \mathcal{A}\mathcal{B}^{k+1}\mathcal{D})$ is inconsistent.*

Furthermore, $\boldsymbol{q}$ is not $\mathsf{FO}(<, \equiv)$-rewritable iff there also exist ABoxes $\mathcal{U}$ and $\mathcal{W}$ such that $\mathcal{B} = \mathcal{U}\mathcal{W}$, $|\mathcal{W}| = |\mathcal{U}|$ the following conditions hold:

- *$\tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^k\mathcal{D}}}(|\mathcal{A}\mathcal{B}^i| - 1) = \tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^k\mathcal{D}}}(|\mathcal{A}\mathcal{B}^i\mathcal{U}| - 1)$, for all $i < k$, and*
- *either $(\mathcal{O}, \mathcal{A}\mathcal{B}^{k+1}\mathcal{D})$ is inconsistent or $\tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^{k+1}\mathcal{D}}}(|\mathcal{A}\mathcal{B}^i| - 1) = \tau_{\mathcal{C}_{\mathcal{O},\mathcal{A}\mathcal{B}^{k+1}\mathcal{D}}}(|\mathcal{A}\mathcal{B}^i\mathcal{U}| - 1)$, for all i, $1 \leq i \leq k$.*

Moreover, if $\mathcal{O}$ is linear*, then $|\mathcal{A}|, |\mathcal{B}|, |\mathcal{D}|, |\mathcal{W}|, |\mathcal{U}|, k = 2^{O(|\boldsymbol{q}|)}$.*

Similarly to the algorithm of [33, Theorem 3], we guess the ABoxes $\mathcal{X}$ required by Lemma 9 in the form of quadruples of binary relations $\mathsf{b}(\mathcal{X})$ on the states of the 2NFA $\mathfrak{A}^{\Xi}_{\mathcal{O}}$, and then prove that checking the conditions of the lemma can be done in PSpace. ◀

We note that it is harder to transform [33, Theorem 1 (*iii*)] to a PSpace-checkable condition on canonical models and ABoxes. The complexity of $\mathsf{FO}(<, \mathsf{MOD})$-rewritability of linear OMPQs remains open.

5 FO(<)-rewritability of $\mathit{LTL}^{\bigcirc}_{krom}$ OMAQs and $\mathit{LTL}^{\bigcirc}_{core}$ OMPQs

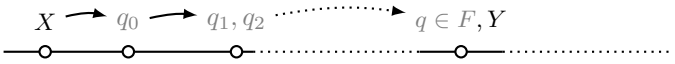
Our next aim is to look for non-trivial OMQ classes deciding FO-rewritability of which could be "easier" than PSPACE. Syntactically, the simplest type of axioms (4) are binary clauses: $C_1 \to C_2$ and $C_1 \wedge C_2 \to \bot$, known as *core* axioms, which together with $C_1 \vee C_2$ form the class Krom. In the atemporal case, the W3C standard language *OWL 2 QL* for ontology-based data access allows core clauses only and uniformly guarantees FO-rewritability [3, 17].

By Theorem 7, OMPEQs with Krom axioms can simulate $\mathit{LTL}^{\bigcirc}_{horn}$ OMAQs, and so are too complex for our aims. On the other hand, $\mathit{LTL}^{\bigcirc}_{krom}$ OMAQs and $\mathit{LTL}^{\bigcirc}_{core}$ OMPQs are all $\mathsf{FO}(<,\equiv)$-rewritable [6], so we can focus on deciding $\mathsf{FO}(<)$-rewritability in these classes.

▶ **Theorem 10.** *$\mathsf{FO}(<)$-rewritability of $\mathit{LTL}^{\bigcirc}_{krom}$ OMAQs over Ξ-ABoxes is coNP-complete.*

Proof. Given $\boldsymbol{q} = (\mathcal{O}, A)$, let $\boldsymbol{q}' = (\mathcal{O}', Y)$ with $\mathcal{O}' = \mathcal{O} \cup \{A \to \bot\}$ and fresh $Y \notin \Xi$. For any Ξ-ABox $\mathcal{A}$, we have $(\mathcal{O}, \mathcal{A}) \models \exists x A(x)$ iff $(\mathcal{O}', \mathcal{A}) \models \exists x\, Y(x)$ iff $(\mathcal{O}', \mathcal{A})$ is inconsistent. One can show, using Kromness, that if $\boldsymbol{L}_{BC} = \{\emptyset^n \mid \mathcal{O} \models B \to \bigcirc_F^{n+1} \neg C\}$ is $\mathsf{FO}(<)$-definable for all $B, C \in \Xi$, then so is $\boldsymbol{L}_\Xi(\boldsymbol{q}')$, and the OMAQ $\boldsymbol{q}$ is therefore $\mathsf{FO}(<)$-rewritable. We can construct a unary NFA accepting $\boldsymbol{L}_{BC}$ in polynomial time [6]. It is also readily seen that a unary language is $\mathsf{FO}(<)$-definable iff it is finite or cofinite. Therefore, deciding $\mathsf{FO}(<)$-definability of a unary NFA is coNP-complete (using [42, Theorem 6.1]) and $\mathsf{FO}(<)$-rewritability of an $\mathit{LTL}^{\bigcirc}_{krom}$ OMAQ is in coNP.

To show coNP-hardness, given a unary NFA $\mathfrak{A} = (Q, \{a\}, \delta, q_0, F)$, we define an $\mathit{LTL}^{\bigcirc}_{core}$ ontology $\mathcal{O}_\mathfrak{A}$ with the axioms $X \to \bigcirc_F q_0$, $p \to \bigcirc_F q$ for $(p, a, q) \in \delta$, and $Y \wedge q \to \bot$ for $q \in F$:



For a $\{X, Y\}$-ABox $\mathcal{A}$ we have $\mathcal{O}, \mathcal{A} \models \exists x A(x)$ iff there are $m, n \in \mathbb{Z}$ such that $X(n) \in \mathcal{A}$, $Y(m) \in \mathcal{A}$, and $a^{m-n-1} \in \boldsymbol{L}(\mathfrak{A})$. Therefore, the OMAQ $(\mathcal{O}_\mathfrak{A}, A)$ is $\mathsf{FO}(<)$-rewritable over $\{X, Y\}$-ABoxes iff $\boldsymbol{L}(\mathfrak{A})$ is $\mathsf{FO}(<)$-definable. ◀

In our next result, the ontology language is weaker (core, which is contained in both Krom and Horn), but the queries are more expressive.

▶ **Theorem 11.** *$\mathsf{FO}(<)$-rewritability of $\mathit{LTL}^{\bigcirc}_{core}$ OMPEQs $\boldsymbol{q} = (\mathcal{O}, \varkappa)$ over Ξ-ABoxes is Π^p_2-complete.*

Proof. Let $\mathcal{B} = \{w_1 \dots w_k \in \Sigma^*_\Xi \mid \forall i\, |w(i)| > 0,\ \sum_i |w(i)| \leq |\varkappa|\}$. For $w \in \mathcal{B}$, consider the language $\boldsymbol{L}_w = \boldsymbol{L}(\emptyset^* w_1 \emptyset^* \dots \emptyset^* w_k \emptyset^*) \cap \boldsymbol{L}_\Xi(\boldsymbol{q})$. For $v, v' \in \Sigma^*_\Xi$, we write $v' \leq v$ if $|v| = |v'|$ and $v'_i \subseteq v_i$, for all i.

As $\boldsymbol{q}$ is an $\mathit{LTL}^{\bigcirc}_{core}$ OMPEQ, we can prove by induction on $|\varkappa|$ that $(\mathcal{O}, \mathcal{A}) \models \exists x \varkappa(x)$ iff $(\mathcal{O}, \mathcal{A}') \models \exists x \varkappa(x)$, for some $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| \leq |\varkappa|$. Therefore, for every $v \in \Sigma^*_\Xi$, we have $v \in \boldsymbol{L}_\Xi(\boldsymbol{q})$ iff there is $v' \leq v$ with $v' \in \boldsymbol{L}_w$ for some $w \in \mathcal{B}$. It follows that $\boldsymbol{L}_\Xi(\boldsymbol{q})$ is $\mathsf{FO}(<)$-definable iff $\boldsymbol{L}_w$ is $\mathsf{FO}(<)$-definable, for every $w \in \mathcal{B}$.

For $w = w_1 \dots w_k \in \mathcal{B}$ and $I = (i_0, \dots, i_k)$ let $v_{w,I} = \emptyset^{i_0} w_1 \emptyset^{i_1} \dots w_k \emptyset^{i_k}$. If $\boldsymbol{L}_w$ is $\mathsf{FO}(<)$-rewritable, then for every $j < k$, the set $\{l \mid v_{w,I'} \in \boldsymbol{L}_w, I' = (i_1, \dots, i_{j-1}, l, i_{j+1}, \dots, i_k)\}$ is finite or cofinite. For $c \in \mathbb{N}$ and I, let $I_{c \to j}$ be I with i_j replaced by $\min(c, i_j)$. We can find $c = 2^{O(|\mathcal{O}|)}$ such that $\boldsymbol{L}_w$ is $\mathsf{FO}(<)$-definable iff, for any $v_{w,I}$ with $\max(I) \leq 2c$ and any $j \leq |I|$, we have $v_{w,I} \in \boldsymbol{L}_w$ iff $v_{w,I_{c \to j}} \in \boldsymbol{L}_w$.

Now, $\boldsymbol{q}$ is not $\mathsf{FO}(<)$-rewritable iff there are $w \in \mathcal{B}$, I and j with $\max(I) \leq 2c$ and $j < |I|$ such that only one of $v_{w,I}$ and $v_{w,I_{c \to j}}$ is in $\boldsymbol{L}_w$. To check that $v_{w,I} \in \boldsymbol{L}_w$ can be done in NP, so $\mathsf{FO}(<)$-rewritability of $\boldsymbol{q}$ is in $\mathrm{coNP}^{\mathrm{NP}} = \Pi^p_2$.

The lower bound is established by reduction of ∀∃3CNF. ◀

If we increase the expressive power of $\mathit{LTL}^{\bigcirc}_{\mathit{core}}$ OMPEQs $\boldsymbol{q} = (\mathcal{O}, \varkappa)$ by allowing $\Box$-operators in $\varkappa$, the problem of deciding $\mathsf{FO}(<)$-rewritability becomes more complex:

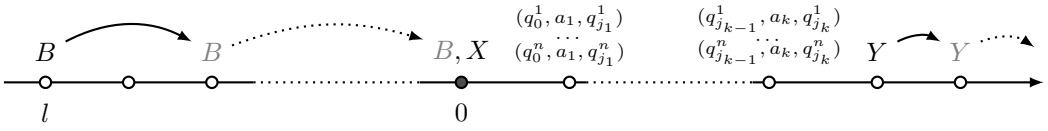
▶ **Theorem 12.** *$\mathsf{FO}(<)$-rewritability of $\mathit{LTL}^{\bigcirc}_{\mathit{core}}$ OMPQs over Ξ-ABoxes is PSpace-complete.*

Proof. The upper bound is by Theorem 8. The lower one is proved by reduction of the PSpace-complete DFA intersection problem [32]. Let $\mathfrak{A}_i = (Q_i, \Sigma, \delta_i, q^i_0, F_i)$, $i \leq n$, be DFAs that do not accept ε and have disjoint $Q_i = \{q^i_j\}$. We let $\Xi = \{X, Y, B\} \cup \bigcup_{i \leq n} \delta_i$ and $\varkappa = B \land X \land \Box_F\big(\big(\bigwedge_{i \leq n} \bigvee_{(q^i_k, a, q^i_l) \in \delta_i} (q^i_k, a, q^i_l)\big) \lor Y\big)$. The ontology $\mathcal{O}$ contains the following axioms: $B \to \bigcirc_F \bigcirc_F B$, $Y \to \bigcirc_F Y$, $X \land \bigcirc_F Y \to \bot$, $X \land \bigcirc_F (q^i_k, a, q^i_l) \to \bot$ for $k \neq 0$, $\bigcirc_F Y \land (q^i_k, a, q^i_l) \to \bot$ for $q^i_l \notin F_i$, $(q^i_k, a, q^i_l) \land (q^i_m, b, q^i_n) \to \bot$ for all $k \neq m$ or $l \neq n$, $(q^i_k, a, q^i_l) \land \bigcirc_F (q^i_m, b, q^i_n) \to \bot$ for all $l \neq m$, $(q^i_k, a, q^i_l) \land (q^j_m, b, q^j_n) \to \bot$ for all $a \neq b$.

Let $\boldsymbol{q} = (\mathcal{O}, \varkappa)$. We prove that $\bigcap_{i \leq n} L(\mathfrak{A}_i) \neq \emptyset$ iff $\boldsymbol{q}$ is not $\mathsf{FO}(<)$-rewritable.

($\Rightarrow$). Suppose $w = a_1 \dots a_k \in \bigcap_{i \leq n} L(\mathfrak{A}_i)$ and let $(q^i_0, a_1, q^i_{l_1}) \dots (q^i_{l_{k-1}}, a_k, q^i_{l_k})$ be the run of the ith automaton on w. Let $R^i_j = (q^i_{l_{j-1}}, a_j, q^i_{l_j})$.

Consider $\mathcal{A}_w = \{X(0)\} \cup \big(\bigcup_{i \in [1,n]} \bigcup_{j \in [1,k]} \{R^i_j(j)\}\big) \cup \{Y(k+1)\}$. The answer to $\boldsymbol{q}$ over $\mathcal{A}_w \cup \{B(l)\}$ is yes iff $l \leq 0$ and even, because only in this case we have $\mathcal{O}, \mathcal{A}_w \cup \{B(l)\} \models B(0)$ and consequently $\mathcal{O}, \mathcal{A}_w \cup \{B(l)\} \models \varkappa(0)$. Since the set $\{l \mid \mathcal{O}, \mathcal{A}_w \cup \{B(l)\} \models \exists x \varkappa(x)\}$ is not $\mathsf{FO}(<)$-definable, the OMQ $\boldsymbol{q}$ is not $\mathsf{FO}(<)$-rewritable. The picture below illustrates the structure of the ABox $\mathcal{A}_w \cup \{B(l)\}$:



($\Leftarrow$). Suppose $\bigcap_{i \leq n} L(\mathfrak{A}_i) = \emptyset$. Then, for any ABox $\mathcal{A}$ and k, we have $\mathcal{O}, \mathcal{A} \models \varkappa(k)$ iff the ABox $\mathcal{A}$ is inconsistent with $\mathcal{O}$, by the construction of $\boldsymbol{q}$. We can then easily construct the $\mathsf{FO}(<)$-rewriting of $\boldsymbol{q}$ by encoding the inconsistency axioms of $\mathcal{O}$ by $\mathsf{FO}(<)$-formulas. ◀

6 Conclusions

Motivated by ontology-based access to temporal data – a paradigm relying on FO-rewritability of ontology-mediated queries – we considered the problem of determining the optimal rewritability type and data complexity of answering any given *LTL* OMQ. We showed that this problem is closely related to deciding $\mathsf{FO}(<)$-, $\mathsf{FO}(<, \equiv)$- and $\mathsf{FO}(<, \mathsf{MOD})$-definability of regular languages given by DFAs, NFAs and 2NFAs of different size. Based on this correspondence, we showed how the clausal form of ontology axioms in OMQs, the temporal operators involved and the type of queries are reflected in the structure of automata accepting the OMQs' yes-data instances and the complexity of deciding their FO-definability.

Interesting open problems include understanding the impact of the $\Box$-operators in linear and core ontologies on the complexity of deciding FO-rewritability, extending our analysis to *MTL*-ontologies where OMQs are not necessarily $\mathsf{FO}(\mathsf{RPR})$-rewritable, and so are outside of NC^1, and to 2D combinations of *LTL* with description logics, in particular *DL-Lite*.

It would be also interesting to experiment with algorithms for checking $\mathcal{L}$-rewritability of *LTL* OMQs and constructing rewritings into various types of SQL queries. For some $\mathit{LTL}^{\Box\bigcirc}_{\mathit{bool}}$ OMQs and linear $\mathit{LTL}^{\bigcirc}_{\mathit{horn}}$ OMQs, the best target rewriting language is $\mathsf{FO}(<, \mathsf{RPR})$, which can only be captured in SQL with recursion or procedural extensions that are not always supported by RDBMSs and are less efficient. The $\mathsf{FO}(<, \mathsf{MOD})$-rewritable OMQs can be implemented in the most basic SQL using the `count` operator, while $\mathsf{FO}(<, \equiv)$-rewritable ones do not need it.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Foto N. Afrati and Christos H. Papadimitriou. The parallel complexity of simple logic programs. *J. ACM*, 40(4):891–916, 1993. doi:10.1145/153724.153752.

3 A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research (JAIR)*, 36:1–69, 2009.

4 A. Artale, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. The complexity of clausal fragments of LTL. In *Proc. of the 19th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2013*, volume 8312 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2013.

5 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 Sven Schewe, Thomas Schneider, and Jef Wijsen, editors, *24th International Symposium on Temporal Representation and Reasoning, TIME 2017, October 16-18, 2017, Mons, Belgium*, volume 90 of *LIPIcs*, pages 1:1–1:37. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.TIME.2017.1.

6 Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. First-order rewritability of ontology-mediated queries in linear temporal logic. *CoRR*, abs/2004.07221, 2020. arXiv:2004.07221.

7 J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9–10):1620–1654, 2011.

8 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

9 David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *J. Comput. Syst. Sci.*, 44(3):478–499, 1992. doi:10.1016/0022-0000(92)90014-A.

10 David A. Mix Barrington and Denis Thérien. Finite monoids and the fine structure of NC^1. *J. ACM*, 35(4):941–952, 1988. doi:10.1145/48014.63138.

11 Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 293–304. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.36.

12 M. Bennett, G. Martin, K. O'Bryant, and A. Rechnitzer. Explicit bounds for primes in arithmetic progressions. *Illinois Journal of Mathematics*, 62(1–4):427–532, 2018.

13 L. Bernátsky. Regular expression star-freeness is PSPACE-complete. *Acta Cybern.*, 13(1):1–21, 1997. URL: http://www.inf.u-szeged.hu/actacybernetica/edb/vol13n1/Bernatsky_1997_ActaCybernetica.xml.

14 M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. *ACM Transactions on Database Systems*, 39(4):33:1–44, 2014.

15 Stefan Borgwardt, Walter Forkel, and Alisa Kovtunova. Finding new diamonds: Temporal minimal-world query answering over sparse aboxes. In Paul Fodor, Marco Montali, Diego Calvanese, and Dumitru Roman, editors, *Rules and Reasoning - Third International Joint Conference, RuleML+RR 2019, Bolzano, Italy, September 16-19, 2019, Proceedings*, volume 11784 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2019. doi:10.1007/978-3-030-31095-0_1.

16 A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 193:87–128, 2012. doi:10.1016/j.artint.2012.08.002.

17 D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

18 Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991. URL: https://core.ac.uk/download/pdf/82662203.pdf.

19 C. Civili and R. Rosati. A broad class of first-order rewritable tuple-generating dependencies. In *Proc. of the 2nd Int. Datalog 2.0 Workshop*, volume 7494 of *Lecture Notes in Computer Science*, pages 68–80. Springer, 2012.

20 Kevin J. Compton and Claude Laflamme. An algebra and a logic for NC^1. *Inf. Comput.*, 87(1/2):240–262, 1990.

21 Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *STOC*, pages 477–490, 1988. doi:10.1145/62212.62259.

22 Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

23 Volker Diekert and Paul Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

24 Cristina Feier, Antti Kuusisto, and Carsten Lutz. Rewritability in monadic disjunctive datalog, MMSNP, and expressive description logics. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:15)2019.

25 M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Logic*, 2(1):12–56, 2001.

26 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

27 D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-Dimensional Modal Logics: Theory and Applications*, volume 148 of *Studies in Logic*. Elsevier, 2003.

28 Olga Gerasimova, Stanislav Kikot, Agi Kurucz, Vladimir V. Podolskii, and Michael Zakharyaschev. A data complexity and rewritability tetrachotomy of ontology-mediated queries with a covering axiom. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 403–413, 2020. doi:10.24963/kr.2020/41.

29 Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Datalog rewritability of disjunctive datalog programs and non-Horn ontologies. *Artif. Intell.*, 236:90–118, 2016. doi:10.1016/j.artint.2016.03.006.

30 Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Computer Science Department, University of California at Los Angeles, USA, 1968.

31 Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998. URL: https://www.worldcat.org/oclc/312898417.

32 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 254–266, 1977. doi:10.1109/SFCS.1977.16.

33 Agi Kurucz, Vladislav Ryzhikov, Yury Savateev, and Michael Zakharyaschev. Deciding FO-rewritability of regular languages, 2021. arXiv:2105.06202.

34 L. Libkin. *Elements Of Finite Model Theory*. Springer, 2004.

35 C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal description logics: A survey. In *Proc. of the 15th Int. Symposium on Temporal Representation and Reasoning (TIME 2008)*, pages 3–14, 2008.

36 Carsten Lutz and Leif Sabellek. Ontology-mediated querying with the description logic EL: trichotomy and linear datalog rewritability. In *Proc. of the 26th Int. Joint Conf. on Artificial Intelligence (IJCAI 2017)*, pages 1181–1187, 2017.

37 Jerzy Marcinkowski. DATALOG sirups uniform boundedness is undecidable. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 13–24. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561299.

38 Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal on Data Semantics*, 10:133–173, 2008.

39 Alexander Rabinovich. A proof of Kamp's theorem. *Logical Methods in Computer Science*, 10(1), 2014.

40 Vladislav Ryzhikov, Yury Savateev, and Michael Zakharyaschev. Deciding FO-rewritability of ontology-mediated queries in linear temporal logic, 2021. URL: https://www.dcs.bbk.ac.uk/~vlad/time21-full.pdf.

41 Jacques Stern. Complexity of some problems from the theory of automata. *Inf. Control.*, 66(3):163–176, 1985. doi:10.1016/S0019-9958(85)80058-9.

42 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

43 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhauser Verlag, 1994.

44 Jeffrey D. Ullman and Allen Van Gelder. Parallel complexity of logical query programs. *Algorithmica*, 3:5–42, 1988. doi:10.1007/BF01762108.

45 Ron van der Meyden. Predicate boundedness of linear monadic datalog is in PSPACE. *Int. J. Found. Comput. Sci.*, 11(4):591–612, 2000. doi:10.1142/S0129054100000351.

46 Moshe Y. Vardi. Decidability and undecidability results for boundedness of linear recursive queries. In Chris Edmondson-Yurkanan and Mihalis Yannakakis, editors, *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas, USA*, pages 341–351. ACM, 1988. doi:10.1145/308386.308470.

47 Moshe Y. Vardi. Automata-theoretic techniques for temporal reasoning. In Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 971–989. North-Holland, 2007. doi:10.1016/s1570-2464(07)80020-6.

48 Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. of the Symposium on Logic in Computer Science (LICS'86)*, pages 332–344, 1986.

49 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. Datalogmtl over the integer timeline. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 768–777, 2020. doi:10.24963/kr.2020/79.

50 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. Tractable fragments of datalog with metric temporal operators. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1919–1925. ijcai.org, 2020. doi:10.24963/ijcai.2020/266.

51 Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 5511–5519. ijcai.org, 2018. doi:10.24963/ijcai.2018/777.

52 Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. Virtual knowledge graphs: An overview of systems and use cases. *Data Intell.*, 1(3):201–223, 2019. doi:10.1162/dint_a_00011.

A Neuro-Symbolic Approach to Structured Event Recognition

Gianluca Apriceno
University of Trento, Italy
Fondazione Bruno Kessler, Italy

Andrea Passerini
University of Trento, Italy

Luciano Serafini
Fondazione Bruno Kessler, Italy

Abstract

Events are structured entities with multiple components: the event type, the participants with their roles, the outcome, the sub-events etc. A fully end-to-end approach for event recognition from raw data sequence, therefore, should also solve a number of simpler tasks like recognizing the objects involved in the events and their roles, the outcome of the events as well as the sub-events. Ontological knowledge about event structure, specified in logic languages, could be very useful to solve the aforementioned challenges. However, the majority of successful approaches in event recognition from raw data are based on purely neural approaches (mainly recurrent neural networks), with limited, if any, support for background knowledge. These approaches typically require large training sets with detailed annotations at the different levels in which recognition can be decomposed (e.g., video annotated with object bounding boxes, object roles, events and sub-events). In this paper, we propose a neuro-symbolic approach for structured event recognition from raw data that uses "shallow" annotation on the high-level events and exploits background knowledge to propagate this supervision to simpler tasks such as object classification. We develop a prototype of the approach and compare it with a purely neural solution based on recurrent neural networks, showing the higher capability of solving both the event recognition task and the simpler task of object classification, as well as the ability to generalize to events with unseen outcomes.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Computing methodologies → Activity recognition and understanding

Keywords and phrases Event recognition, learning and reasoning, neuro-symbolic integration

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.11

Funding This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Events are structured entities with multiple components and relations with other entities [18]. The most important components of an event are the event type, the participants with their roles, the sub-events, and the event outcome. Therefore, the approaches for full fledged event recognition should be able to extract the information about all the components of the events that happen in a data sequence. To this aim, a system for event detection should solve a number of different simpler tasks like recognizing the objects involved in the events and their roles, the outcome of the events as well as the sub-events. In this context, having background knowledge about the event structure, specified in logic languages, could be very useful to solve the aforementioned challenges. However, looking at [22], one can see that the majority of neural approaches (also knows as sub-symbolic) applied in event recognition

strictly rely on the features learnt by the underlying networks with limited, if any, support for background knowledge. Furthermore, the training of the underlying networks of these approaches requires a large amount of training data with a detailed supervision on all the events' components (e.g., a video annotated with events, sub-events, object roles and object bounding boxes). Alternatively, one could think to have data with an annotation limited to the occurrence of an event (i.e., a "shallow" annotation) and exploit the background knowledge to infer information on the event components. For example, if a video clip is annotated with the event "John is preparing a cappuccino to Mary", one can infer from the background knowledge that the video is showing at least two people, one male and one female, that John is preparing the cappuccino by mixing milk and coffee into two cups, etc. All of these inferred facts can be used as supervision to neural networks to solve the simpler tasks defined above and to recognize the structured event as well. In this case, neuro-symbolic frameworks e.g., DeepProbLog [13], Logic Tensor Networks [19], LYRICS [14] that combine low-level neural perceptions with logic reasoning (also know as symbolic) seem to be suitable approaches to achieve these objectives. In this paper, we propose a neuro-symbolic approach for structured event recognition from raw data that uses "shallow" annotation on the high-level events and exploits background knowledge to propagate this supervision to simpler tasks such as object classification. We develop a prototype of the approach and compare it with a purely neural solution based on a recurrent neural network, showing the higher capability of solving both the event recognition task and the simpler task of object classification, as well as the ability to generalize to events with unseen outcomes. The detailed contributions of the paper are the following:

1. a formal definition of the problem of structured event recognition from raw data sequences with "shallow" annotations and of a neuro-symbolic solution combining low-level neural-based predictions with high-level reasoning;
2. a framework for automatically generating simple videos containing events which are fully annotated;
3. a prototypical neuro-symbolic recognition approach based on DeepProbLog;
4. an experimental evaluation that compares our approach with a purely neural solution, showing the advantage of explicitly using background knowledge.

The rest of the paper is organized as follows: Sections 2 formally defines the problem of structured event recognition with "shallow" annotation; Section 3 presents our proposed solution; Section 4 briefly reviews the state of the art approaches that have been proposed in the context of event recognition, and presents some of the most well-known neuro-symbolic frameworks; Section 5 describes the event generation framework; Section 6 presents the experimental setting; Section 7 describes the neural LSTM approach and our prototype approach based on DeepProbLog; Section 8 reports the experimental results; Finally, Section 9 draws some concluding remarks and discusses directions for future work.

2 Problem definition

Let $\mathcal{L}$ be a first order language with three sorts, $\mathbb{O}$, $\mathbb{E}$, and $\mathbb{T}$. Terms of sort $\mathbb{O}$ denote objects, terms of sort $\mathbb{E}$ denote events, and terms of sort $\mathbb{T}$ denote time-points. The language contains the constants $0, 1, 2, \ldots$ of sort $\mathbb{T}$, used to name time points and the binary relation $<: \mathbb{T} \times \mathbb{T} \rightarrow \{\top, \bot\}$. The language $\mathcal{L}$ also contains a set of predicates $\mathcal{P}$ of sort $\mathbb{O}^k \rightarrow \{\bot, \top\}$, which are used to describe the time invariant properties and relations between objects. $\mathcal{L}$ contains also a set of function symbols $\mathcal{E}$ of sort $\mathbb{O}^k \rightarrow \mathbb{E}$ that are used to describe events that involve a (possible empty) tuple of objects. We also have a relationship $outcome(\mathbb{E}, \mathbb{O})$ that

is used to describe the fact that the outcome of an event is an object. Finally, $\mathcal{L}$ contains the predicate $happens(\mathbb{E}, \mathbb{T}, \mathbb{T})$ that is used to describe the fact that a certain event happens within an interval of time. For example, the formula $\exists x.happens(drop(John, x), t_1, t_2)$ states that *John* drops an object x at some time between t_1 and t_2. Notice that events can "create" new objects, for example the result of mixing milk and coffee is a cappuccino. This is expressed by the formula $milk(x) \wedge coffee(y) \rightarrow outcome(mix(x, y), z) \wedge cappuccino(z)$. A *narrative* is an interpretation $\mathcal{I}$ of the language $\mathcal{L}$, where the terms of sort $\mathbb{T}$ are interpreted in the set of natural numbers and $<$ in the usual linear order. The terms of sort $\mathbb{O}$ are interpreted in a domain of objects $\Delta_{\mathbb{O}}$ and those of $\mathbb{E}$ are interpreted in a domain of events $\Delta_{\mathbb{E}}$. Since we are interested in finite narratives, i.e., narratives that involve a finite number of objects and a finite number of events and time points, we can specify a narrative by using the Herbrand Base. In particular, for every $k > 0$ we define a k-narrative as a pair $\mathcal{N} = (\mathcal{C}, \mathcal{F})$ where:

- $\mathcal{C}$ is a finite set of new constants for objects of type $\mathbb{O}$;
- $\mathcal{F}$ is a subset of ground atoms in the language of $\mathcal{L}$ extended with the constants in $\mathcal{C}$ and the constants $0, 1, \ldots, k$ of sort $\mathbb{T}$; such that: if $happens(e, t_1, t_2) \in \mathcal{F}$ then $t_1 \leq t_2$.

Our main aim is to reconstruct a narrative from a data stream using some neuro-symbolic method that is capable of combining low-level data processing capabilities with the ability to leverage background knowledge about the structure of events. More formally, let $\boldsymbol{D} = \{\boldsymbol{d}_i\}_{i=1}^{k}$ be a data sequence of length k, where each $\boldsymbol{d}_i \in \mathcal{X}$ is a low-level representation for sequence element i (like a real-valued vector, matrix or tensor). Our main objective is to generate a k-narrative that describes the events that happen in $\boldsymbol{D}$, when they happens, their participants, and their outcomes. In other words we want to extract from $\boldsymbol{D}$:

- a set of objects;
- the properties and the relations between the objects;
- the set of events that happen;
- the objects (arguments) that are involved in each event that happens;
- the outcomes of the events that happen.

▶ **Example 1.** Let $\boldsymbol{D}$ be a video showing two people, one moving, leaving a bag and then moving away, and the other standing. We would like to produce the following narrative:

$$\mathcal{C} = \{p_1, p_2, b_1\} \qquad \mathcal{F} = \left\{ \begin{array}{c} person(p_1), person(p_2), bag(b_1), \\ happens(move(p_1), 0, 4), happens(drop(p_1, b_1), 4, 5) \\ happens(move(p_1), 5, 7), \end{array} \right\}$$

The type of supervision we suppose to have, in order to learn a model that extracts narratives from data, is partial and consists of a set of n data sequences labelled with *some* (not necessarily all the) ground facts about events happening in the sequence:

$$\left\{ \boldsymbol{D}^{(i)}, \mathcal{F}_p^{(i)} \right\}_{i=1}^{n}$$

where $\boldsymbol{D}^{(i)} = \{\boldsymbol{d}_j^{(i)}\}_{j=1}^{k}$ is a data sequence and $\mathcal{F}_p^{(i)}$ is a set of positive and negative literals, denoting a subset of the events that happen or don't happen in $\boldsymbol{D}^{(i)}$. Notice that we do not need to have a complete labelling for all the events. Furthermore, notice that the supervision provided via $\mathcal{F}_p^{(i)}$ also provide a supervision for the subset of objects $^{(i)}$ that appear in the data stream $\boldsymbol{D}^{(i)}$, which is the set of constants of type $\mathbb{O}$ that appear in the positive literals of $\mathcal{F}_p^{(i)}$.

Events can be related to each other, and structured events can be defined in terms of simpler ones.

▶ **Example 2.** Let *potential_threat* represent a structured event corresponding to a potential threat represented by something happening in a video like the one in the previous example. The threat could be modelled by the following formula:

$$\begin{aligned} &\mathit{happens}(\mathit{potential\_threat}, t_0, t_3) \leftrightarrow \\ &\qquad \exists x, y, t_1, t_2 . \mathit{person}(x) \wedge \mathit{bag}(y) \wedge \\ &\qquad\qquad \mathit{happens}(\mathit{move}(x), t_0, t_1) \wedge \\ &\qquad\qquad \mathit{happens}(\mathit{drop}(x, y), t_1, t_2) \wedge \\ &\qquad\qquad \mathit{happens}(\mathit{move}(x), t_2, t_3) \end{aligned} \tag{1}$$

An example of supervision in this context could be a set of videos $\boldsymbol{D}^{(1)}, \boldsymbol{D}^{(2)}, \ldots, \boldsymbol{D}^{(m)}$ of length k, each of which is annotated with either the single fact $\mathit{happens}(\mathit{potential\_threat}, 0, k)$ or with the single fact $\neg \mathit{happens}(\mathit{potential\_threat}, 0, k)$.

3 Proposed solution

Looking at the examples of the previous section, we observe that a structured event can be expressed in terms of simple events using logical languages. Simple events include the objects participating in the structured event, their relationships and their individual actions. Therefore, the correct recognition of the simple events combined with the definition of the structured event at the logical level will lead to the recognition of the structured event.
Our proposed approach has two aims, respectively learning to recognize the structured event happening in a data sequence and the simple events that compose it. To achieve these aims, we provide both background knowledge on the domain, expressed in terms of logical formulas, and "shallow" annotations on the structured event, like the one for the *potential_threat* example of section 2. In order to solve our problem, we have to complete three tasks:

object detection: in order to build the narrative, we have to find the set of objects $\mathcal{C}$[1] that appear in a data sequence $\boldsymbol{D}$.

object classification and relation detection: We also have to classify the objects in their types, e.g., a chair, a person, ..., and we have to detect relations between objects, e.g. if the person holds a bag or not.

event recognition: we have to recognize the events that happen in the video.

The traditional approach to solve the problem is to use a pipeline, where the above tasks are solved sequentially and the result of the solution of the previous task is provided as input to the next task. However, this requires supervision at all levels, the objects in the data, their class and the events. We instead have only partial supervision on some events.
In our solution we propose to have a fully end-to-end approach in which both the supervision on data and the background knowledge are used to train some neural networks for more data driven tasks such as object detection and classification. We therefore suppose to have the following components:

- A neural network $\mathrm{Det}_{\mathrm{nn}}$ that takes as input a sequence $\boldsymbol{D}$ and returns a set of objects $\mathcal{C}$ each of which is associated with a set of numeric features $f(o)$. For example, if $\boldsymbol{D} = \{\boldsymbol{d}_i\}_{i=1}^{k}$ is a video, then $f(o)$ contains the bounding boxes of object o at each frame $\boldsymbol{d}_i$ and the crop of the image on the bounding box for each frame;
- for some (not necessarily all) object predicates we have a network P_{nn} that takes as input the features of an n-tuple of objects $f(o_1), f(o_2), \ldots, f(o_n)$ and returns a sequence in $[0, 1]^k$ which represents the level of truth or probability of truth of the predicate at

[1] objects and constants are used interchangeably

each time point $0 \leq i \leq k$. For example, for the *person* predicate in the examples in the previous section we would have a network that given a sequence of cropped images outputs for each image the probability that it contains a person.

All the outputs of the neural networks defined above can be combined with the background knowledge which is described in terms of the axioms, such as equation (1). The way in which this combination is achieved could be based both on probabilistic semantics or on fuzzy semantics. At this stage we do not want to commit on one or the other. A list of neuro-symbolic approaches that can be adopted to implement our architecture is provided in the related work section. In the following we provide a proof-of-concept implementation of the architecture described above using DeepProbLog [13].

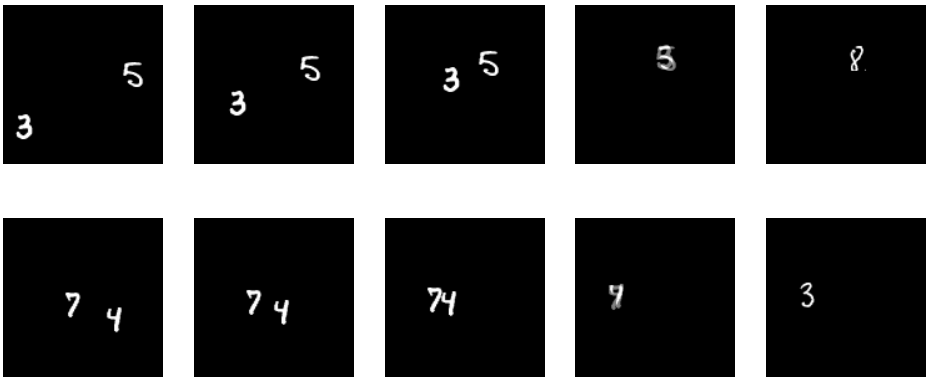
4 Related work

Event recognition from data streams like videos, audio and text is a well-studied problem. With the advent of deep learning, most sub-symbolic approaches to event recognition have moved from processing hand-crafted features to automated representation learning from raw data (see [1] and [22] for a survey). In order to be effective, however, these approaches require large and deeply annotated datasets, where supervision on the different sub-tasks defining the event recognition problem is available for training. On the other hand, purely symbolic approaches to event recognition [4] allow to explicitly define the conditions that lead to the occurrence of an event, but may fail in the presence of noise. As a consequence, symbolic approaches that can deal with uncertainty have emerged (see [2]). In [20, 3], the authors recognize structured events from a combination of low-level events. This recognition was done considering only uncertainty on low-level events and using ProbLog [17] as a probabilistic reasoner. Low-level predictions are however assumed to be given and no attempt is made at predicting low-level events from raw data. A number of approaches [9, 10, 23] combine symbolic reasoning with pre-trained neural networks used to recognize low-level events. These approaches require full supervision at the different levels and cannot be applied in the setting we address in this work.

In this work we aim at combining the advantages of low-level neural processing and high-level symbolic reasoning to achieve effective event recognition from small datasets and shallow annotations. Neuro-symbolic integration is an active area of research and multiple frameworks have been proposed. These frameworks combines low-level neural perception with high level reasoning in different ways. Many approaches, like LYRICS [14] or Logic Tensor Networks [19] combine neural predictors with fuzzy logic. NeurASP [24] combines neural networks with answer set programming, while [6] and [15], use neural networks to define potentials in probabilistic graphical models. Among existing frameworks, DeepProbLog [13] is particularly appealing in terms of expressivity. DeepProbLog extends the probabilistic programming language ProbLog with neural predicates, achieving an elegant and powerful combination of neural networks, logic and probability. We thus leverage DeepProbLog as the underlying integration framework for our neuro-symbolic event recognition prototype.

5 Event generation framework

Most of the available datasets for event recognition provide only limited annotations on some but not all the elements of an events. For example, in the context of event recognition in video there are dataset like Olympic Sport [16] and UCF101 [21] that provide annotation only on the event happening in the video. Datasets like CAVIAR [5], MEVA [7], Cooking [11] and



(a)

(b)

Figure 1 Example of events generated by the framework: join_add (a) and join_sub (b).

HiEve [12] provide a richer annotation, e.g., objects classes, object locations and distinction between simple and structured events, but they introduce a level of complexity in the visual part that requires pre-trained models for processing low level features. Here we are interested in developing a neuro-symbolic system that is trainable end-to-end, where the learning of the low-level processing is influenced by the high-level knowledge. Furthermore, the above mentioned datasets were manually curated, and cannot be extended to consider newly defined structured events without a tedious process of data collection and manual annotation. We, instead, would like to be able to quickly generate new data streams containing new events so as to support a fast prototyping and testing of recognition architectures. For these reasons, we have implemented a video generator of events involving mnist digits. The generator allows to generate videos of different length and with a different number of digits that interact with each other, together to the narrative describing the objects and events in the video. The generator uses an object predicate $digit(x, v)$ to indicate that v is the value corresponding to object x. Concerning events, we distinguish between simple events that involve single digits and structured ones that involve combinations of digits. The simple events we defined are:

- $appear(x)$: a digit x appears in the video
- $disappear(x)$: a digit x disappears from the video
- $enter(x)$: a digit x enters in the video
- $exit(x)$: a digit x exits from the video

The difference between $appear/disappear$ and $enter/exit$ is that in the former case the digit is always fully visible when in the video, while in the latter case the digit is only partially visible upon entering/exiting. Example of structured events definable in the framework are:

- $join\_add(x, y)$: two digits, respectively x and y, approach each other, overlap and then the digit that is the result of the sum of the two digits appears, i.e., $outcome(join\_add(x, y), z)$ with $digit(x, v_x), digit(y, v_y), digit(z, v_z)$ and $v_z = v_x + v_y$. Note that this event can only happen if the sum of the two digits is ≤ 9.
- $join\_sub(x, y)$: two digits, respectively x and y, approach each other, overlap and then the digit that is the result of the difference between the two digits appears, i.e., $outcome(join\_sub(x, y), z)$ with $digit(x, v_x), digit(y, v_y), digit(z, v_z)$ and $v_z = max(v_x, v_y) - min(v_x, v_y)$.
- $split(x)$: a digit x splits into two digits whose sum or difference gives the value of x, i.e., $outcome(split(x), (y, z))$ with $digit(x, v_x), digit(y, v_y), digit(z, v_z)$ and $v_x = v_y + v_z$ or $v_x = max(v_y, v_z) - min(v_y, v_z)$.

Some examples of structured events produced by the generator are shown in Figure 1. For simplicity, each object is assumed to participate in at most one simple event and one structured event for each frame. For each video, a narrative file is also produced that contains the following information for each digit:

- the name: a label
- the class: the corresponding mnist class
- the position: x and y coordinates inside the frame
- the simple event (if any) the digit is involved in
- the structured event (if any) the digit is involved in

6 Experimental setting

Our experimental evaluation is aimed at verifying whether a neuro-symbolic solution has an advantage in recognizing structured events with respect to a fully neural approach. In addition to the capability of correctly classifying each video into the corresponding structured event, we aim at evaluating the ability to learn to correctly classify the underlying objects (the digits) as well as the ability to generalize to unseen outcomes (e.g., the result of a `join_add` being a digit for which no explicit supervision was ever received). The scenario and learning setting we created to this aim are described in the following.

6.1 Scenario

The scenario consists of videos produced with the event generation framework described in Section 5. Each video consists of 10 frames, each frame showing one or two digits. Digits can appear anytime within the first half of the video, and only disappear if they join together. When present, the digits are always completely visible, apart from the frames in which they overlap with each other (e.g., right before a join). We generated three types of videos:

- *join_add*
- *join_sub*
- *no_join*

The first two refer to videos where the corresponding structured event as discussed in Section 5 takes place. The resulting digit can stay in the same position or move. To avoid ambiguities, we refrain from generating videos where one of the two operands is a zero. As consequence, the only way to get a zero is in a *join_sub* when both digits are equal. The third type refers to videos where neither of the two structured events takes place. In this case the video contains two arbitrary digits that wander around with no restrictions, possibly overlapping with each other.

6.2 Learning setting

Our goal is to test the ability of the different approaches to learn to recognize events with partial supervision. The idea is to provide supervision in terms of the structured event taking place (if any) and the outcome of the event (i.e., the result of the addition/subtraction). Supervision is thus provided in terms of sets like the following:

$$\{happens(join\_add(x,y),1,T), outcome(join\_add(x,y),z), digit(z,4)\}$$
$$\{happens(join\_sub(x,y),1,T), outcome(join\_sub(x,y),z), digit(z,2)\}$$
$$\{\neg happens(join\_add(x,y),1,T), \neg happens(join\_sub(x,y),1,T)\}$$

Note that this type of feedback provides information on the classification of the underlying objects, even if only when a join takes place, and only for the digit which is the result of the join. We thus build the task to additionally test the ability of the methods to *generalize* to unseen outcomes, i.e., digits that were never observed as the result of a join during training (or validation). For the sake of conciseness, in the following we will refer to the combination of structured event and outcome as the class of a video (with *no_join* being the class of a video where no join occurs). To generate the videos we first split the original MNIST dataset into training, validation and test set. Then, separately for each set, we randomly picked digits to generate a set of videos for each of the video classes, making sure that each class had the same number of videos. We generated training and validation videos containing *no_join*, *join_add* with outcome from 2 to 7 and *join_sub* with outcome from 0 to 7, for a total of 15 video classes. Test videos contain the same classes as the training and validation ones plus *join_add* with outcome 8 and 9 and *join_sub* with outcome 8, for a total of 18 classes. We generated 1500 videos for training, 150 for validation and 180 for testing, so that each class always contains 100 videos.

7 Event recognition approaches

In this section, we describe the learning approaches that we used to solve the aforementioned task. We start presenting the low-level neural networks that we use for object detection and classification and proceed describing the fully neural and the neuro-symbolic approaches that build on them. The object detector is pre-trained, while the object classifier is trained end-to-end both in the fully neural and neuro-symbolic approaches. Training is performed for 35 epochs, using the Adam optimizer with a learning rate of 0.001 and early stopping on the validation set. Training for more epochs does not lead to improvements in recognition quality.

7.1 Object detector and classifier

The object detector is a neural network that extracts (processed) patches from frames. Its architecture is shown in Figure 2 for the case of a single frame, as the same structure is repeated for all frames in a video. Its main module is a standard convolutional neural network that consists of two convolutional layers, each followed by a max-pooling layer, and two fully-connected layers. ReLU are used as activation in all layers apart from the output layer where a sigmoid is used. The module takes as input a frame of size 128×128 and gives as output a vector of length 6:

$$\boldsymbol{o}_{det} = \langle v_1, v_2, x_1, x_2, y_1, y_2 \rangle$$

where $v_i \in [0, 1]$ indicates whether the i-th digit is present in the frame and $x_i, y_i \in [0, 1] \times [0, 1]$ are the normalized digit coordinates (digits are ordered according to the distance between their predicted coordinates and the origin). The two patches corresponding to the coordinates are extracted from the frame, and their content is multiplied by the value of their visibility flag. In so doing, the detector outputs "soft" patches, that depending on the value of the visibility flag range from the patch itself ($v_i = 1$) to a completely black patch ($v_i = 0$).

The digit classifier has the same architecture as the main module of the detector, with the sigmoid replaced by a softmax in the output layer. The classifier takes as input an image of size of 28×28 which corresponds to a processed patch extracted by the detector and returns as output a vector of length 11, where the first 10 element refers to the 0-9 digits and the last one indicates the absence of a digit. This module is repeated for all patches and all frames of the input video.

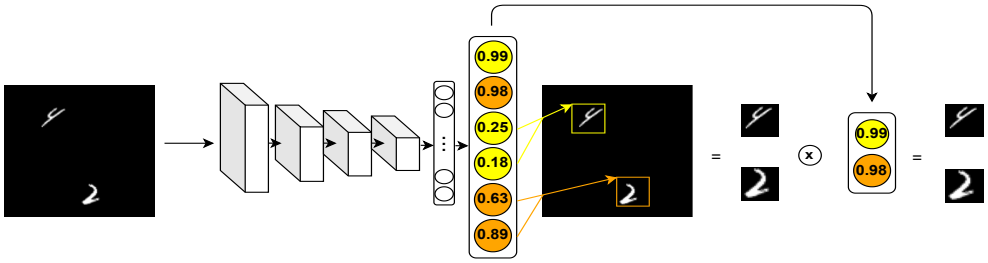


Figure 2 Mnist digit detector.

7.2 Fully neural approach

The fully neural approach combines the predictions of the digit detector and classifier on the different frames using an LSTM recurrent neural network [8]. The overall architecture is shown in Figure 3. For each frame in the input video, the detector extracts a pair of patches and sends them to the digit classifier. The predictions of the classifier are concatenated with the visibility and coordinate predictions from the detector and fed to an LSTM cell. After processing the entire input sequence, the LSTM outputs a prediction in three classes, $join\_add, join\_sub$ and no_join. The outcome of the join event is recovered from the output of the digit classifier on the first patch of the last frame. If the class with the highest prediction is no_join, the outcome prediction is ignored.

7.3 Neuro-symbolic approach

We developed a neuro-symbolic approach for structured event recognition using the DeepProbLog [13] framework. This framework can be seen as a neural extension of the probabilistic extension of Prolog, ProbLog [17]. Like ProbLog, the knowledge about the domain is encoded as a set of logical rules (i.e., horn clauses). In addition, DeepProbLog introduces neural predicates that allow to instantiate facts as outputs of neural predicates processing raw data. The neural extension is realized by enhancing ProbLog with a primitive that allows to declare neural predicates:

$$nn(n_{id}, \boldsymbol{X}_s, Y, \boldsymbol{y}_s)$$

where nn is a reserved functor used to declare a neural predicate, n_{id} is an identifier for the underlying neural network, $\boldsymbol{X}_s$ denotes a sequence of n input variables, Y is the output variable, and $\boldsymbol{y}_s$ denotes a sequence of m possible values that Y can assume. Training of these neural predicates is done by providing supervision on the head of the logical rules expressed as standard logical queries. This means that in our prototype the "shallow" annotations on the structured event will be mapped to queries, while simple events will be mapped to neural predicates.

The DeepProbLog program we defined to address the event recognition task is shown in Figure 4. It consists of the following predicates:

- $digit(X, V, T, V_x)$: a neural predicate that states that the X digit of video V at time T is V_x
- $join\_add\_res(V, V_z)$: a binary predicate that states that video V is a $join\_add$ and the resulting digit of the join is V_z
- $join\_sub\_res(V, V_z)$: a binary predicate that states that video V is a $join\_sub$ and the resulting digit of the join is V_z
- $no\_join(V)$: a unary predicate that states that video V is a no_join video

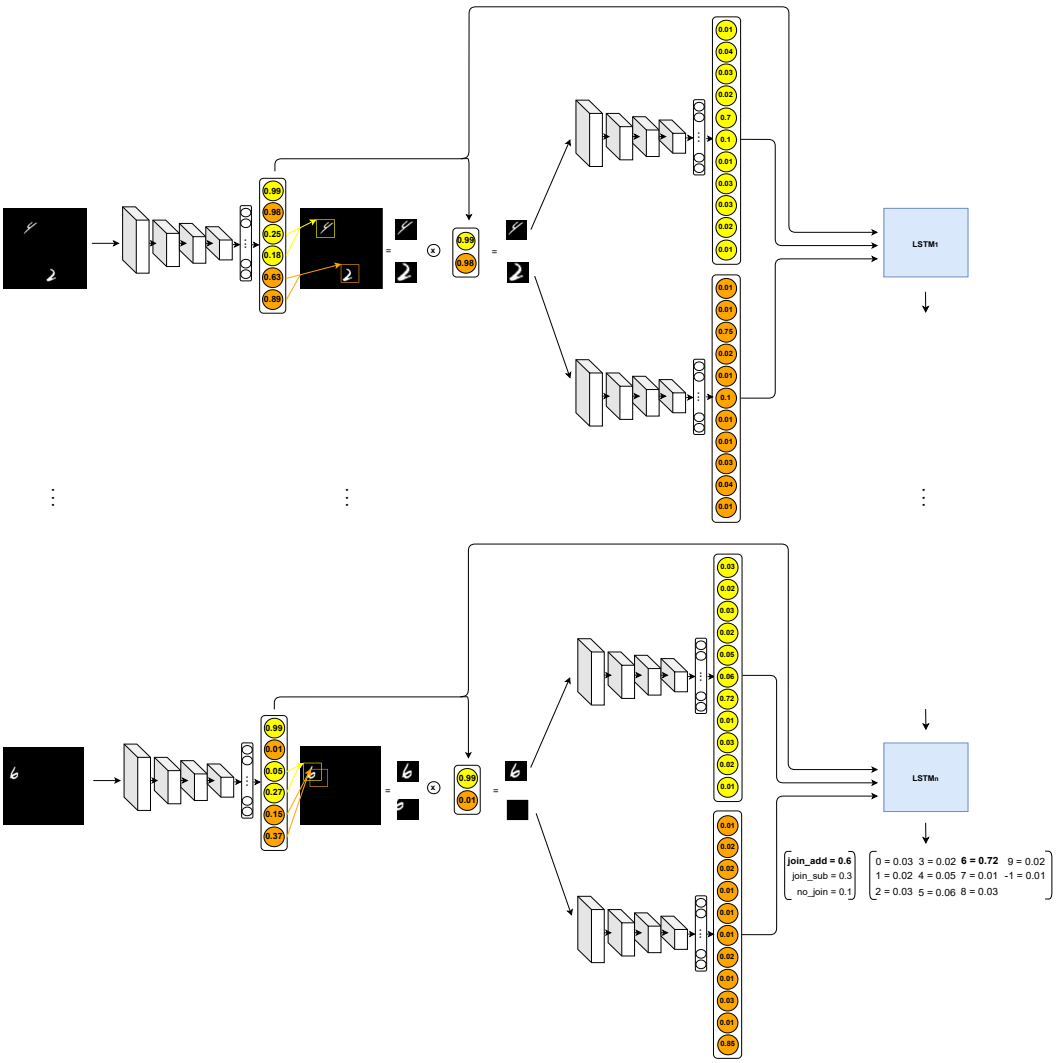


Figure 3 Fully neural approach: LSTM-based architecture.

The neural predicate $digit(X, V, T, V_x)$ is mapped to the combination of digit detector and classifier shown in Figure 3, with the difference that only the output of the classifier (i.e., a probability distribution on the 0-9 digits plus the absence of the digit) is provided. The predicate $join\_add\_res(V, V_z)$ basically represents the combination of $join\_add(X, Y)$, $outcome(join\_add(X, Y), Z)$ and $digit(Z, V_z)$, with the addition of the V variable indicating the video (omitted for simplicity in the formalization throughout the paper). The predicate checks whether there are two digits in the first half of the video and only one digit at the end that is the sum of the two. The $join\_sub\_res$ predicate is similar to $join\_add\_res$ with sum replaced by difference (in absolute value, so that digits do not need to be sorted). Finally, for a no_join, we know that there are two digits for the whole duration of the video. Therefore, we define a rule that only fires when both digits are visible in the last frame.

```
nn(mnist_net, [I, V, T], Y, [0,1,2,3,4,5,6,7,8,9,-1]) :: digit(I, V, T, Y).

join_add_res(V, Z) :-
    between(0, 4, T1),
    digit(0, V, T1, X),
    X > 0, X < 9,
    digit(1, V, T1, Y),
    Y > 0, Y < 10 - X,
    digit(0, V, 9, Z),
    Z is X + Y, Z > 1,
    digit(1, V, 9, -1).

join_sub_res(V, Z) :-
    between(0, 4, T1),
    digit(0, V, T1, X),
    X > 0,
    digit(1, V, T1, Y),
    Y > 0,
    digit(0, V, 9, Z),
    Z is abs(X-Y),
    digit(1, V, 9, -1).

no_join(V) :- digit(1, V, 9, X), X =\= -1.
```

Figure 4 Neuro-symbolic approach: DeepProbLog program.

8 Results

In this section, we present and compare the results of the neural based LSTM approach with our proposed neuro-symbolic approach based on DeepProbLog on the tasks defined in Section 6.

Confusion matrices, where entries (i, j) denote the number of samples of true class i classified as class j, for the event recognition problem and related sub-problems for the two approaches are shown in Figure 5. The first row shows the confusion matrices for the event and outcome recognition (*join_add* with outcome from 1 to 9, *join_sub* with outcome from 0 to 8, *no_join*), while the second row reports the confusion matrices for the underlying task of digit classification (0-9, and -1 corresponding to no digit). The left column reports results for the fully neural approach, the right column those for the neuro-symbolic approach.

Looking at the top left confusion matrix, we can observe that the neural approach is able to recognize the events for which the supervision is provided, even if it sometimes mistakes a *join_add* for a *join_sub* and vice-versa when the outcome is the same. On the other hand, it completely fails in generalizing to unseen events (*join_add* with outcome 8 or 9, *join_sub* with outcome 8). This fact highlights the difficulty of the neural approach in fully learning the semantic behind the join operations. The results of the confusion matrix on digits (bottom left) confirm these findings, as the network fails to classify digits for which no direct supervision is available (i.e., 8 and 9).

The situation with our neuro-symbolic approach is rather different (right column). Indeed, DeepProbLog is capable of predicting the unseen outcomes with reasonable accuracy, and the same holds for the underlying digit classification task. If we compare the confusion matrices on the digits of the two approaches (bottom row), we can observe that our approach has a higher accuracy even on digits for which direct supervision is available. These results clearly indicate the importance of the background knowledge in compensating partial supervision and allowing to generalize beyond what is observed during training.

9 Conclusion and future work

In this work, we have proposed a neuro-symbolic approach for structured event recognition from data sequences, where background knowledge about event structure is combined with deep neural networks used to solve the sub-tasks of event recognition such as object detection

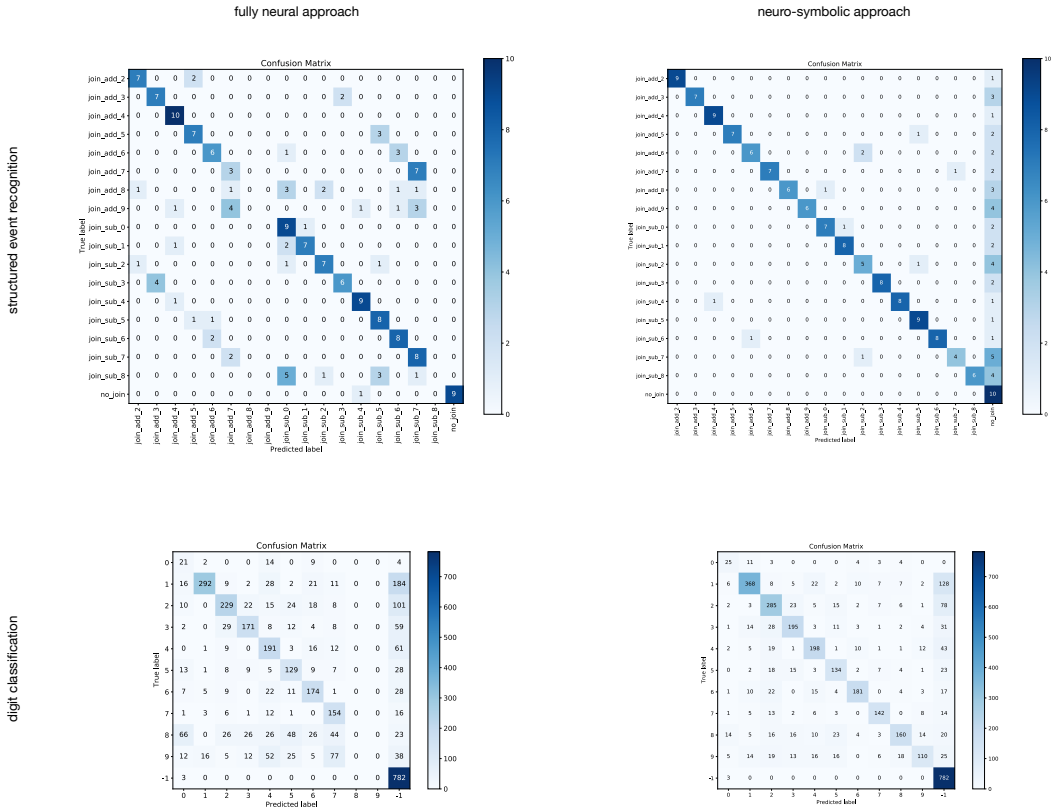


Figure 5 Experimental results: confusion matrices for event + outcome recognition (top row) and digit classification (bottom row). Left: fully neural approach; right: neuro-symbolic approach.

and classification. The proposed architecture can be trained end-to-end with data streams containing only shallow annotations. We prototyped our architecture using DeepProblog as a neuro-symbolic integration framework and tested it on a structured event recognition problem defined on a synthetic dataset automatically generated. The experiments show that the background knowledge about structured events and their outcomes translates supervision on the structured event into supervision on lower-level predictive tasks like object classification, allowing to successfully train the neural components of the architecture. We compare our architecture with a purely neural solution that uses the same basic components for object detection and classification. The comparison shows how the use of background knowledge improves performance for both high-level and low-level prediction tasks. The advantages of these effects are multiple. The first advantage is the fact that we are able to train a classifier without a direct supervision on some of the classes (the classes 8 and 9 in our experiment); a second advantage concerns explanability: while in a fully neural approach it is not possible to explain the happening of an event in terms of its components (object participants, and their types), in our approach the reasoning process that infers the happening of a structured event on the basis of the recognition of some basic facts (detection of an object of a certain type) can be provided as an explanation. As future work, we plan to test the implementation of the proposed architecture on different neuro-symbolic frameworks, and to consider more structured events and also the application of the methodology on real data.

References

1 Kashif Ahmad Ahmad and Nicola Conci. How Deep Features Have Improved Event Recognition in Multimedia: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(2):39:1–39:27, 2019. doi:10.1145/3306240.

2 Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic Complex Event Recognition: A Survey. *ACM Computing Surveys*, 50(5):71:1–71:31, 2017. doi:10.1145/3117809.

3 Alexander Artikis, Evangelos Makris, and Georgios Paliouras. A probabilistic interval-based event calculus for activity recognition. *Annals of Mathematics and Artificial Intelligence*, 89(1-2):29–52, 2021. doi:10.1007/s10472-019-09664-4.

4 Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. Logic-based event recognition. *The Knowledge Engineering Review*, 27(4):469–506, 2012. doi:10.1017/S0269888912000264.

5 Caviar: context aware vision using image-based active recognition, 2011. URL: http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1.

6 Liang-Chieh Chen, Alexander Schwing, Alan Yuille, and Raquel Urtasun. Learning Deep Structured Models. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1785–1794. JMLR.org, 2015. URL: http://proceedings.mlr.press/v37/chenb15.html.

7 Kellie Corona, Katie Osterdahl, Roderic Collins, and Anthony Hoogs. MEVA: A Large-Scale Multiview, Multimodal Video Dataset for Activity Detection. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*, pages 1059–1067. IEEE, 2021. doi:10.1109/WACV48630.2021.00110.

8 Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. doi:10.1162/neco.1997.9.8.1735.

9 Abdullah Khan, Loris Bozzato, Luciano Serafini, and Beatrice Lazzerini. Visual Reasoning on Complex Events in Soccer Videos Using Answer Set Programming. In Diego Calvanese and Luca Iocchi, editors, *GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence, Bozen/Bolzano, Italy, 17-19 September 2019*, volume 65, pages 42–53. EasyChair, 2019. doi:10.29007/pjd4.

10 Abdullah Khan, Luciano Serafini, Loris Bozzato, and Beatrice Lazzerini. Event Detection from Video Using Answer Set Programing. In Alberto Casagrande and Eugenio G. Omodeo, editors, *Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019*, volume 2396 of *CEUR Workshop Proceedings*, pages 48–58. CEUR-WS.org, 2019. URL: http://ceur-ws.org/Vol-2396/paper25.pdf.

11 Paula Lago, Shingo Takeda, Sayeda S Alia, Kohei Adachi, Brahim Bennai, and Sozo Inoue Francois Charpillet. A dataset for complex activity recognition withmicro and macro activities in a cooking scenario. *CoRR*, abs/2006.10681, 2020. arXiv:2006.10681.

12 Weiyao Lin, Huabin Liu, Shizhan Liu, Yuxi Li, Rui Qian, Tao Wang, Ning Xu, Hongkai Xiong, Guo-Jun Qi, and Nicu Sebe. Human in Events: A Large-Scale Benchmark for Human-centric Video Analysis in Complex Events, 2020. arXiv:2005.04490.

13 Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc D. Raedt. DeepProbLog: Neural Probabilistic Logic Programming. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, volume 31, pages 3753–3763, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html.

14 Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. LYRICS: A General Interface Layer to Integrate Logic Inference and deep Learning. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*, volume 11907 of *Lecture Notes in Computer Science*, pages 283–298, 2019. doi:10.1007/978-3-030-46147-8_17.

15 Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. Integrating Learning and Reasoning with Deep Logic Models. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*, volume 11907 of *Lecture Notes in Computer Science*, pages 517–532. Springer, 2020. doi:10.1007/978-3-030-46147-8_31.

16 Juan C. Niebles, Chih-Wei Chen, and Li Fei-Fei. Modeling Temporal Structure of Decomposable Motion Segments for Activity Classification. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision - ECCV 2010, 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part II*, volume 6312 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2010. doi:10.1007/978-3-642-15552-9_29.

17 Luc D. Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2462–2467, 2007. URL: http://ijcai.org/Proceedings/07/Papers/396.pdf.

18 Fabrício Henrique Rodrigues and Mara Abel. What to consider about events: A survey on the ontology of occurrents. *Applied Ontology*, 14(4):343–378, 2019. doi:10.3233/AO-190217.

19 Luciano Serafini and Artur d'Avila Garcez. Learning and Reasoning with Logic Tensor Networks. In Giovanni Adorni, Stefano Cagnoni, Marco Gori, and Marco Maratea, editors, *AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings*, volume 10037 of *Lecture Notes in Computer Science*, pages 334–348, 2016. doi:10.1007/978-3-319-49130-1_25.

20 Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, and Georgios Paliouras. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, 15(2):213–245, 2015. doi:10.1017/S1471068413000690.

21 Khurram Soomro, Amir R. Zamir, and Mubarak Shah. UCF101: A dataset of 101 Human Actions Classes From Videos in The Wild. *CoRR*, abs/1212.0402, 2012. arXiv:1212.0402.

22 Wei Xiang and Band Wan. A Survey of Event Extraction From Text. *IEEE Access*, 7:173111–173137, 2019. doi:10.1109/ACCESS.2019.2956831.

23 Tianwei Xing, Marc R. Vilamala, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. DeepCEP: Deep Complex Event Processing Using Distributed Multimodal Information. In *IEEE International Conference on Smart Computing, SMARTCOMP 2019, Washington, DC, USA, June 12-15, 2019*, pages 87–92. IEEE, 2019. doi:10.1109/SMARTCOMP.2019.00034.

24 Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing Neural Networks into Answer Set Programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1755–1762. ijcai.org, 2020. doi:10.24963/ijcai.2020/243.

Model Checking Timed Recursive CTL

Florian Bruse
School of Electrical Engineering and Computer Science, University of Kassel, Germany

Martin Lange
School of Electrical Engineering and Computer Science, University of Kassel, Germany

Abstract

We introduce Timed Recursive CTL, a merger of two extensions of the well-known branching-time logic CTL: Timed CTL is interpreted over real-time systems like timed automata; Recursive CTL introduces a powerful recursion operator which takes the expressiveness of this logic CTL well beyond that of regular properties. The result is an expressive logic for real-time properties. We show that its model checking problem is decidable over timed automata, namely 2-EXPTIME-complete.

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

Keywords and phrases formal specification, temporal logic, real-time systems

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.12

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Temporal logics are widely used as formal languages for the specification of properties of reactive systems. The most widely known such logics are LTL [19] and CTL [12], having achieved this status partially due to their simplicity as extensions of propositional logic by a small set of intuitive temporal operators. This simplicity in syntax is also reflected by relatively low expressive power; both do not even reach up to full regularity in the sense that they are not equi-expressive to finite-state word, resp. tree automata.

Regular expressive power is a cornerstone in the study of the theory of temporal specification languages, as logics not exceeding this expressivity limit typically possess appealing properties like decidability of their model and satisfiability checking problems.

On the other hand, there are also interesting program properties which are not regular and can therefore not be expressed in such logics, like the absence of buffer over-/underflows, assume-guarantee properties, etc. The literature contains several non-regular extensions of temporal logics or related modal fixpoint logics, e.g. PDL[CFL] [13], FLC [18] and HFL [21]. These have certain features in common: a syntax that makes it difficult to understand the meaning of formulas, and – despite undecidability of their satisfiability problems – a decidable model-checking problem over finite structures [15, 16, 5]. The upshot to take from this is that model checking need not become undecidable when going beyond regular expressiveness.

In order to overcome issues with unintuitive syntaxes in expressive temporal logics, we recently proposed Recursive CTL (RecCTL) [10], an extension of the basic branching-time temporal logic CTL with a single recursion operator which takes formulas as arguments that can be manipulated using other operators and passed into a recursive call. This achieves expressive power, capturing all regular branching-time properties and many non-regular ones. At the same time, model checking is decidable albeit exponentially worse: it is EXPTIME-complete compared to P-completeness for CTL.

Another way of extending the expressive power of temporal logics, which has been followed in the literature for quite some time, is more semantic in nature: the labelled transition systems that logics like CTL are interpreted over model the evolution of time very abstractly by discrete steps taken when passing from one state to another. Hence, the only real timing properties expressible in such logics are unitless and non-quantitative like "*at some point in the future*" etc. This is not sufficient for the modelling of embedded or real-time systems where concrete timing constraints play a role in correctness properties, for instance as in "*within 5 milliseconds of receiving a signal, a control command is issued.*"

In order to capture such effects, transition systems have been extended to model the flow of time more realistically with non-negative, real-numbered delays between time point. Timed Automata [3] are a popular model for the finite representation of such systems. Their great expressiveness compared to ordinary discrete systems shows by the fact that the basis for temporal logics, the simple reachability problem, is already PSPACE-complete.

One of the most popular temporal logics for expressing more complex reachability properties of Timed Automata is Timed CTL [2], an extension of CTL that is capable of making simple assertions about the amount of time that passes before certain events occur on some, resp. all paths. Its model checking problem over Timed Automata is not more difficult than simple reachability: it is also PSPACE-complete, cf. [17].

Here we introduce and study Timed Recursive CTL (TRCTL), a logic that arises from combining the extensions to real-time on one hand, and to non-regular properties on the other. We show that TRCTL retains decidability of model checking over timed automata, but the combination increases the complexity to 2-EXPTIME-completeness.

The paper is organised as follows. In Sect. 2 we recall necessary preliminaries about Timed Automata and TCTL, about RecCTL, and characterise doubly exponential time complexity. In Sect. 3 we introduce TRCTL formally. In Sect. 4 we establish 2-EXPTIME-completeness of its model checking problem. The upper bound is obtained by an exponential reduction to the RecCTL model checking problem, making use of the known region graph abstraction. The lower bound requires a fair amount of encoding large numbers as propositions interpreted over timed automata. TRCTL is then capable of mimicking the aforementioned characterisation of doubly exponential time. We conclude in Sect. 5 with remarks on further work.

2 Preliminaries

2.1 Doubly Exponential Time Complexity

The main result of this paper is 2-EXPTIME-completeness of an expressive extension of Timed CTL interpreted over Timed Automata. For the lower bound we reduce from a problem that is essentially a reformulation of the problem to decide whether a deterministic Turing Machine (DTM) accepts the empty word in 2^{2^n} steps, given some $n \geq 0$.

Suppose Q is the set of states and Γ the tape alphabet of a DTM $\mathcal{M}$, containing a special $\square$ symbol and not containing $\#$. Let $\hat{\Gamma} := \Gamma \cup (Q \times \Gamma) \cup \{\#\}$. Let $f : \mathbb{N} \to \mathbb{N}$. The unique $f(n)$-time-bounded computation of $\mathcal{M}$ on the empty input can be represented by a square, containing $f(n)$ rows, representing time, each of which contains $f(n)$ symbols from $\hat{\Gamma}$, representing a configuration, or space. Each row is of the form $\#w\#$ for some $w \in (\hat{\Gamma} \setminus \{\#\})^{f(n)-2}$, and, if q_0, q_{acc} are $\mathcal{M}$'s starting and accepting states, the bottom row is $\#(q_0, \square)\square^{f(n)-3}\#$, and the top row is of the form $\#(q_{\mathsf{acc}}, \square)w\#$ for some $w \in \hat{\Gamma}^{f(n)-3}$.

Now suppose that δ is $\mathcal{M}$'s transition function. This gives rise to a relation $\hat{\delta} \subseteq \hat{\Gamma}$ such that $(y_1, y_2, y_3, x) \in \hat{\delta}$ iff whenever y_1, y_2, y_3 are consecutive symbols in row t at positions $s-1, s, s+1$, then x is the symbol at position s in row $t+1$.

An $f(n)$-*certificate* (for $\mathcal{M}$ and given n) is a set of mutually recursive predicates $Cert_a : [f(n)] \times [f(n)] \to \{\top, \bot\}$, one for each $a \in \hat{\Gamma}$ with the following properties. Intuitively, $Cert_a(t,s) = \top$ iff the s-th symbol in the t-the configuration of the unique computation of $\mathcal{M}$ on the empty input is a. Clearly, $t, s \leq f(n)$. Formally,

- $Cert_{(q_{\text{acc}},\square)}(f(n)-1, 0) = \top$,
- for all $t \in \{1, \ldots, f(n)-1\}$, $s \in \{1, \ldots, f(n)-2\}$ and $a \in \hat{\Gamma} \setminus \{\#\}$ with $Cert_a(t,s)$ there are $b_1, b_2, b_3 \in \hat{\Gamma}$ with $(b_1, b_2, b_3, a) \in \hat{\delta}$ and

$$Cert_{y_1}(t-1, s-1) \wedge Cert_{y_2}(t-1, s) \wedge Cert_{y_3}(t-1, s+1) \ ,$$

- for all $t \in \{0, \ldots, f(n)-1\}$, $s \in \{0, f(n)-1\}$ we have $Cert_a(t,s)$ iff $a = \#$,
- $Cert_a(0,1)$ iff $a = (q_0, \square)$, and for all $s = 2, \ldots, f(n)-2$: $Cert_a(0,s)$ iff $a = \square$.

Note that the last two clauses determine the values of a in $Cert_a(t,s)$ uniquely for the left, lower und right edge of the square defined by the coordinates t, s, and determinism of the TM $\mathcal{A}$ then determines the values at the inner coordinates uniquely as well.

This characterisation of acceptance in deterministic time-bounded Turing Machines is taken from [11] and can also be used to establish a generic 2-EXPTIME-hardness result.

▶ **Proposition 1.** *It is* 2-EXPTIME*-hard to decide, given a DTM $\mathcal{M}$ and an $n \in \mathbb{N}$ encoded unarily, whether or not there is a 2^{2^n}-certificate for $\mathcal{M}$ and n in the sense above.*

2.2 Models of Real-Time Systems

Timed Transition Systems. A *timed labelled transition system* (TLTS) over a finite set *Prop* of atomic propositions is a $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$ such that

- $\mathcal{S}$ is a set of *states* containing a designated starting state s_0,
- $\to \subseteq \mathcal{S} \times \mathcal{S} \cup \mathcal{S} \times \mathbb{R}^{\geq 0} \times \mathcal{S}$ is the transition relation, consisting of two kinds:
  - *discrete transitions* of the form $s \to t$ for $s, t \in \mathcal{S}$, and
  - *delay transitions* of the form $s \xrightarrow{d} t$ for $s, t \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$, satisfying $s \xrightarrow{0} t$ iff $s = t$ for any $s, t \in \mathcal{S}$, and

$$\forall d, d_1, d_2 \in \mathbb{R}^{\geq 0}, \forall s, t \in \mathcal{S} : d = d_1 + d_2 \text{ and } s \xrightarrow{d} t \Leftrightarrow \exists u \in \mathcal{S} \text{ s.t. } s \xrightarrow{d_1} u \text{ and } u \xrightarrow{d_2} t \ ,$$

- $\lambda : \mathcal{S} \to 2^{Prop}$ labels the states with the set of atomic propositions that hold true in it.

Here we consider TLTS over a singleton set of discrete actions. This is purely done since the temporal logics based on CTL here do not consider different actions. It would be possible to extend the entire theory to TLTS over several discrete transition relations $\xrightarrow{a}, \xrightarrow{b}, \ldots$, and make the logics aware of these. The *extended transition relations* $\overset{d}{\Longrightarrow}$, $d \in \mathbb{R}^{\geq 0}$, are obtained by padding discrete transitions with delays:

$$s \overset{d}{\Longrightarrow} t \quad \text{iff} \quad \exists d_1, d_2 \in \mathbb{R}^{\geq 0}, s', t' \in \mathcal{S} \text{ s.t. } s \xrightarrow{d_1} s', s' \to t', t' \xrightarrow{d_2} t \text{ and } d = d_1 + d_2$$

A *trace* is a sequence $\pi = s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \ldots$

An (untimed) labeled transition system (LTS) is a TLTS over an empty delay transition relation. It is *finite* if the set of its states is finite.

Clock Constraints. Let $\mathcal{X} = \{\mathtt{x}, \mathtt{y}, \ldots\}$ be a set of $\mathbb{R}^{\geq 0}$-valued variables called *clocks*. By $CC(\mathcal{X})$ we denote the set of *clock constraints* over $\mathcal{X}$ which are conjunctive formulas of the form $\top$ or $\mathtt{x} \oplus c$ for $\mathtt{x} \in \mathcal{X}$, $c \in \mathbb{N}$ and $\oplus \in \{\leq, <, \geq, >, =\}$.

A *clock evaluation* is an $\eta : \mathcal{X} \to \mathbb{R}^{\geq 0}$. A clock constraint φ is interpreted in a clock evaluation η in the obvious way:

- $\eta \models \top$ holds for any η,
- $\eta \models \varphi_1 \wedge \varphi_2$ if $\eta \models \varphi_1$ and $\eta \models \varphi_2$,
- $\eta \models \mathtt{x} \oplus c$ if $\eta(\mathtt{x}) \oplus q$ for $\oplus \in \{\leq, <, \geq, >, =\}$.

Given a clock evaluation η, $d \in \mathbb{R}^{\geq 0}$ and a set $R \subseteq \mathcal{X}$, we write $\eta{+}d$ for the clock evaluation that is defined by $(\eta{+}d)(\mathtt{x}) = \eta(\mathtt{x}) + d$ for any $\mathtt{x} \in \mathcal{X}$, and $\eta|_R$ for the clock evaluation that is defined by $\eta|_R(\mathtt{x}) = 0$ for $\mathtt{x} \in R$ and $\eta|_R(\mathtt{x}) = \eta(\mathtt{x})$ otherwise.

Timed Automata. As with TLTS, here we consider timed automata whose transitions are always taken with a single action which is consequently not named. As above, the reason for considering this simplified model is purely the fact that CTL-based logics as defined – the main object of study in this paper – are oblivious of differences in actions anyway.

A *timed automaton* (TA) over *Prop* is a $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$ where

- L is a finite set of so-called *locations* containing a designated *initial* location $\ell_0 \in L$,
- $\mathcal{X}$ is a finite set of clocks,
- $\iota : L \to CC(\mathcal{X})$ assigns a clock constraint, called *invariant*, to each location,
- $\delta \subseteq L \times CC(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is a finite set of transitions. We write $\ell \xrightarrow{g,R} \ell'$ instead of $(\ell, g, R, \ell') \in \delta$. In such a transition, g is called the *guard*, and $R \subseteq \mathcal{X}$ are the *reset* clocks of this transition.

The *index* of the TA $\mathcal{A}$ is the largest constant occurring in its invariants or guards, denoted $m(\mathcal{A})$. The *size* of $\mathcal{A}$ is

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

Note that the size is only logarithmic in the value of constants used in clock constraints as they can be represented in binary notation for instance.

TA are models of state-based real-time systems. The semantics, resp. behaviour of a TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$ is given by an TLTS $\mathcal{T}_\mathcal{A}$ over the time domain $\mathbb{R}^{\geq 0}$ as follows.

- The state set is $\mathcal{S} = \{(\ell, \eta) \mid \ell \in L, \eta \in (\mathcal{X} \to \mathbb{R}^{\geq 0}) \text{ such that } \eta \models \iota(\ell)\}$ consisting of pairs of locations and clock evaluations that satisfy the locations's invariant.
- The initial state is $s_0 = (\ell_0, \eta_0)$ where $\eta_0(\mathtt{x}) = 0$ for all $\mathtt{x} \in \mathcal{X}$.
- Delay transitions retain the underlying location and (possibly) advance the value of clocks in a state: for any $(\ell, \eta) \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$ we have $(\ell, \eta) \xrightarrow{d} (\ell, \eta{+}d)$ if $\eta{+}d \models \iota(\ell)$.
- Discrete transitions possibly change the location and reset clocks: for any $(\ell, \eta) \in \mathcal{S}$, $\ell' \in L$ and $R \subseteq \mathcal{X}$ we have $(\ell, \eta) \to (\ell', \eta|_R)$ if there is $g \in CC(\mathcal{X})$ such that $(\ell, g, R, \ell') \in \delta$ and $\eta|_R \models \iota(\ell')$.
- The propositional label of a state is inherited from the propositional label of the underlying location: $\lambda(\ell, \eta) = \lambda(\ell)$.

In other words, a TA finitely represents a TLTS. Clearly, not every TLTS is finitely representable, so only a subset is captured by TA.

For a detailed introduction into timed automata we refer to the literature [3, 6]. Here we give an example which will be used later on in the lower bound proof in Sect. 4. The TA here act as *gadgets* which means that they have defined locations by which they can be connected to form larger TA. This may entail putting guards or resets onto transitions which do not connect locations in this gadget, as they will be connected later on. The example TA in Fig. 1 are used to encode a counter of some width.

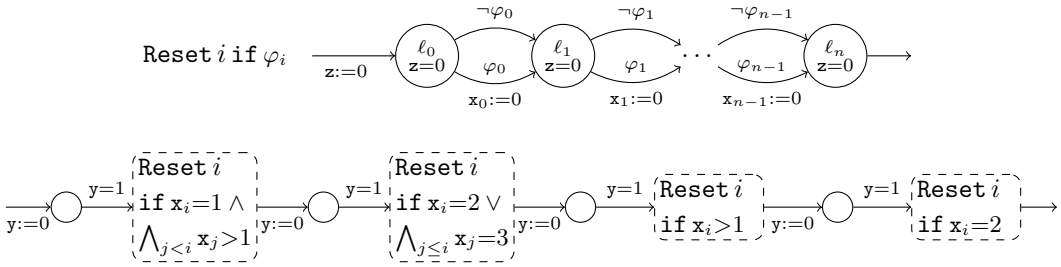


Figure 1 Examples of timed automata: $\texttt{Reset}\, i\, \texttt{if}\, \varphi_i$ for arbitrary clock constraints $\varphi_0, \ldots, \varphi_{n-1}$ (upper TA), and $\texttt{Incr}_{\bar{\mathtt{x}}}$ (lower TA).

▶ **Definition 2.** *Let* $\bar{\mathtt{x}} = (\mathtt{x}_0, \ldots, \mathtt{x}_{n-1})$. *An environment* η *is called a* small $\bar{\mathtt{x}}$*-counter if* $\eta(\mathtt{x}_i) \in \{0, 1\}$ *for all* $i = 0, \ldots, n-1$. *Its* value *is* $\langle \eta_{\bar{\mathtt{x}}} \rangle = \sum_{i=0}^{n-1} \eta(\mathtt{x}_i) \cdot 2^i$.

We drop the annotation by $\bar{\mathtt{x}}$ if it is clear from context. Now consider the TA $\texttt{Reset}\, i\, \texttt{if}\, \varphi$ (read "for $i = 0, \ldots, n-1$ reset the i-th clock if φ_i holds") and $\texttt{Incr}_{\bar{\mathtt{x}}}$ in Fig. 1.

▶ **Observation 3.**

- *Suppose* $\bar{\varphi} = (\varphi_0, \ldots, \varphi_{n-1})$ *is a tuple of clock constraints over the clocks in* $\bar{\mathtt{x}}$. *Then, in the TLTS associated to the TA* $\texttt{Reset}\, i\, \texttt{if}\, \varphi$, *there is a path from* (ℓ_0, η) *to* (ℓ_n, η') *iff* $\eta' = \eta|_{\{\mathtt{x}_i \mid \varphi_i\}}$.
- *Suppose* η *is small counter encoding the value* $m \in [2^n]$ *over* n *clocks in* $\bar{\mathtt{x}}$. *Then, if entering the sub-TLTS generated by the gadget* $\texttt{Incr}_{\bar{\mathtt{x}}}$ *from some state* (ℓ, η), *this sub-TLTS is left towards some* (ℓ', η') *such that* η' *encodes* $m+1$ *modulo* 2^n.

Clearly, there is a gadget similar to $\texttt{Incr}_{\bar{\mathtt{x}}}$ that decreases the value encoded in these clocks. We denote it by $\texttt{Decr}_{\bar{\mathtt{x}}}$. Note that technically, $\texttt{Incr}_{\bar{\mathtt{x}}}$ is not a TA since the conditions on the resets, resp. their negations, which are used as guards in the reset gadget, contain disjunctions. However, in this case, the guards can be brought into disjunctive normal form at a minimal blowup, whence a disjunction $\psi_1 \vee \ldots \vee \psi_k$ in a supposed guard can be replaced by k separate transitions.

The Region Abstraction. There is a well-known abstraction of an TLTS arising from a TA $\mathcal{A}$ into a finite LTS known as the *region graph* $\mathcal{R}_{\mathcal{A}}$, used in decidability proofs for decision problems on TA.

In the following we only consider TLTS $\mathcal{T}_{\mathcal{A}}$ that arise from some TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$. The region abstraction is a mapping of such $\mathbb{R}^{\geq 0}$-TLTS into finite LTS. It is based on an equivalence relation $\simeq_m$, for $m \in \mathbb{N}$, on clock evaluations defined as follows.

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

Here, $\mathit{frac}(r)$ denotes the fractional part of a real number. It is easy to see that $\simeq_m$ is indeed an equivalence relation for any m. It is lifted to states of the TLTS $\mathcal{T}_{\mathcal{A}}$ in the most straight-forward way:

$$(\ell, \eta) \simeq_m (\ell', \eta') \quad \text{iff} \quad \ell = \ell' \text{ and } \eta \simeq_m \eta' \ .$$

We write $[\eta]_m$ for the equivalence class of η under $\simeq_m$ and likewise for $[(\ell, \eta)]_m$. When m is clear from the context we may also drop it and simply write $[\eta]$, resp. $[(\ell, \eta)]$.

This is not only an equivalence relation on the state space of $\mathcal{T}_\mathcal{A}$ but in fact even a congruence w.r.t. the labelling and discrete and delay transitions when $m \geq m(\mathcal{A})$. This is what makes it usable in order to abstract the uncountable state space of $\mathcal{T}_\mathcal{A}$ into a finite discrete state space as follows.

The *region graph* $\mathcal{R}_\mathcal{A}$ of the TA $\mathcal{A}$ is the LTS obtained as the quotient of $\mathcal{T}_\mathcal{A}$ under the congruence relation $\simeq_m$ (with $m = m(\mathcal{A})$), together with an additional collapse of delay transitions for different delays into a single "*some-delay*" value τ. Its components are as follows.

- The state space is $\{(\ell, [\eta]) \mid \ell \in L, \eta \in (\mathcal{X} \to \mathbb{R}^{\geq 0}), \eta \models \iota(\ell)\}$. The initial state is $(\ell_0, [\eta_0])$.
- Discrete transitions from one state to another are obtained by possibly delaying, then performing a discrete transition in the timed space and then possibly delaying again afterwards. We have

$$(\ell, [\eta]) \to (\ell', [\eta']) \quad \text{if there are } d, d' \in \mathbb{R}^{\geq 0}, \hat{\eta}, \hat{\eta}' \text{ s.t. } (\ell, \eta) \xrightarrow{d_1} (\ell, \hat{\eta}) \to (\ell', \hat{\eta}') \xrightarrow{d_2} (\ell', \eta')$$

  for any $\ell, \ell' \in L$, $\eta, \eta' \in \mathcal{X} \to \mathbb{R}^{\geq 0}$.
- The propositional labelling is given as $\lambda(\ell, [\eta]) = \lambda(\ell, \eta) = \lambda(\ell)$.

We obtain the following proposition:

▶ **Proposition 4** ([3]). *Let $\mathcal{A}$ be a TA over n clocks with ℓ locations and of index m. Then $\mathcal{R}_\mathcal{A}$ is an (untimed) LTS of size $\ell \cdot 2^{\mathcal{O}(n(\log n + \log m))}$, i.e. exponential in $|\mathcal{A}|$, and there is a trace $s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \dots$ in $\mathcal{T}_\mathcal{A}$ iff there is a path $[s_0] \to [s_1] \to \dots$ in $\mathcal{R}_\mathcal{A}$.*

2.3 Temporal Logics

We recall the two most relevant temporal logics which form the basis for the definition of Timed Recursive CTL in Sect. 3: Timed CTL, the extension of pure CTL by operators to quantitatively speak about the passage of time, and Recursive CTL, the extension of CTL by a recursion operator which gives it much greater expressive power.

Timed Computation Tree Logic. As before, let *Prop* be a set of atomic propositions. Formulas of Timed CTL (TCTL) are given by the following grammar.

$$\varphi ::= q \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathtt{E}(\varphi\, \mathtt{U}^J\, \varphi) \mid \mathtt{A}(\varphi\, \mathtt{U}^J\, \varphi)$$

where $q \in \mathit{Prop}$ and J denotes a natural-number bounded interval in $\mathbb{R}^{\geq 0}$, i.e. it takes one of the forms $[n, m], (n, m], [n, m), (n, m), [n, \infty), (n, \infty)$ with $n, m \in \mathbb{N}$, $n \leq m$.

Other Boolean connectives are defined as abbreviations in the usual way: $\mathtt{tt} := q \vee \neg q$ for some $q \in \mathit{Prop}$, $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \to \psi := \neg\varphi \vee \psi$, etc. Likewise, other familiar temporal operators can be obtained as abbreviations as well: $Q\mathtt{F}^J\varphi := Q(\mathtt{tt}\, \mathtt{U}^J\, \varphi)$ for $Q \in \{\mathtt{E}, \mathtt{A}\}$, $Q\mathtt{G}^J\varphi := \neg\overline{Q}\mathtt{F}^J\neg\varphi$ where $\overline{\mathtt{E}} = \mathtt{A}$ and $\overline{\mathtt{A}} = \mathtt{E}$. We also use an intuitive way of the form $\oplus n$ with $\oplus \in \{\leq, <, \geq, >, =\}$ for denoting intervals when possible, for instance $\mathtt{EF}^{>2}q$ stands for $\mathtt{EF}^{(2,\infty)}q$, and $\mathtt{AG}^{\leq 5}q$ stands for $\mathtt{AG}^{[0,5]}q$.

Formulas of TCTL are interpreted over $\mathbb{R}^{\geq 0}$-timed transition systems $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$: $[\![\varphi]\!]^\mathcal{T}$ denotes the set of states in $\mathcal{T}$ in which φ holds, defined inductively as follows.

$$\begin{aligned}
[\![q]\!]^\mathcal{T} &:= \{s \mid q \in \lambda\} \\
[\![\varphi \wedge \psi]\!]^\mathcal{T} &:= [\![\varphi]\!]^\mathcal{T} \cap [\![\psi]\!]^\mathcal{T} \\
[\![\neg\varphi]\!]^\mathcal{T} &:= \mathcal{S} \setminus [\![\varphi]\!]^\mathcal{T} \\
[\![\mathtt{E}(\varphi\, \mathtt{U}^J\, \psi)]\!]^\mathcal{T} &:= \{s \in \mathcal{S} \mid \text{there is a trace } \pi = s, \dots \text{ s.t. } \pi \models \varphi\, \mathtt{U}^J\, \psi\} \\
[\![\mathtt{A}(\varphi\, \mathtt{U}^J\, \psi)]\!]^\mathcal{T} &:= \{s \in \mathcal{S} \mid \text{for all traces } \pi = s, \dots \text{ we have } \pi \models \varphi\, \mathtt{U}^J\, \psi\}
\end{aligned}$$

and for a non-zeno trace $\pi = s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} s_2 \xrightarrow{d_2} \dots$ we have $\pi \models \varphi \,\mathtt{U}^J\, \psi$ iff

$$\exists i \geq 0, \exists d \in [0, d_i], \exists s' \text{ s.t. } s_i \xrightarrow{d} s' \text{ and } (\sum_{h=0}^{i} d_i) + d \in J \text{ and } \mathcal{T}, s' \models \psi \text{ and}$$
$$\forall j < i, \forall d' \in [0, d_j], \forall s' \text{ s.t. } s_j \xrightarrow{d'} s' \text{ we have } \mathcal{T}, s' \models \varphi \vee \psi \text{ and}$$
$$\forall d' \in [0, d), \forall s' \text{ s.t. } s_i \xrightarrow{d'} s' \text{ we have } \mathcal{T}, s' \models \varphi \vee \psi.$$

We write $\mathcal{T}, s \models \varphi$ if $s \in [\![\varphi]\!]^{\mathcal{T}}$ for arbitrary $s \in \mathcal{S}$, and also $\mathcal{T} \models \varphi$ if $\mathcal{T}, s_0 \models \varphi$.

The *model checking problem* for TCTL is the following: given a TA $\mathcal{A}$ and a TCTL formula φ, decide whether or not $\mathcal{R}_{\mathcal{A}} \models \varphi$.

▶ **Proposition 5** ([1, 17]). *The model checking problem for* TCTL *is* PSPACE*-complete, even for TA over a single clock.*

Temporal Logic with Recursion. We briefly present Recursive CTL (RecCTL), the other building block besides TCTL that make up Timed Recursive CTL, to be defined in the following section.

Let *Prop* be a set of atomic propositions. Formulas of RecCTL are obtained by addition of the recursion operator to the (purely modal part of) CTL. Let $\mathcal{V}_1 = \{x, y, \dots\}$ be a set of propositional variables and $\mathcal{V}_2 = \{\mathcal{F}, \dots\}$ be a set of so-called *recursion* variables. Formulas of RecCTL are given by the following grammar.

$$\varphi ::= q \mid x \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathtt{EX}\varphi \mid \Phi(\varphi, \dots, \varphi) \qquad \Phi ::= \mathcal{F} \mid \mathtt{rec}\, \mathcal{F}(x_1, \dots, x_k; y_1, \dots, y_h).\, \varphi$$

where $x, x_i, y_i \in \mathcal{V}_1$, $\mathcal{F} \in \mathcal{V}_2$.

A formula derived from φ in this grammar is called *propositional*, those derived from Φ are called *first-order*. Formulas are interpreted over (untimed) LTS $\mathcal{T}$ over some state set $\mathcal{S}$. A propositional formula φ denotes a *predicate* $[\![\varphi]\!]^{\mathcal{T}} \in 2^{\mathcal{S}}$, i.e. a set of states just like any CTL formula does; a first-order formula however denotes a *predicate transformer* $[\![\Phi]\!]^{\mathcal{T}} : 2^{\mathcal{S}} \times \dots \times 2^{\mathcal{S}} \to 2^{\mathcal{S}}$.

We do not give the details of the formal semantics here. It suffices to note that the recursion operator is interpreted as the least fixpoint in the corresponding complete lattice of first-order functions called predicate transformers. For this to work seamlessly, i.e. these fixpoints to exist, we need to guarantee that any variable $\mathcal{F}$ is used monotonically in φ inside of $\mathtt{rec}\, \mathcal{F}(\vec{x}; \vec{y}).\, \varphi$ only. The fact that the logic features negation ($\neg$) and application ($\Phi(\varphi_1, \dots, \varphi_k)$) requires a slightly more involved syntactic criterion for monotonicity. In particular, in order to know whether some variable is used monotonically, it may be required to know this for others as well. This is why the formal parameters $x_1, \dots, x_k; y_1, \dots, y_h$ ($k, h \geq 0$) to a recursion operator are separated into two parts: those left of the divider ";" are used monotonically, those to the right are used antitonically. RecCTL employs a small type system to ensure these properties. For details we refer to the literature [10] or the next section where the machinery is carried out for full Timed Recursive CTL anyway.

An important result on RecCTL to notice here, as it will be used later on in Sect. 4, is decidability of its model checking problem.

▶ **Proposition 6** ([10]). *The model checking problem for* RecCTL *over finite LTS is* EXPTIME*-complete.*

3 Timed Recursive Computation Tree Logic

The formal syntax. Let *Prop* be a set of atomic propositions. The syntax of Timed Recursive CTL (TRCTL) is similar to that of RecCTL in that we distinguish between propositional and first-order formulas. We also need two kinds of variables again: first-order variables $\mathcal{V}_2 = \{\mathcal{F}, \mathcal{G}, \ldots\}$ to form recursion anchors and propositional variables $\mathcal{V}_1 = \{x, y, \ldots\}$ for formal parameters of recursive formulas. Formulas are then given by

$$\varphi ::= p \mid x \mid \varphi \wedge \psi \mid \neg\varphi \mid \mathtt{E}(\varphi\, \mathtt{U}^J\, \varphi) \mid \Phi(\varphi, \ldots, \varphi) \qquad \Phi ::= \mathcal{F} \mid \mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_k).\, \varphi$$

where $p \in \mathit{Prop}$, $k \geq 0$, $x, x_1, \ldots, x_k \in \mathcal{V}_1$, $\mathcal{F} \in \mathcal{V}_2$ and J denotes an interval in $\mathbb{R}^{\geq 0}$ with integer bounds as in the syntax for TCTL. We write $m(\varphi)$ to denote the largest constant that occurs in interval annotations of the Until operators in φ.

Note that CTL features the *Next* operators $Q\mathtt{X}$ as well as the *Until* operators $Q\mathtt{U}$. The former is missing in TCTL since there is no "next" moment in dense real time. RecCTL, however, seems to feature the *Next* but not the *Until*. This is simply because $Q(\varphi U \psi)$ is expressible via $Q\mathtt{X}$ using the recursion operator which is stronger than propositional fixpoints, i.e. $Q(\varphi \mathtt{U} \psi) \equiv (\mathtt{rec}\, \mathcal{F}().\, \psi \vee (\varphi \wedge Q\mathtt{X}\mathcal{F}()))()$, written more conveniently as $\mathtt{rec}\, \mathcal{F}.\, \psi \vee (\varphi \wedge Q\mathtt{X}\mathcal{F})$, along the lines of the embedding of CTL into the modal μ-calculus. This does not work for the time-bounded *Until* operator anymore. Hence, TRCTL features such the *Until* but not the *Next* operator just like TCTL.

Other Boolean and temporal operators are defined in the usual way, for instance $\mathtt{EF}^J \varphi := \mathtt{E}(\mathtt{tt}\, \mathtt{U}^J\, \varphi)$, $\mathtt{AG}^J \varphi := \neg \mathtt{EF}^J \neg\varphi$, etc. and will be used freely henceforth.

Vectorial form. The semantics of the recursion operator will be explained using least fixpoints in complete function lattices. This makes the Bekič Lemma [7] available which allows formulas with mutual dependencies between recursion variables to be written down in a more readable form. A formula in *vectorial form*, cf. [4] for its use in $\mathcal{L}_\mu$, is a

$$\mathtt{rec}_{\,i} \begin{pmatrix} \mathcal{F}_1(x_1, \ldots, x_k) & . & \varphi_1 \\ & \vdots & \\ \mathcal{F}_n(x_1, \ldots, x_k) & . & \varphi_n \end{pmatrix} (\psi_1, \ldots, \psi_k).$$

Informally, this defines not just one but several functions $\mathcal{F}_1, \ldots, \mathcal{F}_n$ which may all depend on each other in a mutually recursive way formalised in the φ_j's. In the end, the function named by $\mathcal{F}_i$ is applied to the initial arguments $\psi_1, \ldots, \psi_k$.

Well-formed formulas. Not every formula generated by the formal syntax as introduced above is well-formed. For instance, when a recursion formula has k formal parameters as in $\Phi = \mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_k).\, \varphi$, it should only be applied to a tuple of k arguments as in $\Phi(\varphi_1, \ldots, \varphi_k)$. The same goes for any subformula of the form $\mathcal{F}(\psi_1, \ldots, \psi_k)$.

More importantly, a well-defined semantics can only be given to recursive formulas when the recursion variable occurs monotonically in the defining fixpoint formula only. Here we refrain from giving further formalities in terms of a type system that ensures well-formedness. For what follows, it suffices to work with the intuitive notion of "occurring only monotonically". For formal details we refer to [10] where the notion of well-formedness is made precise for RecCTL. The same principles can be applied here to this real-time extension of this logic.

The formal semantics. As with TCTL, (propositional) formulas of TRCTL are interpreted in states of an TLTS $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$. In fact, it suffices to extend the semantics of TCTL to those operators (propositional variables and first-order formulas) which do not already occur in the syntax of TCTL. Due to the presence of variables, we need variable interpretations ϑ in order to explain the meaning of a formula inductively. Such a ϑ maps propositional variables to sets of states, $\vartheta(x) \in 2^{\mathcal{S}}$ for $x \in \mathcal{V}_1$, and first-order variables to functions of corresponding arity over these: $\vartheta(\mathcal{F}) : 2^{\mathcal{S}} \times \ldots \times 2^{\mathcal{S}} \to 2^{\mathcal{S}}$.

These functions form a complete Boolean lattice ordered pointwise, hence least fixpoints of monotone functionals mapping one such function to another exist due to the Knaster-Tarski Theorem [20]. These are used to explain the meaning of the recursion operator. For details, we refer to the exposition on RecCTL [10] or on HFL [21] that this idea goes back to – the only difference is that there, $\mathcal{S}$ is the state space of an untimed LTS rather than a TLTS.

A propositional formula φ gives rise to a set $[\![\varphi]\!]^{\mathcal{T}}_{\vartheta}$ of states that satisfy it under the variable interpretation ϑ, and similarly for first-order formulas and corresponding first-order functions. The semantics is defined as follows. The clauses presented for $\varphi \in$ TCTL apply here as well under the provision that each $[\![\cdot]\!]^{\mathcal{T}}$ is replaced by $[\![\cdot]\!]^{\mathcal{T}}_{\vartheta}$. Additionally,

$$[\![x]\!]^{\mathcal{T}}_{\vartheta} := \vartheta(x) \quad \text{for } x \in \mathcal{V}_1 \ , \qquad [\![\Phi(\varphi_1, \ldots, \varphi_k)]\!]^{\mathcal{T}}_{\vartheta} := [\![\Phi]\!]^{\mathcal{T}}_{\vartheta}([\![\varphi_1]\!]^{\mathcal{T}}_{\vartheta}, \ldots, [\![\varphi_k]\!]^{\mathcal{T}}_{\vartheta})$$

for propositional formulas, while for first-order formulas we set $[\![\mathcal{F}]\!]^{\mathcal{T}}_{\vartheta} := \vartheta(\mathcal{F})$ if $\mathcal{F} \in \mathcal{V}_2$ and

$$[\![\mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_k).\, \varphi]\!]^{\mathcal{T}}_{\vartheta} := \bigsqcap \{ f : (2^{\mathcal{S}})^k \to 2^{\mathcal{S}} \mid \forall S_1, \ldots, S_k : [\![\varphi]\!]^{\mathcal{T}}_{\vartheta[\mathcal{F} \mapsto f, x_1 \mapsto S_1, \ldots, x_k \mapsto S_k]} \subseteq f(S_1, \ldots, S_k) \}$$

where $\sqcap$ denotes the point-wise intersection for functions: $(f \sqcap g)(S) := f(S) \cap g(S)$.

Examples. We illustrate the use of the recursion operator in TRCTL to form structurally complex properties which cannot be expressed in TCTL. We refer to [10] for more exposition regarding RecCTL. It is helpful, though, to imagine the recursive formulas to be unrolled so that new arguments are being built and these to be plugged in for the formal parameters.

▶ **Example 7.** Consider $\varphi_{\mathsf{ag}} := \big(\mathtt{rec}\, \mathcal{F}(x,y).\, (x \wedge \neg y) \vee \mathcal{F}(\mathtt{AF}^{\leq 3} x, \mathtt{AF}^{\leq 2y})\big)(p,p)$. Unrolling of the recursion shows that it is equivalent to

$$\bigvee_{i \geq 0} \underbrace{\mathtt{AF}^{\leq 3}\mathtt{AF}^{\leq 3} \ldots \mathtt{AF}^{\leq 3}}_{i \text{ times}} p \wedge \neg \underbrace{\mathtt{AF}^{\leq 2}\mathtt{AF}^{\leq 2} \ldots \mathtt{AF}^{\leq 2}}_{i \text{ times}} p$$

stating "there is an i such that on all paths we see i occurrences of p in distances of at most 3 seconds, but not in distances of at most 2 seconds." Negating this to $\neg\varphi_{\mathsf{ag}}$ then formalises "whenever it is possible to see p in distances of 3 seconds i times on a path, then it is also possible to do so in distances of 2 seconds on some path." This is inspired by the formalisation of assume-guarantee properties in HFL [21].

▶ **Example 8.** Note that the context-free grammar G with productions

$$F_1 \to F_2 F_3 \ , \quad F_2 \to \mathsf{out} \mid \mathsf{in} F_2 F_2 \ , \quad F_3 \to \varepsilon \mid \mathsf{in} F_3 \mid \mathsf{out} F_3$$

generates the set of all $\mathsf{in}, \mathsf{out}$-sequences such that some prefix contains more out's than in's. It can be seen as the set of all finite computations in which a buffer underflow occurs. Now consider the TRCTL formula

$$\varphi_{\mathsf{buf}} := \mathtt{rec}_{\,1} \begin{pmatrix} \mathcal{F}_1(x) & . & \mathcal{F}_2(\mathcal{F}_3(x)) \\ \mathcal{F}_2(x) & . & \mathtt{E}(p_{\mathsf{out}}\, \mathtt{U}^{\geq 1}\, x) \vee \mathtt{E}(p_{\mathsf{in}}\, \mathtt{U}^{\geq 1}\, \mathcal{F}_2(\mathcal{F}_2(x))) \\ \mathcal{F}_3(x) & . & x \vee \mathtt{E}(p_{\mathsf{in}}\, \mathtt{U}^{\geq 1}\, \mathcal{F}_3(x)) \vee \mathtt{E}(p_{\mathsf{out}}\, \mathtt{U}^{\geq 1}\, \mathcal{F}_3(x)) \end{pmatrix} (\mathtt{tt}) \ .$$

It states that there is a path forming a buffer underflow, provided that consecutive traversal of states satisfying p_{in}, resp. p_{out} for at least 1sec are taken as input/output actions for the buffer. Then $\neg\varphi_{\mathsf{buf}}$ formalises absence of such underflows under this interpretation.

4 The Complexity of Model Checking

In this section we show that the model checking problem for TRCTL is 2-EXPTIME-complete. We begin with the upper bound.

Upper Bound. We follow the same principles as usual decidability proofs for problems on TA, using so-called *untiming* constructions like the one for the region graph. Let $\varphi \in$ TRCTL and $\mathcal{A}$ be a TA not using the clock $\mathtt{z}$. Let $\mathcal{A}^{\mathtt{z}}$ result from it by simply adding the clock $\mathtt{z}$ to it (which is not accessed or manipulated anywhere). Let $\mathcal{R}_{\mathcal{A}^{\mathtt{z}}}$ be the corresponding region graph. Note that its states are of the form $(\ell, [\eta])$, where ℓ is a location of $\mathcal{A}$ and $[\eta]$ is a region, i.e. an equivalence class of a clock evaluation η that is also defined on $\mathtt{z}$ now.

We construct a new LTS $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}}$ by extending $\mathcal{R}_{\mathcal{A}^{\mathtt{z}}} = (\mathcal{S}, \to, s_0, \lambda)$ in the following two ways:

- For each state $(\ell, [\eta])$ and each $c \leq m(\varphi)$, add new proposition $p_{\mathtt{z}\oplus c}$ to $\lambda(\ell, [\eta])$ if $\eta \models \mathtt{z} \oplus c$ for $\oplus \in \{\leq, <, \geq, >, =\}$.
- For each state $(\ell, [\eta])$ introduce a new state $s_{\ell,[\eta]}$ with the sole label $\{r_{\mathtt{z}}\}$, and add transitions $(\ell, [\eta]) \to s_{\ell,[\eta]} \to (\ell, [\eta|_{\{\mathtt{z}\}}])$.

This has introduced new traces in this region graph: at any moment, it is now possible to reset clock $\mathtt{z}$, and then continue some original trace. Moreover, the resetting of $\mathtt{z}$ becomes visible through the traversal of a state that satisfies $r_{\mathtt{z}}$. Since $\mathtt{z}$ is not used in $\mathcal{A}$, this is the only way that it is being reset. Moreover, the values of $\mathtt{z}$ are also accessible through propositions of the form $p_{\mathtt{z}\oplus c}$.

Next we rewrite φ so that it can make use of these propositions. The formula $\varphi^{\mathtt{z}}$ results from φ by replacing each subformula of the form

- $\mathtt{E}(\psi_1\, \mathtt{U}^{[c,d]}\, \psi_2)$ by $\mathtt{EX}(r_{\mathtt{z}} \wedge \mathtt{EXE}((\neg r_{\mathtt{z}} \wedge \psi_1)\, \mathtt{U}\, (\neg r_{\mathtt{z}} \wedge p_{\mathtt{z}\geq c} \wedge p_{\mathtt{z}\leq d} \wedge \psi_2)))$, resp.
- $\mathtt{A}(\psi_1\, \mathtt{U}^{[c,d]}\, \psi_2)$ by $\mathtt{EX}(r_{\mathtt{z}} \wedge \mathtt{EXA}((\neg r_{\mathtt{z}} \to \psi_1)\, \mathtt{U}\, (\neg r_{\mathtt{z}} \to p_{\mathtt{z}\geq c} \wedge p_{\mathtt{z}\leq d} \wedge \psi_2)))$.

For open intervals on one side, the p-propositions are amended accordingly to $p_{\mathtt{z}>c}$ etc.

The following forms the basis of an exponential reduction of TRCTL model checking to RecCTL model checking.

▶ **Lemma 9.** *Let $\mathcal{A}$ be a TA, $\varphi \in$* TRCTL*.*

(a) *$\varphi^{\mathtt{z}}$ is a formula of (untimed)* RecCTL *and is constructible in time $\mathcal{O}(|\varphi|)$.*

(b) *$\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}}$ is an (untimed) LTS of size at most (singly) exponential in $|\mathcal{A}|$ and $m(\varphi)$ and also constructible in such time.*

(c) *$\mathcal{T}_{\mathcal{A}} \models \varphi$ iff $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}} \models \varphi^{\mathtt{z}}$.*

Parts (a) and (b) are easily checked. Part (c) can be proved by simple induction on the structure of φ using Prop. 4.

▶ **Theorem 10.** *The model checking problem for* TRCTL *over TA is decidable in* 2-EXPTIME*.*

Proof. Let a TA $\mathcal{A}$ and a TRCTL formula φ be given. To check whether $\mathcal{T}_{\mathcal{A}} \models \varphi$ holds, first construct $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}}$ and $\varphi^{\mathtt{z}}$. According to Lemma 9, this can be done in exponential time, and it suffices to check whether or not $\mathcal{R}^{\mathtt{z}}_{\mathcal{A}} \models \varphi^{\mathtt{z}}$ holds. According to Prop. 6, the latter can be solved in exponential time. Altogether, this gives a doubly exponential upper bound on the time complexity of model checking TRCTL over TA. ◀

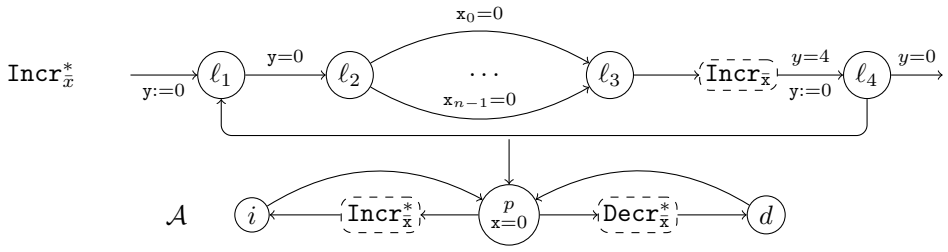


Figure 2 A gadget for arbitrary incrementation (upper part), the TA $\mathcal{A}$ (lower part).

Encoding Large Numbers. We now want to encode numbers in the range $[2^{2^n}]$ in TRCTL. We have seen in Obs. 3 how numbers in the range $[2^n]$ can be encoded in small counters and that there is a polynomially-sized gadget such that passing through (the TLTS generated by) that gadget increases the value encoded in the counter by 1. We now extend this to larger numbers. Clearly, increasing the number of clocks involved is not sufficient unless we use exponentially many clocks. However, note that a TA already generates an exponentially large TLTS through the values of its associated clocks, even after the region graph abstraction. This stems from the fact that both locations and clock values in a TA contribute to the TLTS, and with multiple clocks present, one location may generate many TLTS states. Informally put, the question whether some proposition holds at some location, and for which *small counter values*, already has exponentially many possible answers. We use small counters to represent the bits in binary numbers of exponential width, exactly enough to represent numbers of doubly exponential size. Hence, *large counters* are sets of TLTS states that agree on the location component, say ℓ. What varies are the clock values. Intuitively, we shall consider a bit $b \in [2^n]$ set in the representation of a number through such a set, if said set contains the state (ℓ, η) such that η is a small counter with $\langle \eta \rangle = b$.

▶ **Definition 11.** *Consider the TLTS $\mathcal{T}_\mathcal{A}$ generated by the TA $\mathcal{A}$, depicted in the upper part of Fig. 2. Let $\bar{\mathtt{x}} = (\mathtt{x}_0, \ldots, \mathtt{x}_{n-1})$. A set S of states of the form (p, η), where η is a small $\bar{\mathtt{x}}$-counter, is called a* large $(\bar{\mathtt{x}})$-counter. *Its* value *is $\langle S \rangle = \sum_{i=0}^{2^n-1} b_i \cdot 2^i$ where $b_i = 1$ if the small counter with value i belongs to S, and $b_i = 0$ otherwise.*

For example, the empty set encodes $m = 0$ since $(p, \eta) \in S$ for no η. Any set that contains all states of the form (p, η) encodes $m = 2^{2^n} - 1$, since $(p, \eta) \in S$ for all small counters η. On the other hand, a set that contains only (p, η) such that $\langle \eta \rangle = 0$ encodes $m = 1$, while a set that contains all states but those with $\langle \eta \rangle = 0$ encodes $m = 2^{2^n} - 2$. Note that the first two sets are expressible in TRCTL via $\mathtt{ff}$ and $\mathtt{tt}$. In fact, every TRCTL-formula of the form $p \wedge \psi$ defines a large counter, whence we write $\varphi = \langle m \rangle$ for a formula that defines a counter that encodes m.

Now assume that the gadget $\mathtt{Decr}^*_{\bar{\mathtt{x}}}$ is obtained from $\mathtt{Incr}^*_{\bar{\mathtt{x}}}$, by replacing $\mathtt{Incr}_{\bar{\mathtt{x}}}$ by $\mathtt{Decr}_{\bar{\mathtt{x}}}$ and by replacing the test for 0 by a test for 1. We use the following lemma.

▶ **Lemma 12.** *Consider $\mathcal{T}_\mathcal{A}$ where $\mathcal{A}$ is the TA in the lower part of Fig. 2. Let (p, η) and (p, η') be states in $\mathcal{T}_\mathcal{A}$ such that η, η' are small counters. If (p, η') is reachable from (p, η) by passing exactly once through the sub-TLTS generated by the gadget $\mathtt{Incr}^*_{\bar{\mathtt{x}}}$ and η encodes $m \in 2^n - 1$, then η' encodes a value in in $\{m+1, \ldots, 2^n - 1\}$. Moreover, for each such $m' \in \{m+1, \ldots, 2^n - 1\}$, there is a path through the gadget such that the η' encodes m', and there is no such path if $m = 2^n - 1$. The analogue holds for $\mathtt{Decr}^*_{\bar{\mathtt{x}}}$.*

Proof. In $\texttt{Incr}^*_{\texttt{x}}$, time flows only between ℓ_3 and ℓ_4, namely for exactly 4 units in $\texttt{Incr}_{\bar{\texttt{x}}}$ (cf. Obs. 3). Now assume that the sub-TLTS generated by $\texttt{Incr}^*_{\bar{\texttt{x}}}$ is entered from (p, η). When passing directly from ℓ_3 to ℓ_4, the value encoded in η is increased by 1. Moreover, passing from ℓ_1 to ℓ_3 and, hence to the end of the gadget, is only possible if at least one of the $\texttt{x}_i$ is 0, i.e. if η encodes a number less than $2^n - 1$. Finally, it is not hard to see that the return path from ℓ_4 to ℓ_1 can be taken without making it impossible to leave the gadget as long as at least one of the $\texttt{x}_i$ is 0, whence the gadget can be left with any value in the range $\{m+1, \dots, 2^n - 1\}$ encoded in η. ◀

We use Lem. 12 to manipulate large counters by TRCTL-formulas. Recall that, when incrementing a binary number m given as $b_0 \cdots b_{n-1}$ with least significant bit left, a bit is set in the representation of $m+1$ iff it is already set in (the representation) of m, and there is a bit of lesser significance that is not set, or if it is not set in m, but all bits of lesser significance are. We now apply this to large counters. Let m be a number encoded in a large counter S. Then the set that encodes $m+1$ comprises exactly all states (p, η) such that (p, η) is already included in X and there is η' with $\langle \eta'_{\bar{\texttt{x}}} \rangle < \langle \eta_{\bar{\texttt{x}}} \rangle$ such that $(p, \eta) \notin S$, and all such (p, η) such that $(p, \eta) \notin S$, but for all (p, η') with $\langle \eta'_{\bar{\texttt{x}}} \rangle < \langle \eta_{\bar{\texttt{x}}} \rangle$ we have $(p, \eta') \in S$. This, incrementation, and a test for 0 are expressed by the following formulas:

$$inc(X) = (X \wedge \texttt{E}(\neg(p \vee i)\ \texttt{U}^{>0}\ (p \wedge \neg X))) \vee (\neg X \wedge \texttt{A}(\neg p\, \texttt{U}^{>0}\ (i \vee p \wedge X)))$$
$$dec(X) = (X \wedge \texttt{E}(\neg(p \vee i)\ \texttt{U}^{>0}\ (p \wedge X))) \vee (\neg X \wedge \texttt{A}(\neg p\, \texttt{U}^{>0}\ (i \vee p \wedge \neg X)))$$
$$eq_0(X) = \neg X \wedge \texttt{A}(\neg p\, \texttt{U}^{>0}\ (p \wedge \neg X))$$

▶ **Lemma 13.** *Let S be a large counter in $\mathcal{T}_{\mathcal{A}}$ such that S encodes m for $m \in [2^{2^n}]$. Then $inc(S)$ encodes $m+1$ modulo 2^{2^n} and $dec(X)$ encodes $m-1$ modulo 2^{2^n}. Finally, $(p, \eta) \in eq_0(S)$ iff S encodes 0.*

Proof. We show the claim for $inc()$. Let S encode m. The left part of the disjunction concerns the case of a bit that is already set in the representation of m, i.e. $(p, \eta) \in S$ such that $\langle \eta \rangle$ is some $k \in [2^n]$. Then (p, η) is in the representation of $m+1$ modulo 2^{2^n} iff there is $k' \in [k]$ such that $(p, \eta') \notin S$ if η' encodes k'. This is formalised in the $\texttt{EU}$-formula, which requires the existence of a path of length different than 0 where, at the first occurrence of p after the initial state, $\neg S$ holds. Moreover, since i can also not hold, that path must go exactly once through the gadget $\texttt{Decr}^*_{\texttt{x}}$. By Lem. 12, respectively its analogue for $\texttt{Decr}^*_{\bar{\texttt{x}}}$, we obtain that, for each $k' < k$, there is a path through this gadget such that $\bar{\texttt{x}}$ encodes k' after leaving it, and no other paths through this gadget exist. Hence, the $\texttt{EU}$-formula only holds if there is $k' \in [k]$ representing the bit of lesser significance than k not set in the representation of m. The other disjunct follows the same pattern, except here, the $\texttt{AU}$-formula formalises the forall quantifier in the logic of binary incrementation, and the proposition i moves to formalise that each path either goes through i and, hence, is of no concern, or ends up in the location p such that the corresponding lower bit is set.

The formula for decrementation follows the same logic. Finally, a set S encodes 0 iff it contains no states (p, η) with η of any kind. In other words, (p, η) is in (the semantics of) $eq_0(S)$ iff it is not in S and, no matter which path is taken through one of the two gadgets, one ends up outside of S upon reaching the location p for the first time, i.e. no matter how the clocks in $\bar{\texttt{x}}$ are changed, one cannot reach S. ◀

We add that the following are also expressible via the formulas given below: The fact that a large counter encodes 1, the fact that it encodes a number greater than 0 or 1, and the fact that it encodes a number less than $2^{2^n} - 1$:

$$eq_1(X) = eq_0(dec(X)) \quad eq_{2^{2^n}-1}(X) = eq_0(inc(X)) \quad gt_1(X) = \neg eq_0(X) \wedge \neg eq_0(dec(X))$$
$$gt_0(X) = \neg eq_0(X) \quad lt_{2^{2^n}-1}(X) = \neg eq_0(dec(X))$$

Lower Bound. A matching lower bound can be obtained by a polynomial reduction from problem stated in Prop. 1. We construct, given such a DTM $\mathcal{M}$ and an $n \in \mathbb{N}$, a TA $\mathcal{A}_{\mathcal{M},n}$ and a TRCTL formula $\varphi_{\mathcal{M},n}$ each of polynomial size in $|\mathcal{M}|$ and n, such that $\mathcal{A}_{\mathcal{M},n} \models \varphi_{\mathcal{M},n}$ iff there is a 2^{2^n}-certificate for $\mathcal{M}$ and n. Given the previous work on encodings or large numbers, the existence of such a certificate is easily defined in TRCTL, as we will see below.

▶ **Theorem 14.** *The model checking problem for* TRCTL *over Timed Automata is* 2-EXPTIME*-hard.*

Proof. Let $\mathcal{M}$ and n be given. Let $\hat{\Gamma} = \{a_1, \ldots, a_m\}$ and $\hat{\delta}$ be as defined in Sect. 2.1, resulting from $\mathcal{M}$'s state set, tape alphabet and transition function. Let

$$\varphi_{\mathcal{M},n} := \mathtt{rec}_{(q_{\mathsf{acc}},\square)} \begin{pmatrix} \vdots & & \\ \mathcal{C}_{a_i}(t,s) & . & chk_{a_i}(t,s) \ \vee \bigvee_{(b_1,b_2,b_3,a_i)\in\hat{\delta}} nxt_{b_1,b_2,b_3}(t,s) \\ \vdots & & \end{pmatrix} (\langle 2^{2^n}-1\rangle, \langle 0\rangle)$$

where

$$chk_a(t,s) := \begin{cases} eq_0(s) \vee eq_{2^{2^n}-1}(s) & , \text{ if } a = \# \\ eq_0(t) \wedge eq_1(s) & , \text{ if } a = (q_0, \square) \\ gt_1(s) \wedge lt_{2^{2^n}-1}(s) & , \text{ if } a = \square \\ \mathtt{ff} & , \text{ otherwise} \end{cases}$$

$$nxt_{b_1,b_2,b_3}(t,s) := gt_0(t) \wedge gt_0(s) \wedge lt_{2^{2^n}-1}(s) \wedge \mathcal{C}_{b_1}(dec(t), dec(s)) \wedge \mathcal{C}_{b_2}(dec(t), s) \wedge \mathcal{C}_{b_3}(dec(t), inc(s))$$

Let $\mathcal{A}$ be the TA from Fig. 2. Then $\mathcal{T}_{\mathcal{A}} \models \varphi_{\mathcal{M},n}$ iff there is a 2^{2^n}-certificate for $\mathcal{M}$. This follows from the fact that the definition of the $\mathcal{C}$ mirrors the pattern of the certificate *Cert* described in Sec. 2.1. The arithmetic used is described above. Note that $\varphi_{\mathcal{M},n}$ is well-defined w.r.t. monotonocity of the $\mathcal{C}$ since all of them occur only positively in $nxt_{b_1,b_2,b_3}(t,s)$. The variables s and t occur both positively and negatively but they are not recursion variables, so this is unproblematic. ◀

5 Conclusion & Further Work

We have introduced Timed Recursive Temporal Logic (TRCTL) and shown that its model-checking problem is 2-EXPTIME-complete. Its satisfiability problem is undecidable, this is inherited from Recursive Temporal Logic [10]. TRCTL is strictly stronger in expressive power than its two constituent parts RecCTL and TCTL since either can express properties that the other cannot, namely higher-order properties (cf. [10]) or real-time properties. A fine-grained analysis of the expressive power of TRCTL, i.e. which properties of e.g. TLTS become accessible that are not accessible in TCTL, is still to be done. It should be noted that our lower bounds already hold in the setting with just two clocks, the constructions from [17] carry over with few adaptations.

Further research concerns two angles: practicability and extensions in expressive power. With respect to the former, the 2-EXPTIME-complete model checking problem might seem prohibitive, yet higher-order algorithms are open to optimisations that can yield surprisingly competitive algorithms [9, 14]. The latter angle includes straightforward extensions such as propositions that test for the value of some clock that are unlikely to require new methods, but also more intricate ones like diagonal constraints etc. which, of course, are also likely to lead to undecidability [8].

References

1 R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th Ann. IEEE Symp. on Logic in Computer Science, LICS'90*, pages 414–427. IEEE Computer Society Press, 1990. doi:10.1109/LICS.1990.113766.

2 R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Inform. and Comp.*, 104(1):2–34, 1993. doi:10.1006/inco.1993.1024.

3 R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

4 A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

5 R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Logical Methods in Computer Science*, 3:1–33, 2007. doi:10.2168/LMCS-3(2:7)2007.

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

7 H. Bekić. *Programming Languages and Their Definition, Selected Papers*, volume 177 of *LNCS*. Springer, 1984.

8 P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine, and J. Worrell. Timed temporal logics. In *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, volume 10460 of *LNCS*, pages 211–230. Springer, 2017. doi:10.1007/978-3-319-63121-9_11.

9 F. Bruse, J. Kreiker, M. Lange, and M. Sälzer. Local higher-order fixpoint iteration. In *Proc. 11th Int. Symp. on Games, Automata, Logics, and Formal Verification, GandALF'20*, volume 326 of *EPTCS*, pages 97–113, 2020. doi:10.4204/EPTCS.326.7.

10 F. Bruse and M. Lange. Temporal logic with recursion. In *Proc. 27th Int. Symp. on Temporal Representation and Reasoning, TIME'20*, volume 178 of *LIPIcs*, pages 6:1–6:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.TIME.2020.6.

11 A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

12 E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982. doi:10.1016/0167-6423(83)90017-5.

13 D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983. doi:10.1016/0022-0000(83)90014-4.

14 Y. Hosoi, N. Kobayashi, and T. Tsukada. A type-based HFL model checking algorithm. In *Proc. 17th Asian Symp. on Programming Languages and Systems, APLAS'19*, volume 11893 of *NCS*, pages 136–155. Springer, 2019. doi:10.1007/978-3-030-34175-6_8.

15 M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2005. doi:10.1016/j.jal.2005.08.002.

16 M. Lange and C. Stirling. Model checking fixed point logic with chop. In *Proc. 5th Conf. on Foundations of Software Science and Computation Structures, FOSSACS'02*, volume 2303 of *LNCS*, pages 250–263. Springer, 2002. doi:10.1007/3-540-45931-6_18.

17 F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 387–401. Springer, 2004. doi:10.1007/978-3-540-28644-8_25.

18 M. Müller-Olm. A modal fixpoint logic with chop. In *Proc. 16th Symp. on Theoretical Aspects of Computer Science, STACS'99*, volume 1563 of *LNCS*, pages 510–520. Springer, 1999. doi:10.1007/3-540-49116-3_48.

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

20 A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–309, 1955. doi:10.2140/pjm.1955.5.285.

21 M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004. doi:10.1007/978-3-540-28644-8_33.

Efficient Anytime Computation and Execution of Decoupled Robustness Envelopes for Temporal Plans

Michael Cashmore
Strathclyde University, Glasgow, UK

Alessandro Cimatti
Fondazione Bruno Kessler, Trento, Italy

Daniele Magazzeni
Kings College London, UK

Andrea Micheli
Fondazione Bruno Kessler, Trento, Italy

Parisa Zehtabi
Kings College London, UK

Abstract

One of the major limitations for the employment of model-based planning and scheduling in practical applications is the need of costly re-planning when an incongruence between the observed reality and the formal model is encountered during execution. Robustness Envelopes characterize the set of possible contingencies that a plan is able to address without re-planning, but their exact computation is expensive; furthermore, general robustness envelopes are not amenable for efficient execution.

In this paper, we present a novel, anytime algorithm to approximate Robustness Envelopes, making them scalable and executable. This is proven by an experimental analysis showing the efficiency of the algorithm, and by a concrete case study where the execution of robustness envelopes significantly reduces the number of re-plannings.

2012 ACM Subject Classification Computing methodologies → Robotic planning

Keywords and phrases Temporal Planning, Robustness Envelopes

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.13

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

When planning and scheduling techniques are employed in practical applications, one of the major problems is the need for on-line re-planning when the observed contingencies are not aligned with the ones that were considered at planning time. These situations are common, because it is arguably impossible to predict the entire range of situations an autonomous system can encounter, especially when the planning domain encompasses time and temporal constraints. Unfortunately, re-planning can be costly in terms of time, and computational resources can be scarce on-board, so limiting the use of re-planning is very important for practical purposes. In principle, it is also possible to continue with the execution of a plan even when the observed contingencies are unexpected, optimistically hoping for a successful completion. However, this approach offers no formal guarantee, and is prone to the risk of continuing execution of a plan that is bound to fail.

Several approaches have been proposed in the literature to address this problem (see [13] for a survey focused on robotics). Some authors propose to post-process plans and generalize them relying on the scheduling constraints that are relevant for execution [17, 15, 10]. Another line of research focuses on the creation of "least commitment plans", i.e. plans that are left partially open by the planner so that the execution can be adapted to some variation in the contingencies [11, 20, 9, 4, 19]. Others tackled the idea of transforming temporal plans with no adaptability into flexible plans [7]. Finally, one can explicitly model the uncertainties in the planning problem and construct a plan that offers formal guarantees with respect to such a model. Examples include Conformant and Contingent Planning [12], Probabilistic Planning [14] and Strong Temporal Planning with Uncertain Durations [5] that considers temporal uncertainty in the durations of actions.

Recently, *Robustness Envelopes* (REs) have been proposed to overcome several limitations of the approaches mentioned above. REs formally capture the possible contingencies that a given temporal plan, obtained by planning in a *deterministic domain*, can deal with, without having to re-plan [2]. REs are regions defined over a set of numeric parameters that represent possible contingencies, and contain all the parameter valuations ensuring plan validity. In general, REs may be non-convex, and can express dependencies between the parameters. However, the technique proposed in [2] has two main drawbacks limiting its practical applicability. First, the exact computation of REs is extremely expensive: the proposed approach is doubly exponential in the size of the planning problem. Second, REs in their general form are not suited for efficient execution: the dependencies among parameters might require run-time reasoning.

In this paper, we overcome these limitations, achieving scalability and executability. We focus on *Decoupled Robustness Envelopes* (DREs), i.e. hyper-rectangular REs where the dependencies among parameters are not present, and are thus much easier to execute. Our first contribution is a novel and scalable algorithm for computing DREs as sound approximations of REs. A sound approximation ensures that every point within the DRE belongs within the RE. The algorithm is anytime, and proceeds by incrementally under-approximating the RE with increasingly large DREs. The algorithm can be stopped at any time, providing a meaningful result already amenable to start execution. In its general formulation, the RE for a given plan is naturally modeled as a quantified first order formula in the theory of Linear Real Arithmetic. Our algorithm does not need to precisely compute the quantifier-free description of the RE (which requires an expensive step of quantifier elimination, and is ultimately responsible for the inefficiency demonstrated in [2]). Rather, it starts from a degenerate DRE consisting of a single point, and progressively tries to enlarge it along different dimensions, checking if each extension is contained in the RE, until a given precision is reached. The algorithm relies on *quantifier-free* queries to a Satisfiability Modulo Theory [1] solver.

Our second contribution is to demonstrate the practical use of DREs in a robotic executor, extending the classical flow from planning to execution to re-planning, as follows. First, a plan is generated from a deterministic model using temporal planning technologies, and transformed into a Simple Temporal Network (STN) formulation [6]; at this point, we parametrize the durations of some of the actions in the plan and/or the consumption rates in the domain specification. DREs are then computed for the introduced parameters and passed to the executor. In turn, the dispatching of the actions in the STN plan begins and continues until one observed duration or consumption rate happens to be outside of the DRE. At this point, the executor detects that the plan is no longer guaranteed to succeed, and re-planning is triggered.

The proposed approach was implemented in the ROSPlan [3] framework, and experimentally evaluated along two directions. The algorithm for DRE generation was compared against the base line in [2], demonstrating orders-of-magnitude improvements compared to the exact computation of REs, and the ability to deal with a much larger number of parameters. The overall execution loop has been evaluated on a family of concrete case studies in a logistic domain, showing that the use of DREs, compared to the optimistic executor in ROSPlan, significantly reduces the number of re-plannings and improves the execution success-rate.

2 Background

We consider planning problems expressed in the PDDL 2.1 [8] temporal planning language; for the sake of brevity we do not report the full syntax and semantics of such planning problems, but we directly introduce the parametrized planning problem idea adapted from [2].

▶ **Definition 1.** *A **parametrized planning problem** $\mathcal{P}_\Gamma$ is a tuple $\langle \Gamma, \mathcal{P} \rangle$, where Γ is a finite set of real-valued parameters $\{\gamma_1, \cdots, \gamma_n\}$ and $\mathcal{P}$ is a PDDL 2.1 planning problem in which action conditions, action effects, goals and initial states can all contain parameters.*

Intuitively, symbols (from a known set Γ) can be used in expressions where real-typed constants are usually allowed.

As customary in many cases of plan execution, we use plans expressed as Simple Temporal Networks (STN) [6]. An STN plan is a set of constraints of the form $t_i - t_j \leq k$ where t_i and t_j are time points linked to action happenings (i.e. either the start or the end of an action instance in the plan) and $k \in \mathbb{Q}$. In addition, we allow parameters in the plan specification by generalizing the notion of STN plans.

▶ **Definition 2.** *A **parametrized STN plan** π_Γ for a parametrized planning problem $\mathcal{P}_\Gamma \doteq \langle \Gamma, \mathcal{P} \rangle$ is an STN Plan where some constraints are in the form $t_i - t_j \leq \gamma_i$ where t_i and t_j are time-points of the STN plan and $\gamma_i \in \Gamma$.*

We define the Robustness Envelope (RE) for a parametrized problem and plan as the set of possible values for the parameters that make the plan valid when the symbols are substituted with values in the plan and problem specifications. In order to compute the RE, [2] defines a set of logical formulae that characterize the RE and use quantifier elimination techniques (e.g. [18]) to explicitly construct the region. The encoding is divided in three expressions: indicated as $enc^{\pi_\Gamma}_{tn}$, $enc^{\pi_\Gamma}_{\mathit{eff}}$ and $enc^{\pi_\Gamma}_{\mathit{proofs}}$. The formula $enc^{\pi_\Gamma}_{tn}$ encodes the temporal constraints imposed by π_Γ limiting the possible orderings of time points. The formula $enc^{\pi_\Gamma}_{\mathit{eff}}$ encodes the effects of each time point on the state variables, while $enc^{\pi_\Gamma}_{\mathit{proofs}}$ encodes the validity properties of the plan, namely that the conditions of each executed action are satisfied, that the goal is reached, and that the ϵ-separation constraint [8] imposed by PDDL 2.1 is respected. Then, let $\bar{X}$ be the set of all the variables appearing in the formulae above except the parameter values, the RE is characterized by all the models of the following formula.

$$\exists \bar{X}.(enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{\mathit{eff}}) \wedge \forall \bar{X}.((enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{\mathit{eff}}) \rightarrow enc^{\pi_\Gamma}_{\mathit{proofs}})$$

As observed in [2], any under-approximation of the RE gives sound information on the contingencies in which the plan is guaranteed to be valid; in particular, a convenient restriction for the representation and handling of REs is to associate a closed interval of possible values to each parameter, defining an hyper-rectangle. If such hyper-rectangle is contained in the RE, we have a "Decoupled Robustness Envelope" (DRE) that retains the guarantees of the RE but avoids the complexity of inter-dependencies among parameters.

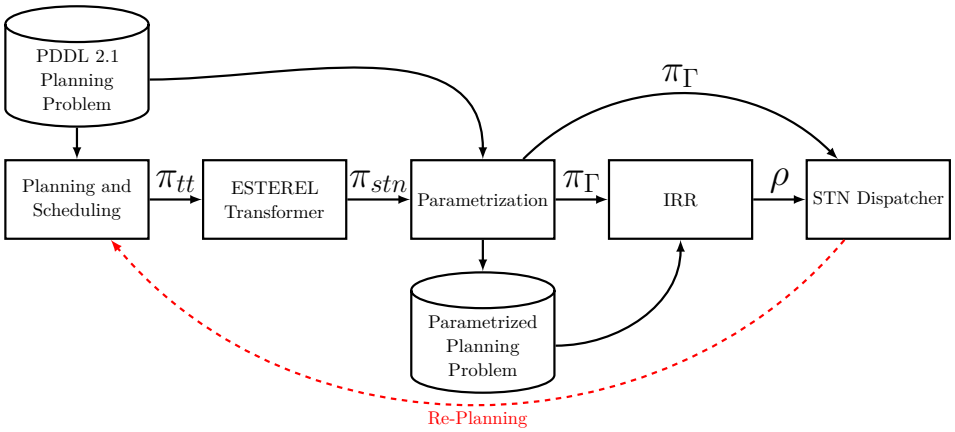


Figure 1 Overview of the proposed flow.

▶ **Definition 3.** *A **Decoupled Robustness Envelope** for a parametrized planning problem $\mathcal{P}_\Gamma$ and STN plan π_Γ is a bound assignment $\rho : \Gamma \to \mathbb{Q}_{>=0} \times \mathbb{Q}_{>=0}$, such that any parameter assignment $v : \Gamma \to \mathbb{Q}_{>=0}$, with $l \leq v(\gamma) \leq u$ and $\langle l, u \rangle \doteq \rho(\gamma)$, is contained in the RE for $\mathcal{P}_\Gamma$ and π_Γ.*

Note that many DREs are possible for a given problem and plan: it suffices that all the assignments allowed by the DRE are points in the RE. In this paper, we elaborate on this idea and propose an algorithm that incrementally builds DREs that are contained within the unconstrained RE without paying the cost of explicitly computing the RE itself.

Finally, we highlight that given any two DREs ρ_1 and ρ_2 three cases are possible: either ρ_1 is subsumed by ρ_2 (i.e. for each parameter γ, $\rho_1(\gamma) \subseteq \rho_2(\gamma)$), or ρ_1 subsumes ρ_2, or the two DREs are incomparable. Hence, there is no absolute best DRE in general: we aim for a DRE that is not subsumed by any other, but there can be multiple DREs with this property.

3 Execution Flow

The general idea we pursue in this paper is to exploit the information and the generalization provided by the synthesis of REs to limit the number of re-plannings and increase the success-rate in execution. In particular, we propose the flow from planning to execution depicted in Figure 1. Starting from a planning problem formulation expressed in PDDL 2.1, we use any off-the-shelf temporal planner[1] to compute a timed sequence of actions that reaches the goal from the initial state. We call this plan "time-triggered" (indicated with π_{tt}) in the picture. This plan is not natively amenable for execution because it defines one specific trace that does not allow any adaptability: it is extremely unlikely for a real system to be perfectly controlled to satisfy a specific trace. Hence, π_{tt} needs to be converted in a flexible, executable STN (π_{stn}) by the ESTEREL transformer of ROSPlan. The usual flow would pass this STN directly to the dispatcher for translating the plan actions into commands for the robotic platform at the proper time. Instead, here we pre-process this

[1] Several existing PDDL planners are unable to generate flexible STNs either because of an implementation limitation or because the technique does not allow it (e.g. SAT-based planners). Our approach is able to generate DREs from these planners as well, and work in concert with existing algorithms for the execution of STNs.

plan using REs in the hope of generalizing its applicability and reducing the number of re-plannings. In particular, the STN plan is passed to a parametrization component that re-reads the planning problem formulation and enriches it with parameters, generating a Parametric Planning Problem and a parametric STN plan (π_Γ). Those are the inputs for the computation of the RE. In our flow, for performance reasons and to avoid complex run-time reasoning, instead of computing the exact, unconstrained RE, we use a novel algorithm, called Incremental Rectangular-Robustification (IRR for short), that computes a DRE. The algorithm is anytime, so that it is possible to retrieve unfinished computations and exploit them in execution: in fact, any under-approximation of the final result retains the needed properties of the RE. At this point, we pass the DRE (ρ) together with the parametrized STN plan to the STN dispatcher. We modified the dispatching algorithm to exploit the information in the DRE to limit the re-plannings to situations where they are needed. In particular, the dispatcher translates the actions, while checking that the observed values for the parameters (being either action durations, resources or rates) fall within the bounds imposed by ρ. If this is not the case, re-planning is needed and the whole flow is re-executed.

Parametrization. The first non-standard step highlighted in Figure 1 is the parametrization. In fact, there are multiple ways in which parameters can be added to a deterministic temporal planning problem to characterize useful quantities for execution. In general, one can parametrize any numeric quantity in the planning problem whose value might differ from the environment in which the plan will be executed. In order to be useful for the STN dispatcher, however, such quantities must be eventually observable (directly or indirectly). Otherwise, it is impossible for the executor to check whether the RE is still satisfied or if a re-planning is needed. In this paper experimentation, we focused on two such quantities, namely the durations of actions and resource consumption rates. The former is a classical source of uncertainty when temporal planning is employed in a robotics scenario, the latter is another source of uncertainty that can perturbate the execution of a plan, for example when the resource harvesting is not fully controllable (e.g. a solar panel yield depends on the weather) or when the consumption is not fully predictable (e.g. the battery consumption is very hard to precisely estimate as it depends on temperature, exact capacity and so on).

4 Incremental Rectangular-Robustification

We now present our novel algorithm for incrementally computing decoupled robustness envelopes. We call this algorithm "Incremental Rectangular-Robustification" (IRR).

The idea behind the algorithm is to construct incrementally better hyper-rectangular under-approximations of the RE for a given problem and plan. In fact, this constitutes a direct way of computing a DRE by generate-and-test. The starting point is the de-generated hyper-rectangle composed of the single point given by the parameter values of the original plan. The algorithm tries to extend the hyper-rectangle along one dimension (i.e. it tries to widen the interval of possibilities associated to one of the parameters) and checks if the resulting hyper-rectangle is in fact an under-approximation of the RE. If it is, the new hyper-rectangle is kept as it is guaranteed to be a valid DRE. Otherwise, another dimension or another increment is chosen for the algorithm to proceed. The general intuition behind the algorithm is depicted in Figure 2.

Algorithm 1 reports the pseudo-code of IRR. The formula enc_{valid} is computed once and off-line. It corresponds to the basic requirements for the hyper-rectangle to be a valid DRE: only parameter values that are not contradicting the STN plan and the causal flow of effects

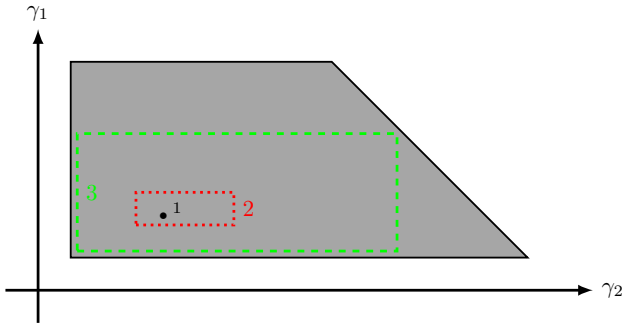


Figure 2 A graphical representation of IRR: starting from the parameter values from the original plan (depicted as the black point), IRR tries to construct increasingly better under-approximations (the colored rectangles) of the RE (the gray area), without actually computing it. Upon termination, each edge of the resulting DRE is guaranteed to be at most β apart from the border of the actual region.

are admissible. This is the same as the first piece of the logical formulation in [2], but luckily it is the easier part of the quantification and can be efficiently computed. Then, the IRR function is in charge of computing a hyper-rectangle R maintaining the following invariant: at each step, R is a subset of the RE of the problem. The hyper-rectangle R is represented as a pair of bounds (lower- and upper-) assigned to each parameter (this directly models a DRE as per Definition 3), and is initialized (line 3) with the values of the non-parametric plan π. The algorithm uses two functions to control how the hyper-rectangle is transformed from one cycle to the next. Δ associates to each parameter a number that is the value used to increase the upper-bound or to decrease the lower-bound for that parameter. The initial value for Δ is the original value of the parameter scaled by a weight for such parameter, but any positive number bigger than β is enough to guarantee soundness and termination of the algorithm. Note that these weights can be used to express preferences on the parameters: a higher weight pushes the algorithm to expand a specific parameter more than others. The function Θ is used to decide in which direction the interval of a parameter can be extended. Two directions are possible: `UB` indicates that we want to extend the upper-bound and `LB` indicates that we want to decrease the lower-bound (line 10). Initially both directions are possible, but when we discover (line 13) that one direction is infeasible with the current Δ, we remove this direction from the possibilities. This process will be continued with the current Δ values until expansion in all directions is infeasible. At this stage the value of Δ is halved, eventually reaching a value lower than β. Each time that the values of Δ are updated, we reset Θ to allow expansion in both directions once again.

The main loop of the algorithm continues until all the values of Δ are lower than β: this is to guarantee that the minimum distance from each border of the hyper-rectangle and the border of the actual RE is at most β. The algorithm picks a parameter $\tilde{\gamma}$ to be analyzed among the parameters having at least one direction available in Θ and that have not converged already (line 7); then, it generates a candidate hyper-rectangle R' by extending either the lower- or the upper- bound of $\tilde{\gamma}$. At this point, we check if R' is contained in the RE or not (line 12). If it is, we keep it and continue the loop, otherwise, we discard this hyper-rectangle and we record that with the current Δ we cannot extend $\tilde{\gamma}$ in this direction by removing the direction θ from $\Theta(\tilde{\gamma})$. Moreover, if no direction is left for $\tilde{\gamma}$, we halve its value of Δ and reset Θ so that $\tilde{\gamma}$ can be tentatively extended again using a smaller step (lines 15–16).

Algorithm 1 Incremental Rectangular-Robustification.

1: $enc_{valid} \leftarrow$ QuantifierElimination$(\exists \bar{X}.enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{eff})$
2: **function** IRR$(\beta : \mathbb{Q}_{>0})$
3: $R \leftarrow \{\gamma \rightarrow [\pi(\gamma), \pi(\gamma)] \mid \gamma \in \Gamma\}$
4: $\Delta \leftarrow \{\gamma \rightarrow max(\pi(\gamma) \times \omega_\gamma, \beta) \mid \gamma \in \Gamma\}$
5: $\Theta \leftarrow \{\gamma \rightarrow \{\texttt{UB}, \texttt{LB}\} \mid \gamma \in \Gamma\}$
6: **while** $\exists \gamma \in \Gamma.\Delta(\gamma) \geq \beta$ **do**
7: $\tilde{\gamma} \leftarrow$ Pick$(\{\gamma \mid \gamma \in \Gamma \wedge \Theta(\gamma) \neq \emptyset \wedge \Delta(\gamma) \geq \beta\})$
8: $\theta \leftarrow$ Pick$(\Theta(\tilde{\gamma}))$
9: $[l, u] \leftarrow R(\tilde{\gamma})$
10: **if** $\theta = \texttt{UB}$ **then** $u \leftarrow (u + \Delta(\tilde{\gamma}))$ **else** $l \leftarrow (l - \Delta(\tilde{\gamma}))$
11: $R' \leftarrow \{\gamma \rightarrow R(\gamma) \mid \gamma \in \Gamma \wedge \gamma \neq \tilde{\gamma}\} \cup \{\tilde{\gamma} \rightarrow [l, u]\}$
12: **if** CheckInEnvelope(R') **then** $R \leftarrow R'$
13: **else**
14: $\Theta(\tilde{\gamma}) \leftarrow \Theta(\tilde{\gamma}) \setminus \theta$
15: **if** $\Theta(\tilde{\gamma}) = \emptyset$ **then**
16: $\Delta(\tilde{\gamma}) \leftarrow \Delta(\tilde{\gamma})/2$; $\Theta(\tilde{\gamma}) \leftarrow \{\texttt{LB}, \texttt{UB}\}$
17: **return** R
18: **function** CheckInEnvelope(R)
19: $enc_R \leftarrow \bigwedge_{\gamma \in \Gamma, [l,u] = R(\gamma)} l \leq \bar{\gamma} \wedge \bar{\gamma} \leq u$
20: **if** IsSAT$(enc_R \wedge \neg enc_{valid})$ **then return** false
21: **else**
22: **return** IsValid$((enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{eff} \wedge enc_R) \rightarrow enc^{\pi_\Gamma}_{proofs})$

The core part of the algorithm consists in checking a candidate hyper-rectangle for containment in the actual RE, without explicitly computing the region itself. This is done via the CheckInEnvelope function that performs two SMT checks corresponding to the two quantifiers appearing in the RE logical formulation of [2]. The first check looks for points belonging to R that are not parts of the validity region enc_{valid}, the second checks if the rectangle (together with the guarantees from the plan and the effects) implies the proof requirements characterizing the REs. The important point here, is that both checks are quantifier-free, i.e. no quantifier elimination is involved.

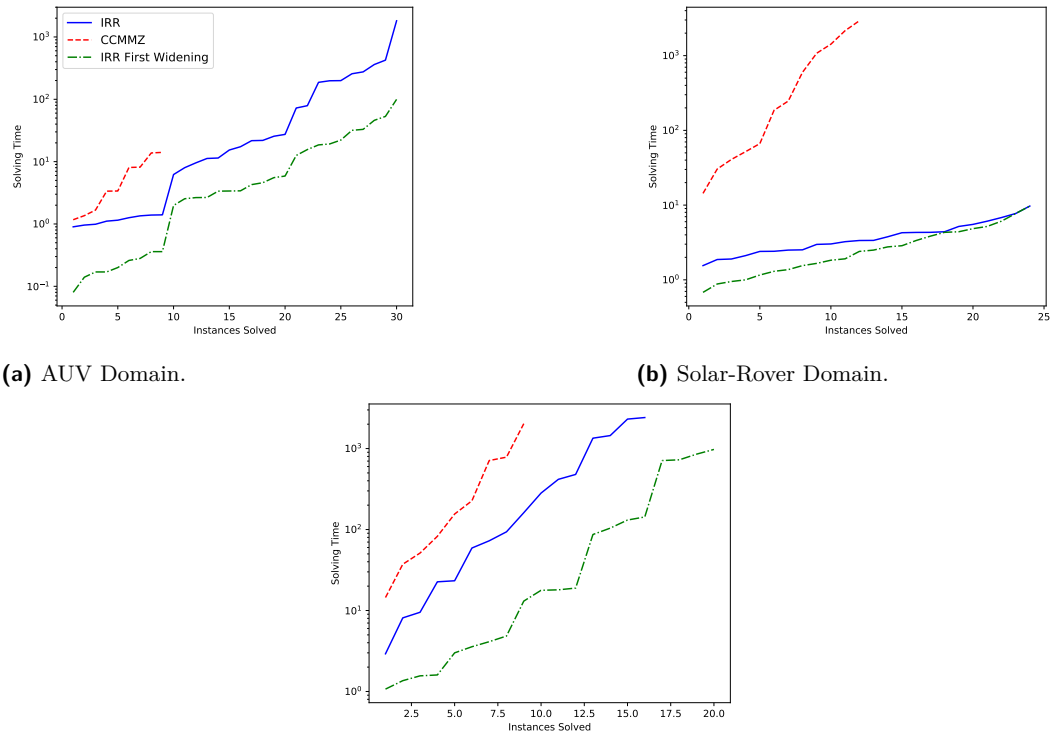
▶ **Theorem 4.** *The CheckInEnvelope(R) function returns true if and only if R is a valid DRE.*

Proof. The algorithm logically checks the following formula: $\neg(\exists \bar{\Gamma}.enc_R \wedge \neg enc_{valid}) \wedge \forall \bar{\Gamma}, \bar{X}.(enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{eff} \wedge enc_R) \rightarrow enc^{\pi_\Gamma}_{proofs}$, that can be rewritten as $\forall \bar{\Gamma}.enc_R \rightarrow (enc_{valid} \wedge (\forall \bar{X}.(enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{eff}) \rightarrow enc_{proofs}))$ that states that enc_R is a subset of the encoding of the RE. Then, for Definition 3, R is the encoding of a valid DRE. ◀

An interesting feature of the algorithm is that it is "anytime", i.e. at each time, we can take the hyper-rectangle R and we have the guarantee that R is contained in the RE and is thus a valid DRE. Moreover, the algorithm is guaranteed to terminate if the RE is finite in all dimensions.

▶ **Theorem 5.** *If the robustness envelope is bounded in all dimensions, IRR always terminates.*

Proof. All the values in Δ are initially positive and whenever the candidate rectangle is found to exit the RE (line 13) one of the values in Δ is halved. Eventually all the parameters will be considered and they will be eventually found to exit the RE because it is bounded in all dimensions. Therefore, all the values of Δ will become smaller than β. ◀



(a) AUV Domain.

(b) Solar-Rover Domain.

(c) Generator Linear Domain.

Figure 3 Scalability experiments on [Cashmore *et al.*, 2019] domains: the number of solved instances (sorted by difficulty for each solver) is considered on the X axis and is compared with the logarithmic time needed to solve each instance (lower, longer lines are better).

We highlight that IRR is in fact an optimization procedure that incrementally maximizes the size of a starting DRE, terminating when a maximal DRE is found within the given precision limit β.

5 Experiments

We now present our empirical analysis that comprises three sets of experiments. The first aims at showing the superior performance of IRR as compared with the logical approach of [2]. The second shows a practical use-case of the execution flow proposed in this paper when the duration of actions is uncertain. In the third experimentation we use our DRE technique to execute plans when the consumption rates of resources is uncertain.

IRR. We start by considering the experimental dataset and the tool (hereafter called CCMMZ) provided in [2]. The benchmarks use a varying number of parameters; in particular, AUV ranges from 1 to 8 parameters, Generator Linear from 1 to 4 and Solar Rover between 1 and 4. We compare our IRR implementation with CCMMZ on all the available instances and domains, measuring the total run-time and using the "decoupled envelope generation" functionality of the tool. Moreover, in order to take into account the anytime nature of IRR, we also measure the time at which the rectangle R in IRR widens and becomes different than a single point (i.e. we measure the first time the Algorithm 1 reaches line 17) and we call this timing "IRR First Widening". In all our experiments we

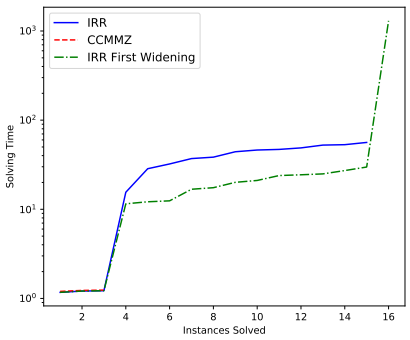


Figure 4 Scalability experiments on the delivery domain: the number of solved instances (sorted by difficulty for each solver) is considered on the X axis and is compared with the logarithmic time needed to solve each instance (lower, longer lines are better).

set $\beta = 1$ and all $\omega_i = 1$ to find the decoupled region approximated to a single unit with no preferences among the parameters (obviously, we set the same parameter preference also in CCMMZ). We executed all of the instances on a Xeon E5-2620 2.10GHz machine setting a time limit of 3600s and a RAM memory limit of 20GB.

Figure 3 shows the result of this analysis. IRR is able to solve many more instances than CCMMZ and is consistently quicker. Moreover, we note how the first widening is often encountered quite early in the execution, marking the margin for anytime exploitation of IRR. In fact, after the first widening, IRR already computed a meaningful and non-trivial under-approximation of the RE that can be used for execution. This is particularly evident in the Generator Linear domain where the algorithm is unable to fully terminate in some cases, but the first widening point is reached.

In addition to these domains, we also experiment with several instances of a service-robot domain that we also use for the following execution experimental analysis. The domain, called "Robot Delivery" is a simplified version[2] of the domain used in the Planning and Execution Competition for Logistics Robots in Simulation [16]. The domain comprises a fleet of small robots that can navigate in an euclidean graph. These robots are tasked to pick and deliver orders within a deadline. Collecting orders requires two robots present at a machine. We scaled the number of parameters in the instances between 1 and 33. Figure 4 shows the scalability of IRR and CCMMZ on this domain. These instances are much harder for both the solvers compared to the previous domains; in fact, CCMMZ is only able to solve 3 instances, while IRR is able to solve 15 of them. Also in this case, the anytime nature of IRR is evident by observing the difference from the first widening and the algorithm completion.

Finally, we investigate how quickly the IRR algorithm converges in our experiments. We define convergence at step i in a run of IRR that terminates with hyper-rectangle R_{end} as follows (R_i indicates the hyper-rectangle at step i).

$$Convergence(i) = \frac{\sum_{[l,u] \in R_i} u - l}{\sum_{[l,u] \in R_{end}} u - l} \times 100$$

[2] A simplified RCLL domain was used because the PDDL provided in the RCLL image is not complete and the RCLL simulation requires external processes, e.g. a referee box. We are interested in the flexible execution success rate, so we created PDDL instances encoding logistics problems without any external processes.

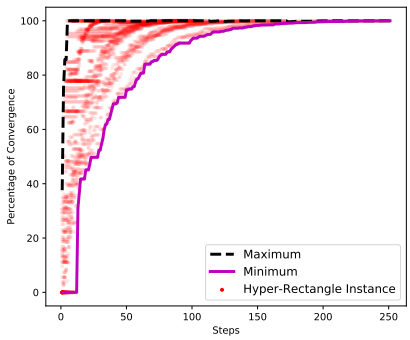


Figure 5 Convergence of IRR in terms of steps: each red dot is a DRR computed by IRR and the plot shows the progression in terms of convergence at each step of the algorithm. The purple line indicates the poorest convergence percentage for each step in any experiment; similarly the black, dashed line shows the best convergence.

Table 1 Coverage and average number of re-plans in the duration-uncertain delivery domain.

Executor	1 Parameter		2 Parameters		3 Parameters		4 Parameters		5 Parameters	
	Coverage	Avg Replans	Coverage	Avg Replans	Coverage	Avg Replans	Coverage	Avg Replans	Coverage	Avg Replans
DREEx	**92.2%**	**0.1**	**85.8%**	**0.2**	**83.2%**	**0.2**	**72.8%**	**0.1**	**63.0%**	**0.1**
BLEx(0)	0.0%	NA	0.0%	NA	0.0%	NA	0.0%	NA	0.0%	NA
BLEx(10)	24.5%	1.0	4.8%	1.0	0.8%	1.2	0.2%	0.8	0.0%	NA
BLEx(20)	44.4%	0.8	19.9%	1.0	6.3%	1.0	2.4%	1.6	0.8%	1.9
BLEx(30)	58.9%	0.7	34.0%	0.9	18.8%	1.2	9.2%	1.2	4.5%	1.4
BLEx(40)	68.8%	0.6	52.2%	0.8	34.7%	1.0	23.8%	1.1	11.8%	1.5
BLEx(50)	75.0%	0.5	62.2%	0.7	49.0%	0.9	37.8%	1.1	27.9%	1.2
BLEx(60)	78.8%	0.4	69.0%	0.5	59.4%	0.7	53.4%	0.9	44.5%	1.1

Intuitively, this gives the percentage of the region covered by R_i with respect to R_{end}. (Note that R_{end} contains R_i because the IRR algorithm only expands previous hyper-rectangles.) Figure 5 shows, for all problems solved by IRR in our benchmark set, the percentage of convergence achieved after any number of steps of the IRR algorithm. Clearly from the plot, in a limited number of steps we often approximate very well the final intervals; in particular, within 50 steps we already cover more than 70% of the final sum of the interval sizes in all the cases.

Duration-Uncertain Flexible Execution. We use the Robot Delivery domain to investigate the merits of an on-line plan executor equipped with our IRR algorithm. In particular, we begin by focusing the analysis on the number of re-plans and on the plan execution success rate when only the duration of actions is uncertain during execution. In this domain, a robot has to collect a spindle from a shelf, construct a base by performing six steps (possibly in parallel), then mount a number of rings, and finally deliver the order. Orders have deadlines that must be met for delivery. The domain allows the agent to drop an order and restart from scratch with a new one at any time, but this disposal action takes some time (10 seconds in our case) and the robot needs to navigate on a symbolic euclidean graph to pick the parts, assemble and deliver the order. Each instance is simulated in an environment where actions have a non-deterministic duration described by a normal distribution with a minimum value. Due to the difficulty in manipulation tasks, the actions executed for preparing the base (in which the robots interact with machines) have the highest degree of variance. These

Algorithm 2 STN Dispatch.

```
1:  function STNDispatch(π_stn, ρ)
2:    finished = false
3:    while ¬finished do
4:      for each node n ∈ π_stn do
5:        min, max ← MinMaxDispatchTime(n, π_stn, ρ)
6:        if n is action start then
7:          if (min ≤ n ≤ max) ∧ ¬started(n) then
8:            StartExecuting(n)
9:          else if (n ≥ max) ∧ ¬started(n) then
10:           finished = true
11:       else if n is action end then
12:         if (n ≥ max) ∧ ¬Completed(n) then
13:           finished = true
14:         else if (n ≤ min)∧ Completed(n) then
15:           finished = true
16:   return GoalsAchieved( )
```

actions have mean durations of 120, 130, 140, 150, 160 and 170 seconds, and a standard deviation of 70. Due to this uncertainty and the presence of deadlines for the order delivery, the execution of a plan can fail even when a re-planning schema is employed. We generated a total of 100 problems by varying the deadlines for the orders.

Our DRE-based approach was implemented in ROSPlan, as described in Section 3. The STN dispatcher starts the execution of actions following the temporal constraints of the STN: the process is illustrated in Algorithm 2. For each node, the minimum and maximum dispatch times are calculated during execution (line 5). The dispatch ends when an action completes outside of the temporal constraints allowed by the STN, or has not been started after the maximum allowed dispatch time. When the dispatch ends, it returns *true* if the goals have been achieved; otherwise, re-planning is triggered as shown in Figure 1. The system will continuously attempt to re-plan until the deadlines make the PDDL planning problem unsolvable.

We compare the executor described in Section 3 (indicated as DREEx) against several baselines in which we dispatch the STN plan π_{stn} without parameterization. In such baselines, the executor dispatches the STN plan allowing for a fixed deviation in the duration of actions and ends dispatch only when the action duration falls outside of this interval. This is the optimistic technique for execution implemented in ROSPlan that, differently from DREEx, offers no formal guarantees. We consider baseline executors named BlEx(0) to BlEx(60) allowing for 0% to 60% variability in action duration before triggering a re-plan. For example, given an action with a predicted duration 100 seconds, BlEx(0) will re-plan if the duration is not exactly 100; BlEx(20) will re-plan if the duration is outside of the interval $[80, 120]$. The baseline BlEx(0) corresponds to formally executing the time-triggered plan π_{tt}: re-planning happens if any action duration differs from what was expected in π_{tt}. We highlight that, when DREEx is employed and the observation is within the envelope computed by IRR, we have the formal guarantee of plan success; as soon as one observation is outside of the envelope, we choose to re-plan.

The overarching idea in these experiments is that the planner usually optimistically selects the easier, quicker goal and the agent starts to execute the plan. If the execution of the preparation actions goes overlong, it might become impossible to deliver the order, so the only way to successfully recover is to immediately dispose the current order and switch

Table 2 Coverage and average number of re-plans in the resource-uncertain delivery domain.

Executor	Coverage	Avg Replans
DREEx	**99.2%**	**0.1**
BLEx(0)	1.0%	2.0
BLEx(10)	25.6%	**0.1**
BLEx(20)	50.7%	**0.1**
BLEx(30)	66.4%	**0.1**
BLEx(40)	77.9%	**0.1**
BLEx(50)	82.1%	**0.1**
BLEx(60)	86.8%	**0.1**

to another one with a less imminent deadline. If the executor fails in realizing this situation, it continues to execute the plan until it tries to deliver the order, at which point it realizes that the deadline is not met. Since a lot of time has been wasted in the preparation, it might be impossible to recover from this situation. Ideally, we expect that the predictive power of DREs allows the identification of situations where the deadline cannot be met and a swift re-planning to change the objective order is needed.

Table 1 reports the results of the experiment. We report the coverage percentage (i.e. the percentage of problems successfully executed over the benchmark set) as well as the average number of re-plannings for successful runs. The baseline BLEx(0), not accounting for any variance in action duration, was unable to solve any problem successfully. Allowing for more flexibility in the duration of actions increases the coverage as should be expected. However, the DREEx approach achieves greater coverage than all baselines in all the cases. This is because in this problem, the ability to realize early that the agent is late for the first order and change course of actions to achieve the second order is pivotal for achieving a good success rate.

Resource-Uncertain Flexible Execution. Finally, we show that our flow can be used when parameters are not just action durations. We expanded the delivery domain to consider the battery consumption of the robots. In particular, each action in the revised domain checks that enough battery is present upon start and consumes a fixed amount of battery. We parametrized the consumption rate of actions, so that the DRE will compute the possible consumption values for which a given plan is valid. The executor is then demanded to observe the contingent consumption and possibly invoke a re-planning if the observation does not fall in the DRE prescription. Also in this case, the baselines BLEx(X) invoke the replanning when the battery consumption is observed to be $X\%$ higher or lower than the nominal value.

Table 2 reports the results of the experiment, and shows how the use of DREEx is beneficial for the success-rate achieving an almost perfect success-rate with very few replannings on average.

6 Conclusion

In this paper, we make the case for the use of Robustness Envelopes (RE) in a plan execution framework. We present a novel, anytime algorithm to compute Decoupled Robustness Envelopes (DRE) that is empirically superior to the previously known logic-based construction. Moreover, we demonstrate the usefulness of the produced artifacts by integrating them in the ROSPlan framework and showing on a concrete example the positive impact on the number of re-plannings and the plan success-rate.

In the future, we will consider other kinds of approximations for the robustness envelope (e.g. hyper-octagons instead of hyper-rectangles). We will also explore the link to temporal uncontrollability and non-deterministic planning. Finally, using Incremental Rectangular-Robustification (IRR) in parallel with the dispatcher could allow variation in parameters being considered during execution.

References

1 C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.

2 M. Cashmore, A. Cimatti, D. Magazzeni, A. Micheli, and P.a Zehtabi. Robustness envelopes for temporal plans. In *AAAI*, 2019.

3 M. Cashmore, M. Fox, D.Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtós, and M. Carreras. Rosplan: Planning in the robot operating system. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 333–341, 2015.

4 A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and R. Rasconi. The APSI Framework: a Planning and Scheduling Software Development Environment. In *ICAPS (Application Showcase)*, 2009.

5 A. Cimatti, M. Do, A. Micheli, M. Roveri, and D. Smith. Strong temporal planning with uncontrollable durations. *Artif. Intell.*, 256:1–34, 2018. doi:10.1016/j.artint.2017.11.006.

6 R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.

7 M. Do and S. Kambhampati. Improving temporal flexibility of position constrained metric temporal plans. In *ICAPS*, pages 42–51, 2003.

8 M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003. doi:10.1613/jair.1129.

9 J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003.

10 J. Frank and P. Morris. Bounding the resource availability of activities with linear resource impact. In *ICAPS*, pages 136–143, 2007.

11 M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *AIPS*, pages 61–67, 1994.

12 M. Ghallab, D. Nau, and P. Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

13 Félix Ingrand and Malik Ghallab. Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 247:10–44, 2017.

14 Mausam and A. Kolobov. Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210, 2012. doi:10.2200/S00426ED1V01Y201206AIM017.

15 N. Muscettola. Computing the envelope for stepwise-constant resource allocations. In *CP*, pages 139–154, 2002.

16 T. Niemueller, G. Lakemeyer, and A. Ferrein. The robocup logistics league as a benchmark for planning in robotics. *Planning and Robotics (PlanRob-15)*, page 63, 2015.

17 N. Policella, S. Smith, A. Cesta, and A. Oddi. Generating robust schedules through temporal flexibility. In *ICAPS*, pages 209–218, 2004.

18 A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley & Sons, 1998.

19 A. Umbrico, A. Cesta, M. Cialdea Mayer, and A. Orlandini. Integrating resource management and timeline-based planning. In *ICAPS*, pages 264–272, 2018.

20 B. Williams and V. Gupta. Unifying model-based and reactive programming within a model-based executive. In *Workshop on Principles of Diagnosis*, 1999.

A Planning Semantics

In this section we reproduce the additional syntax and semantics of planning problems presented in [2], which provide additional background on definitions 1 and 2. For a more complete description linking these concepts we refer the reader to the source. We start by defining our planning language: we adopt the full PDDL 2.1 [8] with continuous change.

▶ **Definition 6.** *A **planning problem** $\mathcal{P}$ is a tuple $\langle P, V, A, I, G\rangle$, where P is a set of propositions; V is a set of real variables, called fluents; A is a set of durative and instantaneous actions; $I : P \cup V \to \{\top, \bot\} \cup \mathbb{R}$ is the total function describing the initial state of the predicates and the fluents. $G : P \cup V \to \{\top, \bot\} \cup \mathbb{R}$ is a (possibly partial) function indicating the goal condition. A durative action a is a tuple $\langle pre_a, eff_a, dur_a\rangle$, where pre_a is a set of conditions for the actions partitioned in three subsets $pre_{\vdash a}$, $pre_{\leftrightarrow a}$ and $pre_{\dashv a}$ of at-start, over-all and at-end conditions; eff_a is the set of action effects, partitioned in seven sets: $eff^{+}_{\vdash a}$ (positive starting effects), $eff^{-}_{\vdash a}$ (negative starting effects), $eff^{num}_{\vdash a}$ (numeric starting effects), $eff^{+}_{\dashv a}$ (positive ending effects), $eff^{-}_{\dashv a}$ (negative ending effects), $eff^{num}_{\dashv a}$ (numeric ending effects) and $eff^{num}_{\leftrightarrow a}$ (continuous numeric effects); and dur_a is a set of duration constraints. An instantaneous action a is a tuple $\langle pre_a, eff_a\rangle$, where pre_a is a set of pre-conditions and eff_a is the set of action effects, partitioned in eff^{+}_{a} (positive effects), eff^{-}_{a} (negative effects) and eff^{num}_{a} (numeric effects).*

In the usual PDDL 2.1 setting, a plan is defined as a set of actions associated with a starting time and a duration. We define this kind of plans as time-triggered plans.

▶ **Definition 7.** *A **time-triggered plan** π for a planning problem $\mathcal{P} \doteq \langle P, V, A, I, G\rangle$ is a set of tuples $\langle t, a, d\rangle$, with $t \in \mathbb{R}_{\geq 0}$, $a \in A$ and $d \in \mathbb{R}_{>0}$ iff a is a durative action.*

For the sake of brevity, we omit the formal definition of validity for such a plan, which can be found in [8]. Here, it suffices to remind oneself that a plan is valid if by simulating the system controlled by the plan, all the prescribed actions are applicable (all their conditions are satisfied at the time the action is executed) and the goal is reached after the last action terminates.

We define an STN plan as a constraint network of time points indicating the starting or the ending of actions. Note that the STN plan contains all the information of, and is strictly more general than a time-triggered plan. Moreover, that it is not necessary to first find a time-triggered plan in order to generate an STN plan.

▶ **Definition 8.** *An **STN plan** π for $\mathcal{P} \doteq \langle P, V, A, I, G\rangle$ is a tuple $\langle T, C\rangle$, where T is the set of time points $\{z\} \cup \{t^s_{da}, t^e_{da} \mid da \text{ is a durative action instance}\} \cup \{t_a \mid a \text{ is an instantaneous action instance}\}$ and C is a set of constraints in the form $t_i - t_j \leq b$ with $t_i, t_j \in T$, and $b \in \mathbb{R}$.*

Finally, we can define the validity of an STN plan by considering the set of all possible time-triggered plans that are compatible with the STN specification. If all such plans are valid, we say that the STN plan is valid.

▶ **Definition 9.** *Given an STN plan $\pi \doteq \langle T, C\rangle$ and an assignment $\mu : T \to \mathbb{R}$ s.t. $\mu(z) = 0$, the **induced time-triggered plan** by μ is the time-triggered plan $tt(\mu) \doteq \{\langle \mu(t^s_{da}), da, \mu(t^e_{da}) - \mu(t^s_{da})\rangle \mid da \text{ is a durative action}\} \cup \{\langle \mu(t_a), a, 0\rangle \mid a \text{ is an instantaneous action}\}$.*

▶ **Definition 10.** *An STN plan $\pi \doteq \langle T, C\rangle$ for $\mathcal{P}$ is **valid** if for each assignment $\mu : T \to \mathbb{R}$ s.t. $\mu(z) = 0$ and for all $t_i - t_j \leq b \in C$ $\mu(t_i) - \mu(t_j) \leq b$, the time-triggered plan $tt(\mu)$ is valid for $\mathcal{P}$.*

Achieving a Sequenced, Relational Query Language with Log-Segmented Timestamps

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

M. A. Manazir Ahsan
Department of Computer Science, Utah State University, Logan, UT, USA

Abstract

In a period-timestamped, relational temporal database, each tuple is timestamped with a period. The timestamp records when the tuple is "alive" in some temporal dimension. *Sequenced semantics* is a special semantics for evaluating a query in a temporal database. The semantics stipulates that the query must, in effect, be evaluated simultaneously in each time instant using the tuples alive at that instant. Previous research has proposed changes to the query evaluation engine to support sequenced semantics. In this paper we show how to achieve sequenced semantics without modifying a query evaluation engine. Our technique has two pillars. First we use log-segmented timestamps to record a tuple's lifetime. A log-segmented timestamp divides the time-line into segments of known length. Any temporal period can be represented by a small number of such segments. Second, by taking advantage of the properties of log-segmented timestamps, we translate a sequenced relational algebra query to a non-temporal relational algebra query, using the operations already present in an unmodified, non-temporal query evaluation engine. The primary contribution of this paper is how to implement sequenced semantics using log-segmented timestamped tuples in a generic DBMS, which, to the best of our knowledge, has not been previously shown.

2012 ACM Subject Classification Information systems → Temporal data; Information systems → Relational database query languages

Keywords and phrases Temporal databases, sequenced semantics, query evaluation, relational algebra

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.14

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

A tuple-timestamped, temporal relational database is a relational database in which each tuple is annotated with a period timestamp, that is, a period of time from some start time to some end time. The timestamp is *metadata* about the tuple; it records when the data was "live" in some temporal dimension.

Temporal relational database management systems (TDMBSs) provide special handling for time metadata in queries. For instance, the *timeslice* operation retrieves the data that is alive at a specified time. TDBMSs typically support a wide range of temporal query operations but the most important is arguably *sequenced semantics* [2]. Informally, sequenced semantics states that the meaning of a sequenced query is that it is equivalent to the (non-temporal) query applied to every snapshot of the data, effectively sequenced semantics is akin to running the query simultaneously in every snapshot in the data's history. We previously showed that sequenced semantics sequenced semantics can be leveraged to support other kinds of semantics [11], e.g., nonsequenced semantics [3].

```
SELECT dept
FROM storesGoldCoast
WHERE dept NOT IN (SELECT dept FROM storesRobina)
```

Figure 1 Query to compute the difference between two tables.

Data	*Metadata*
Dept	**Time**
Shoe	[1,11]

(a) Relation `storesGoldCoast`

Data	*Metadata*
Dept	**Time**
Shoe	[2,3]
Shoe	[5,6]

(b) Relation `storesRobina`

Data	*Metadata*
Dept	**Time**
Shoe	[1,1]
Shoe	[4,4]
Shoe	[7,11]

(c) Result of sequenced evaluation of query in Figure 1.

Figure 2 Example relations.

The history of data can span many instants so it is infeasible to actually run a query on each and every snapshot. To support sequenced semantics a TDBMS must evaluate a sequenced query in some other way. Generally sequenced semantics is implemented by modifying the query evaluation engine c.f., [7]. Previously it was thought not possible to perform sequenced queries on an unaltered relational database management system (RDBMS), e.g., using an unaltered installation of MySQL or Postgres.

To illustrate what makes sequenced query evaluation challenging, consider the SQL query given in Figure 1 which computes the difference between the `dept` attribute in two relations, `storesGoldCoast` and `storesRobina` shown in Figure 2. The query evaluates when there were departments in a `storesGoldCoast` relation and no departments with the same name in the `storesRobina` relation (Robina is a small area within the Gold Coast in Australia). The result of the *sequenced evaluation* of the query is shown in Figure 2(c). What makes the computation complicated is that no single pairing of tuples from the relations computes each tuple in the result, i.e., it cannot be produced by a Cartesian product of the two relations. For instance, we can only figure out the timestamp of the second tuple in the result `[4,4]` by determining that `[2,3]` and `[5,6]` leaves a gap of `[4,4]` within `[1,11]` and that there is no other tuple in `storesRobina` that overlaps `[4,4]`. When moving to the extended relational algebra or SQL, (sequenced) temporal grouping and aggregation, and some subqueries, e.g., `NOT IN` subqueries, are similarly problematic.

In this paper we show how it is possible to translate a sequenced query into a non-temporal query. The translation uses a kind of timestamp that we describe in Section 3. We focus on relational algebra as an example of a complete query language that is widely-known, easy to describe, has a procedural semantics, and provides the basic operations to implement an SQL query evaluation engine. We give a translation of sequenced relational algebra to non-temporal relational algebra in Section 4.

2 Related Work

This paper extends previous research in the area of temporal query languages, There are many temporal extensions of query languages, c.f., [4, 8, 13, 16, 17]. These extensions are designed to add to, rather than change or modify, the prior syntax and semantics of a language. The extensions have been broadly characterized in various ways. *Sequenced* vs. *nonsequenced* distinguishes extensions, in part, by whether the time metadata is manipulated implicitly or explicitly. Temporal languages have also been characterized as *abstract* vs. *concrete* based on whether their syntax and semantics depends on a specific representation of the time metadata [5].

Two implementation approaches are common for SQL-like temporal query languages. A *stratum*-approach adds a source-to-source translation layer to translate a query in a temporal extension into an equivalent query in the original, non-extended language [19, 20]. Some constructs prove not possible to translate using period timestamps, e.g., sequenced outer join, so the only feasible approach is to extend the DBMS itself [7]. In general, sequenced semantics cannot be directly supported in standard SQL because some of the needed operations are not part of SQL, hence the second strategy extends the DBMS to support additional operations for sequenced semantics. A related approach is to translate to a non-standard variant of SQL [10]. To the best of our knowledge this is the first paper to implement sequenced semantics by translating to standard relational algebra. The translation supports implementation in garden-variety, unaltered relational DBMSs, e.g., MariaDB, Postgres, etc.

Finally, hierarchical partitioning of intervals into smaller segments, similar to log segments, for the purposes of indexing has been explored recently [6]. Our research [9] predates this effort and supports indexing by B-tree indexes.

3 Log-segmented Timestamps

Most temporal database research and implementation uses period timestamps to annotate data with temporal metadata [15]. Period timestamping appends a timestamp to each data item to represent its lifetime. Research has also explored *coalesced* period timestamping in which value-equivalent tuples must have maximally disjoint periods [1]. Another way to represent a coalesced timestamp is with a temporal element, which is a set of disjoint periods [14]. Since a temporal element is a set, it can only be directly stored in a non-1NF data model. A variation of tuple-timestamped models is *attribute timestamping* where timestamps are appended to each attribute in a tuple rather than to the entire tuple [18].

Period timestamps are a poor fit for architectures which need to partition large data sets into smaller shards to process, e.g., mapreduce architectures. Consider, for instance a join operation. Hash-join is usually a good strategy for mapreduce. The mapreduce hash-join maps data items that have the same join values to a common shard, and then joins the items in the shard. The strategy is efficient since it ensures that only data items that actually will join are put into a shard. A *sequenced (temporal)* join adds a further condition that two data items join only on the times at which they are both alive. For period timestamps this is computed as the *temporal intersection* of the timestamps. If the intersection is empty, the items do not join since they do not coexist at any point in time. The problem is that periods cannot be directly mapped to shards in a way that ensures that the items within a shard temporally intersect. Consider the periods `[1,2]`, `[8,9]`, and `[0,10]`. `[1,2]` and `[8,9]` should be placed in different shards since they do not intersect, and hence, never represent data that coexists. But `[0,10]` intersects both, it has to be placed into both. Since a period of size n has n^2 sub-periods that could intersect, every period potentially needs to belong to many shards.

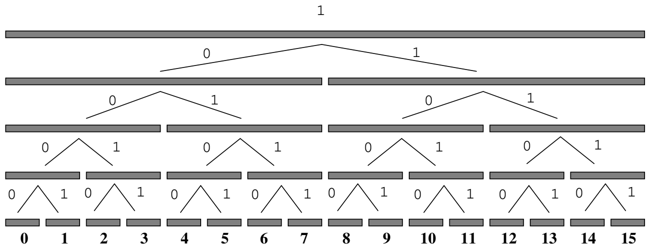


Figure 3 Log segments on a time-line.

Table 1 Some example labels for the time-line 0. . .15.

Label	**Period**	t_x	t_y
1	0 – 15	0	$15 = 0 + (2^4 - 1)$
10	0 – 7	$0 = 0 * 2^4$	$8 = 0 + (2^3 - 1)$
110	8 – 11	$8 = 1 * 2^3$	$11 = 8 + (2^2 - 1)$
1101	10 – 11	$10 = 1 * 2^3 + 1 * 2^1$	$11 = 10 + (2^1 - 1)$
10011	3 – 3	$3 = 1 * 2^1 + 1 * 2^0$	$3 = 3 + (2^0 - 1)$

To address this challenge we developed a *log-segmented timestamp* [9]. The timestamp uses a labelling scheme for pre-determined periods on a time-line. A label is a binary number that has the following meaning.

▶ **Definition 1** (Log-segment Label). *Let a (discrete) time-line consist of the times* $t_0, \ldots, t_n$*, where* $n = 2^k - 1$*. Note that* n *can be represented using a binary number of length* k *with each digit set to 1. A label is a binary number,* $b_0 \ldots b_j$*, and* b_0 *is always 1. The label* $1b_1 \ldots b_j$*,* $j \leq k$*, represents the time period* t_x *to* t_y *where* $t_x = b_1 2^{k-1} + b_2 2^{k-2} + \ldots b_j 2^{k-j}$ *and* $t_y = t_x + (2^{k-j} - 1)$*.*

The log segments for a time-line from `0` to `15` are depicted in Figure 3. The chronons in the time-line are numbered at the bottom of the figure. Each gray rectangle in the figure is a segment. A label for a segment is the concatenation of `1`'s and `0`'s along the path from the root to a segment. Some example labels are shown in Table 1. Note that only $2n - 1$ of the n^2 possible periods in the timeline are labelled.

A *log-segmented timestamp* is the minimal set of segments that spans a given period. For example, the log-segmented timestamp representing the period `[3,11]` is `{10011, 101, 110}` (naming the periods `{[3,3], [4,7], [8,11]}`, respectively). The log-segmented timestamps for the times in the relations in Figure 2 a) and b) is graphically depicted in Figure 5. Figure 4 shows the log-segmented tuples for the relations in Figure 2.

Log-segmented timestamps have the following properties.

- Comprehensive – A time-line of size n has at most $2n - 1$ labels. Each label will have a maximum length of $1 + \lceil \log_2(n) \rceil$ bits. So a label of 64 bits (the size of a `long long` scalar in C++) can represent a time-line of $2^{63} - 1$ time values, which encompasses a time-line longer than current estimates of the lifetime of the universe to the granularity of microseconds [12].
- Compact – The maximum number of segments in a log-segmented timestamp for a period, $[t_x, t_y]$, is $\lceil \log_2((1 + t_y) - t_x) \rceil$. So assuming 64 bit labels, a log-segmented timestamp has at most 64 labels.

Data	*Metadata*
Dept	**Time**
Shoe	10001
Shoe	1001
Shoe	101
Shoe	110

(a) Relation `storesGoldCoast`

Data	*Metadata*
Dept	**Time**
Shoe	1001
Shoe	10101
Shoe	10110

(b) Relation `storesRobina`

Figure 4 Example log-segmented relations.

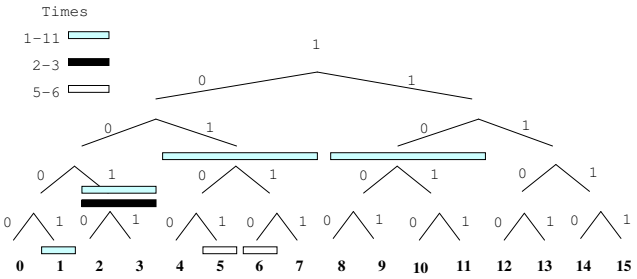


Figure 5 Log segments for the times in the relations in Figure 2 a) and b).

- Efficient for temporal predicates – Predicates in Allen's algebra can be quickly computed. For example for overlaps, given two labels, L_1 and L_2,

$$\texttt{overlaps}(L_1, L_2) = \begin{cases} L_1 & \text{if } L_2 \text{ is a prefix of } L_1 \\ L_2 & \text{if } L_1 \text{ is a prefix of } L_2 \\ \textit{nothing} & \text{otherwise} \end{cases}$$

- Groups – In temporal aggregation a *membership-constant* period is a period of time when some data items, and only, those data items, belong to a group. In a log-segmented timestamp, a label and all prefixes/suffixes of it describe a membership-constant period. So, assuming a timestamp of length 4 the membership-constant period `1001` includes data timestamped with any prefix. Said differently, if we want to compute an aggregate for the period `1001`, we use the data timestamped with `1001`, `100`, `10`, and `1`. So for a time-line of size n there are $\lceil \log_2(n) \rceil$ segments for a membership-constant period.

4 Relational Algebra

In this section we describe a complete set of relational algebra operators for sequenced semantics with log-segmented timestamps. The algebra is defined in terms of non-temporal relational algebraic operators.

4.1 Sequenced Projection

Log-segmented, sequenced projection, $\pi^{\mathcal{T}}$, for some set of attributes A on relation r is defined as follows.

$$\pi^{\mathcal{T}}_{A}(r) = \Phi(\pi_{\bar{A},r.T}(r))$$

Φ is the sequenced duplicate elimination operator, which is needed because projection in relational algebra produces a set of tuples, unlike SQL where the underlying model is a bag of tuples. Sequenced duplicate elimination is simple to define for log-segments since for any pair

Data		*Metadata*
Name	**Dept**	**Time**
Joe	Shoe	10
Fred	Shoe	1001
Jennifer	Shoe	101

Figure 6 Example `Employee` relation.

Data		*Metadata*
Dept	**Floor**	**Time**
Shoe	2	10
Shoe	2	111
Photo	1	101

Figure 7 Example `Departments` relation.

of value-equivalent tuples, t and v, if t's timestamp is temporally during v's timestamp, then t can be removed because it is a duplicate. The sequenced duplicate elimination operator is defined below, where ρ is the relation renaming operator (to give a copy of a relation a unique name), $D(t_1, t_2)$ is the timestamp *during* predicate, $r.T$ ($s.T$) is the timestamp for a tuple in relation r (s), $r.V$ ($s.V$) is the list of non-temporal attributes in r (s), and $\bowtie_{r.V=s.V}$ is a value-equivalent equi-join, i.e., the timestamps are ignored in the join, only the non-temporal values are used.

$$\Phi(r) = r - \pi_{r.V,r.T}(\sigma_{D(r.T,s.T)}(r \bowtie_{r.V=s.V} \rho_s(r)))$$

As an example, consider the relation shown in Figure 6 and the query

$$\pi^{\mathcal{T}}_{\texttt{Dept}}(\texttt{Employees}).$$

First we project the `Dept` attribute, as well as the timestamp metadata yielding a relation with three tuples as shown in Figure 9. Next we eliminate sequenced duplicates, yielding the result in Figure 8. The sequenced duplicate elimination removes the second and third tuples because they are during the first tuple's timestamp and are value-equivalent to the first tuple.

4.2 Sequenced Selection (Restriction)

The next operation is log-segmented, sequenced selection, where P is a predicate for deciding if a tuple is in the result relation.

$$\sigma^{\mathcal{T}}_P(r) = \sigma_P(r)$$

Sequenced selection is straightforward it is the same as non-temporal selection; duplicate elimination is not needed since the relation being selected does not contain duplicates, hence the result of a selection cannot have duplicates.

4.3 Sequenced Cartesian Product

Sequenced Cartesian product similarly cannot produce duplicates, but result tuples only exist at the time given by the intersection of two tuples. In the definition, $O(r.T, s.T)$ is the overlaps temporal predicate, $I(r.T, s.T)$ is the temporal intersection constructor, and $r.V$ ($s.V$) is the list of non-temporal attributes in tuple r (s). Note that the projection operator in the definition is a generalized projection since it constructs a timestamp value not present in the operand relations.

$$r \times^{\mathcal{T}} s = \pi_{r.V,s.V,I(r.T,s.T)}(\sigma_{O(r.T,s.T)}(r \times s))$$

As an example if we take the Cartesian product of the relation in Figure 6 with itself, we end up with the relation in Figure 10.

Data	*Metadata*
Dept	**Time**
Shoe	10

Figure 8 After sequenced duplicate are eliminated.

Data	*Metadata*
Dept	**Time**
Shoe	10
Shoe	1001
Shoe	101

Figure 9 The (non-temporal) projection of the `Dept` attribute, need to eliminate sequenced duplicates.

Data		*Metadata*		
Name	**Dept**	**Name**	**Dept**	**Time**
Joe	Shoe	Joe	Shoe	10
Fred	Shoe	Joe	Shoe	1001
Jennifer	Shoe	Joe	Shoe	101
Joe	Shoe	Fred	Shoe	1001
Fred	Shoe	Fred	Shoe	1001
Joe	Shoe	Jennifer	Shoe	101
Jennifer	Shoe	Jennifer	Shoe	101

Figure 10 Example sequenced Cartesian Produce of the `Employee` relation with itself.

4.4 Sequenced Union

Log-segmented, sequenced union adds duplicate elimination to the result of a non-temporal union.

$$r \cup^{\mathcal{T}} s = \Phi(r \cup s)$$

As an example, consider the union of the `Departments` relation shown in Figure 7 with the `Employees` relation in Figure 6 (or rather the projection of each on the `Dept` attribute) as follows.

$$\pi^{\mathcal{T}}_{\texttt{Dept}}(\texttt{Departments}) \cup^{\mathcal{T}} \pi^{\mathcal{T}}_{\texttt{Dept}}(\texttt{Employees})$$

The projection of the `Employees` relation is in Figure 8 and the projection of the `Departments` relation is shown in Figure 11. The result of the union is shown in Figure 12.

4.5 Sequenced Intersection

Sequenced intersection can be expressed using sequenced Cartesian product, selection, and sequenced projection.

Data	*Metadata*
Dept	**Time**
Shoe	10
Shoe	111
Photo	101

Figure 11 Sequenced projection of the `Departments` relation.

Data	*Metadata*
Dept	**Time**
Shoe	10
Shoe	111
Photo	101

Figure 12 Example of a union operation.

$$r \cap^{\mathcal{T}} s = \pi^{\mathcal{T}}_{r.V}(\sigma_{r.V=s.V)}(r \times^{\mathcal{T}} s))$$

Intersection can be computed by first taking the sequenced Cartesian product. From this, for all tuples that have value-equivalent pairs in the underlying relation, it takes the sequenced projection of r's attributes. As an example, consider the intersection of the `Employee` relation with itself. First we take the Cartesian product as shown in Figure 10. Next the selection restricts the result to the first, fifth, and seventh tuples since these tuples have the same departments and employee names. Finally the sequenced projection produces the result shown in Figure 6.

4.6 Sequenced Difference

The problem of sequenced, relational difference was described in Section 1. Log-segmented, sequenced relational difference is somewhat complicated. The operation is defined below assuming $C(t_1, t_2)$ is the temporal contains predicate, $O(t_1, t_2)$ is the temporal overlaps predicate, and $E(t_1, t_2)$ is the temporal equals predicate.

$$r -^{\mathcal{T}} s = \Phi(r_c \cup (r_d - (r_d \ltimes_{r.V=s.V \ \wedge \ (C(r_d.T,s.T) \ \vee \ E(r_d.T,s.T))} s)))$$

where

$r_c = r - (r \ltimes_{r.V=s.V \ \wedge \ O(r.T,s.T)} s)$,

$r_d = \pi_{r.V,\mathbb{P}.T_3}((r \bowtie_{r.V=s.V \ \wedge \ C(s.T,r.T)} s) \bowtie_{r.T=\mathbb{P}.T_1 \wedge s.T=\mathbb{P}.T_2} \mathbb{P}))$, and

$\mathbb{P}(T_1, T_2, T_3)$ is the pre-computed log-segmented difference relation.

First, r_c, is the set of tuples that have no value-equivalent match in s or if they have a value-equivalent match do not overlap in time with any tuple in s. Second, r_d is the tuples in r that have a value-equivalent match in s and a lifetime that is during (excluding equals) the lifetime of the tuple in s, which we will call the *during tuples*. The challenge in computing the during tuples is determining potentially *when* they exist since the time is usually not the time of either the tuple in r or in s, which is why relation $\mathbb{P}$ is needed. $\mathbb{P}$ is the *log-segmented difference* relation. It computes the log-segments, attribute T_3, in the difference between a pair of times T_1 and T_2 and is defined as follows, assuming S is the domain of log segments, $C(t_1, t_2)$ is the temporal contains predicate, and $O(t_1, t_2)$ is the temporal overlaps predicate.

$$\mathbb{P}(T_1, T_2, T_3) = \Phi(\{(t_1, t_2, t_3) \mid t_1, t_2, t_3 \in S \ \wedge \ C(t_1, t_3) \ \wedge \ C(t_1, t_2) \ \wedge \ \neg O(t_2, t_3)\})$$

Figure 13 shows some of the tuples in $\mathbb{P}$. For instance, the difference between `10` and `10001` yields the log-segments in the set (in different tuples) {`101`, `1001`, `10000`}. Observe that in Figure 3 these log segments are a set of log segments that together with `10001` span `10`, and are *coalesced*, i.e., no log segment in the set is contained within some log segment, x, such that x is not in the set and x is contained by `10`.

As an example suppose that we take the difference between the `Employees` relation in Figure 6 and the relation in Figure 14. The result is shown in Figure 16. First `Fred` is in the result unchanged from the `Employee` relation since the time in his tuple, `1001`, does not overlap time `11`. That is, `Fred`'s tuple is in r_c. Second, `Jennifer` is not in the result since her tuple's time, `101`, is contained within the time of her tuple in the difference relation, `10`. `Jennifer`'s tuple is not in r_d (or r_c). Finally, consider `Joe`. His tuple has a value-equivalent match that has a lifetime, `10`, which contains his lifetimes in s, `10001` and `101`. `10 - 10001` is {`101`,`1001`,`10000`} while `10 - 101` yields {`100`}. So r_d is the relation shown in Figure 15. From this relation we remove any tuple that is value-equivalent and contains or is equal to a time in the difference relation (Figure 14. The first (`101` is equal to `101`), third (`1001` is equal to `1001`), and fourth tuples (`100` is equal to `100`) are removed yielding only the third tuple to be added to the final result.

T_1	T_2	T_3
	...	
10	101	100
10	1001	101
10	1001	1000
10	10001	101
10	10001	1001
10	10001	10000
	...	

Figure 13 Some tuples in $\mathbb{P}$.

	Data	*Metadata*
Name	**Dept**	**Time**
Joe	Shoe	10000
Fred	Shoe	11
Jennifer	Shoe	10

Figure 14 `Employee` difference relation.

	Data	*Metadata*
Name	**Dept**	**Time**
Joe	Shoe	101
Joe	Shoe	1001
Joe	Shoe	10000
Joe	Shoe	100

Figure 15 The during tuples in computing the difference.

	Data	*Metadata*
Name	**Dept**	**Time**
Joe	Shoe	10001
Fred	Shoe	1001

Figure 16 Result of the sequenced difference of Figure 6 and Figure 14.

4.7 Sequenced Grouping and Aggregation

Sequenced grouping and aggregation is also possible with log segments, though the process is somewhat complicated. We first give an informal example of sequenced aggregation and group by, and then a formal definition.

Assume that we want to count the number of `Employees` per `Department` over time, i.e., a sequenced aggregation and grouping. Furthermore, assume that our relation has four tuples for the `Clothing` department timestamped with log-segments 1010, 1010, 101, and 1 as shown in Figure 17.

Step 1: Determine log segment fragments Long-lived tuples potentially span many temporal groups. For instance, in the relation in Figure 17, `Freya`'s tuple contains the lifetime of all the other tuples in the relation so should belong to each group, but also to groups not in the lifetimes of those tuples, e.g., `Freya` is present at time `11` while none of the other tuples are (they are all within `10`). So the goal of this step is to split the timestamps to determine coverage with respect to the other timestamps in the relation. We use temporal difference to split the lifetimes, that is for any lifetime that is contained in another, we take the difference. For instance, in our running example, `(Susan, Clothing, 1010)`

	Data	*Metadata*
Name	**Dept**	**Time**
Susan	Clothing	1010
Pedro	Clothing	1010
Malik	Clothing	101
Freya	Clothing	1

Figure 17 Example relation for grouping.

	Data	*Metadata*
Name	**Dept**	**Time**
Malik	Clothing	1011
Freya	Clothing	1011
Freya	Clothing	11
Freya	Clothing	100

Figure 18 Fragments of lifetimes.

Data		*Metadata*
Name	**Dept**	**Time**
Freya	Clothing	1010
Malik	Clothing	1010
Freya	Clothing	101

Figure 19 Long-lived tuple are potential group members.

	Data		*Metadata*
Count	**Name**	**Dept**	**Time**
4	Susan	Clothing	1010
4	Pedro	Clothing	1010
4	Freya	Clothing	1010
4	Malik	Clothing	1010
2	Malik	Clothing	1011
2	Freya	Clothing	1011
1	Freya	Clothing	100
2	Malik	Clothing	101
2	Freya	Clothing	101
1	Freya	Clothing	11
1	Freya	Clothing	1

Figure 20 Union of the original relation, Figure 18 and Figure 19 with the aggregate computed.

lifetime is contained in that of `(Malik, Clothing, 101)` so we take the difference of `101` and `1010` to get `1011` and so generate the tuple `(Malik, Clothing, 1011)`. We also do the other pairs, `1 - 101` yielding `(Freya, Clothing, 11)` and `(Freya, Clothing, 100)`, and the pair `1 - 1010` yielding `(Freya, Clothing, 1011)` and `(Freya, Clothing, 11)`. The result relation is shown in Figure 18.

Step 2: Add long-lived tuples to contained lifetime groups This step add long-lived tuples to the groups that have lifetimes that are contained within the lifetimes of the long-lived tuple. For instance, in the relation in Figure 17, `Freya`'s tuple contains the lifetime of all the other tuples in the relation so should belong to each group, e.g., `Freya` is present at time `101` and `1010`. The resulting relation is shown in Figure 19.

Step 3: Gather potential group members Form the union of the results of the original relation, Step 1, and Step 2. The result relation is shown in Figure 20 (the relation depicted has the computed aggregates as well, but those will be added in the next step).

Step 4: Group and aggregate Group and aggregate the result of Step 3, pre-pending the aggregate value (computed for the group) to each tuple. The result relation is shown in Figure 20.

Step 5: Remove containing lifetimes Since lifetimes were fragmented in Step 1 to represent smaller periods, this step removes duplicate counts. A duplicate count is for any tuple that has a lifetime that contains that of another tuple in the relation produced in Step 4. For instance, `(2, Mailik, Clothing, 101)` is a duplicate tuple since its lifetime contains the lifetime of another tuple `(4, Freya, Clothing, 1010)`. Hence it has already been counted and should be removed. The result of this step is shown in Figure 21, which is the sequenced count of `Employees` grouped by `Dept`.

The aggregation operator ${}_{\bar{G}}\mathcal{F}^{\mathcal{T}}_{\bar{A}}$, where $\bar{G}$ is a list of grouping attributes and $\bar{A}$ is a list of aggregate functions, is defined below.

$${}_{\bar{G}}\mathcal{F}^{\mathcal{T}}_{\bar{A}}(r) = r_5$$

where (note: relation r_i is produced by Step i)

$$r_1 = \pi_{r.V,\mathbb{P}.T_3}((r \bowtie_{C(r.T,s.T)\ \wedge\ r.\bar{G}=s.\bar{G}} \rho_s(r)) \bowtie_{r.T=\mathbb{P}.T_1 \wedge s.T=\mathbb{P}.T_2} \mathbb{P}),$$
$$r_2 = \pi_{r.V,s.T}(r \bowtie_{C(r.T,s.T)\ \wedge\ r.\bar{G}=s.\bar{G}} \rho_s(r)),$$
$$r_3 = r \cup r_1 \cup r_2, \text{ and}$$
$$r_4 =_{\bar{r}.G} \mathcal{F}_{\bar{A}}(r_3).$$
$$r_5 = r_4 - (r_4 \ltimes_{r.\bar{G}=s.\bar{G}\ \wedge\ C(r_4.T,s.T)} \rho_s(r_4))$$

	Data		*Metadata*
Count	**Name**	**Dept**	**Time**
4	Susan	Clothing	1010
4	Pedro	Clothing	1010
4	Freya	Clothing	1010
4	Malik	Clothing	1010
2	Malik	Clothing	1011
2	Freya	Clothing	1011
1	Freya	Clothing	100
1	Freya	Clothing	11

Figure 21 Sequenced count of `Employees` grouped by `Dept`.

4.8 Cost Analysis

The primary disadvantage of log-segmented relational algebra is cost since the log-segmented increases the size of the relations. Note however, that the size cost could be reduced by normalizing a log-segmented relation, that is, by splitting the data and metadata columns into separate tables, with a foreign key from the metadata table into the data table. In this analysis we do not assume such normalization.

Let relation r (s) be a period timestamped relation with N (M) tuples. Representing the relations using log segments increase the size of the relation by a factor of $f = \log_2(k)$ where k is the maximum time (assuming a time domain from 0 to k). Then the relational algebra operators have the following cost.

- Sequenced projection of r: The cost is $\mathcal{O}(fN)$ to project r and $\mathcal{O}((fN)^3)$ to perform duplicate elimination, so the cost is dominated by duplicate elimination.
- Sequenced selection of r: The cost is $\mathcal{O}(fN)$ to scan through the relation.
- Sequenced Cartesian product of r with s: The cost is $\mathcal{O}(f^2NM)$.
- Sequenced Union of r with s: The cost is $\mathcal{O}(fN) + \mathcal{O}(fM) + \wr((f(N+M))^3)$, so the cost is dominated by duplicate elimination.
- Sequenced Intersection of r with s: The cost is $\mathcal{O}((fN) * (fM))$ since the projection and selection can performed as the Cartesian product is computed.
- Sequenced Difference of r minus s: To compute the *during tuples* costs $\mathcal{O}(f^3NM)$ assuming that $\mathbb{P}$ can by dynamically computed, e.g., such as using a table function in Postgres. To compute r_c costs $\mathcal{O}(f^3N^2M)$. The union of r_c with the during tuples and performing the duplicate elimination costs $\mathcal{O}(f^9N^3M^2)$, so the duplicate elimination again dominates the cost.
- Sequenced Grouping and Aggregation: There are five steps. To compute r_1 costs $\mathcal{O}(f^3NM)$. Computing r_2 squares the cost of r_1 and, assuming linear-time union can be performed, the cost of r_3 is $\mathcal{O}((fN)^2)$, which is the maximum possible size of r_2 or r_3. We will assume computing the aggregate can be done in linear time, so the cost of r_5 is $\mathcal{O}((fN)^4)$

Note that the most frequent query operations are projection, selection, and Cartesian product. The cost of selection and Cartesian product are the same as their non-temporal counterparts (except for the increased size of the relation). But unlike temporal periods, log segments can be indexed using a non-temporal index, e.g., a B^+-tree, so there are likely significant query optimization opportunities for sequenced queries using standard SQL query optimization techniques involving indexes. Only projection is significantly more expensive,

but the cost is largely due to duplicate elimination, which can be thought of as optional in an SQL-based DBMS, which allows duplicates in the data model. The cost of the other operations (except intersection which is the same as the non-temporal cost) is much higher than their non-temporal counterpart (which do not support sequenced semantics, with the additional functionality comes increased cost). But, overall sequenced queries can be supported in a vanilla SQL-based DBMS and we suspect that query optimization combined with standard indexes can achieve reasonable run-time efficiency.

5 Conclusion and Future Work

The primary contribution of this paper is to show how sequenced semantics can be implemented for a relational query language using the non-temporal form of the language. This demonstration means that it is possible to implement sequenced semantics when evaluating queries in a relational DBMS such as MariaDB without having to make any changes to the DBMS.

In this paper we presented sequenced relational algebra by defining its operations entirely in terms of standard relational algebra, lacking any temporal semantics or constructs. The key to the translation is to interpret timestamps in a different way. Rather than taking the standard approach of using period timestamps we chose to timestamp using *log segments*. The log segments are an *a priori* dividing of the time-line into segments such that the segments cover the time-line and form a hierarchy in which smaller segments group into larger segments. The labels on the segments can be used to efficiently and easily determine temporal relationships such as overlaps or contains. We showed how the segments are used in various operations such as sequenced aggregation and grouping.

Future work is focused on implementation. We are currently implementing a sequenced SQL to SQL translator using Postgres. An open question is the impact of the translation on query optimization. That is, can the query optimizer take advantage of indexes for the log segments in the translated queries? We are also investigating the benefits and costs of normalized representation (factoring the metadata into separate tables). We have not yet begin to look at other issues such as implementation of sequenced constraints using log segments, recursive queries, or application to other query languages such as sequenced GraphQL.

References

1 Michael H. Böhlen. Temporal coalescing. In *Encyclopedia of Database Systems*, pages 2932–2936. Springer, 2009. doi:10.1007/978-0-387-39940-9_388.

2 Michael H. Böhlen and Christian S. Jensen. Sequenced semantics. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_1053.

3 Michael H. Böhlen, Christian S. Jensen, and Richard T. Snodgrass. Nonsequenced semantics. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_1052.

4 Cindy Xinmin Chen and Carlo Zaniolo. Sqlst: A spatio-temporal data model and query language. In *ER*, pages 96–111, 2000. doi:10.1007/3-540-45393-8_8.

5 Jan Chomicki and David Toman. Abstract versus concrete temporal query languages. In *Encyclopedia of Database Systems*, pages 1–6. Springer, 2009. doi:10.1007/978-0-387-39940-9_1559.

6 George Christodoulou, Panagiotis Bouros, and Nikos Mamoulis. Hint: A hierarchical index for intervals in main memory, 2021. arXiv:2104.10939.

7 Anton Dignös, Michael H. Böhlen, and Johann Gamper. Temporal Alignment. In *SIGMOD*, pages 433–444, 2012. doi:10.1145/2213836.2213886.

8 Curtis E. Dyreson. Observing Transaction-Time Semantics with TTXPath. In *WISE (1)*, pages 193–202, 2001. doi:10.1109/WISE.2001.996480.

9 Curtis E. Dyreson. Using couchdb to compute temporal aggregates. In *18th IEEE International Conference on High Performance Computing and Communications; 14th IEEE International Conference on Smart City; 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016, Sydney, Australia, December 12-14, 2016*, pages 1131–1138. IEEE Computer Society, 2016. doi:10.1109/HPCC-SmartCity-DSS.2016.0159.

10 Curtis E. Dyreson and Venkata A. Rani. Translating temporal SQL to nested SQL. In Curtis E. Dyreson, Michael R. Hansen, and Luke Hunsberger, editors, *23rd International Symposium on Temporal Representation and Reasoning, TIME 2016, Kongens Lyngby, Denmark, October 17-19, 2016*, pages 157–166. IEEE Computer Society, 2016. doi:10.1109/TIME.2016.24.

11 Curtis E. Dyreson, Venkata A. Rani, and Amani Shatnawi. Unifying sequenced and non-sequenced semantics. In Fabio Grandi, Martin Lange, and Alessio Lomuscio, editors, *22nd International Symposium on Temporal Representation and Reasoning, TIME 2015, Kassel, Germany, September 23-25, 2015*, pages 38–46. IEEE Computer Society, 2015. doi:10.1109/TIME.2015.22.

12 Curtis E. Dyreson and Richard T. Snodgrass. Timestamp semantics and representation. *Inf. Syst.*, 18(3):143–166, 1993. doi:10.1016/0306-4379(93)90034-X.

13 Fabio Grandi. T-SPARQL: A TSQL2-like Temporal Query Language for RDF. In *ADBIS*, pages 21–30, 2010. URL: http://ceur-ws.org/Vol-639/021-grandi.pdf.

14 C. S. Jensen and C. E. Dyreson (editors). A Consensus Glossary of Temporal Database Concepts - February 1998 Version. In *Temporal Databases: Research and Practice, Lecture Notes in Computer Science 1399*, pages 367–405. Springer-Verlag, 1998.

15 R. T. Snodgrass. Introduction to TSQL2. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, chapter 2, pages 19–31. Kluwer Academic Publishers, 1995.

16 Richard T. Snodgrass. The Temporal Query Language TQuel. *ACM Trans. Database Syst.*, 12(2):247–298, 1987. doi:10.1145/22952.22956.

17 Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

18 A. U. Tansel. Modelling temporal data. *Information and Software Technology*, 32(8):514–520, October 1990.

19 Kristian Torp, Christian S. Jensen, and Michael H. Böhlen. Layered temporal DBMS: concepts and techniques. In *DASFAA*, pages 371–380, 1997.

20 Kristian Torp, Christian S. Jensen, and Richard T. Snodgrass. Stratum Approaches to Temporal DBMS Implementation. In *IDEAS*, pages 4–13, 1998. doi:10.1109/IDEAS.1998.694346.

Olisipo: A Probabilistic Approach to the Adaptable Execution of Deterministic Temporal Plans

Tomás Ribeiro
Institute for Systems and Robotics, Instituto Superior Tecnico, Lisbon, Portugal

Oscar Lima
DFKI German Research Center for Artificial Intelligence, Saabrücken, Germany

Michael Cashmore
University of Strathclyde, Glasgow, UK

Andrea Micheli
Fondazione Bruno Kessler, Trento, Italy

Rodrigo Ventura
Institute for Systems and Robotics, Instituto Superior Tecnico, Lisbon, Portugal

Abstract

The robust execution of a temporal plan in a perturbed environment is a problem that remains to be solved. Perturbed environments, such as the real world, are non-deterministic and filled with uncertainty. Hence, the execution of a temporal plan presents several challenges and the employed solution often consists of replanning when the execution fails. In this paper, we propose a novel algorithm, named Olisipo, which aims to maximise the probability of a successful execution of a temporal plan in perturbed environments. To achieve this, a probabilistic model is used in the execution of the plan, instead of in the building of the plan. This approach enables Olisipo to dynamically adapt the plan to changes in the environment. In addition to this, the execution of the plan is also adapted to the probability of successfully executing each action. Olisipo was compared to a simple dispatcher and it was shown that it consistently had a higher probability of successfully reaching a goal state in uncertain environments, performed fewer replans and also executed fewer actions. Hence, Olisipo offers a substantial improvement in performance for disturbed environments.

2012 ACM Subject Classification Computing methodologies → Robotic planning

Keywords and phrases Temporal Planning, Temporal Plan Execution, Robotics

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.15

Supplementary Material *Software (Source Code)*: `https://github.com/TomasRibeiro96/Olisipo-planner`; archived at `swh:1:dir:21c4dfd9815b8d086ab714a5c687224529ad90af`

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Automated temporal planning is the technology of choice when it comes to automatically controlling systems subject to temporal constraints such as deadlines. However, one of the key underlying assumptions of (classical) temporal planning is the perfect knowledge of a deterministic environment. Formalisms such as PDDL 2.1 [14] allow the modeling of planning problems where actions to be executed have known and deterministic effects on the system and on its environment. Moreover, the world is assumed to be static, meaning that when no action is executed by the agent, then no change is possible.

All these assumptions greatly simplify the reasoning needed to efficiently find a plan, but rarely apply to application scenarios. In robotics, for example, the environment where the machine operates is almost never static and there is uncertainty on the actual effect of the actions, due to interference of other actors (e.g. humans) or simply because some actions may fail due to lack of dexterity or because of mishaps.

In this paper, we want to tackle the problem of executing a temporal plan generated under the assumptions above in a system where these assumptions may not hold, exploiting probabilistic information to maximize the likelihood of achieving the original plan goal. To this end, we follow the execution schema proposed by [21] but we revise the on-line execution part. In particular, we propose an execution algorithm, called Olisipo, that exploits a probabilistically-sound reasoning on Dynamic Bayesian Networks (DBNs) to select the next action to be executed. The algorithm is able to cope with the need of skipping planned actions, of repeating actions, and of re-ordering the actions to achieve the original goal taking into account the observed state of the world. The former case can be useful to avoid operations that become useless (and may fail) due to the non-static nature of the environment: for example, consider a plan prescribing to open a door that was supposedly closed, while it is in fact open because a human forgot to close it. The second feature allows to repeat actions that might have failed to achieve their effects or whose effects have been canceled by the dynamic nature of the environment. Finally, re-ordering actions allows the handling of mutated temporal and precedence constraints.

We implemented the proposed approach in the ROSPlan [5] framework and we experimentally compare its merits against the original ROSPlan dispatcher, showing that the use of our approach is able to reduce the number of re-planning invocations and to increase the probability of successfully reach a goal state at the end of the execution.

1.1 Related Work

How to robustly execute and adapt plans at runtime is a central problem for the development of autonomous systems [18]. In fact, when a plan is being generated it is often impossible to foresee all the situations the controlled agent could encounter during execution; therefore, different approaches have been explored to tackle these issues. A common technique consists in relying on runtime monitoring to check that the conditions under which the plan was generated hold at runtime and if they do not a new plan is generated online; this approach is usually called replanning [13, 4]. Another direction is to model some of the uncertainties at planning time (most notably the action durations) as uncontrollable entities in the model and use planning algorithms that synthesize strategies to deal with such uncertainties [23, 7]. Finally, the approach we also pursue in this paper consists in generalizing a plan that was optimistically generated and allow the executor to adapt to unforeseen circumstances.

Probabilistic planning is a standard approach for planning with discrete uncertainty. An overview of approaches to probabilistic planning is provided in [17]. A common approach is to model the task as a Markov Decision Problem, which can handle the kind of exogenous and unforeseen events that we consider in this paper. Solutions to the MDPs typically formulate policies with finite horizon [2]. In contrast we rely on the efficiency and speed of deterministic search to find a solution that can then be generalized for an execution.

Some authors [1, 16, 12, 3] have focused on the problem of generalizing a temporal plan to make it more likely to succeed when executed. On the same line, some planners try to produce "flexible" plans, meaning that they reduce the commitments in the plan to a minimum to allow the exploitation of more degrees of freedom at execution time [15, 6]. Generally, a flexible temporal plan is represented as Simple Temporal Network (STN) [11], a Disjunctive

Temporal Network (DTN) [28] or as a Temporal Plan Network [20]; possibly with an explicit representation of the uncertainty [23]. The executor is then responsible for dispatching the prescribed events at the right time given the available observations [24, 9, 22]. However, one key assumption of these approaches is that the plan is never invalidated according to the original model, only valid re-scheduling is admitted.

In this paper, we take a radically different view: we want to exploit any PDDL 2.1 temporal planner and change the way in which the plan is dispatched by creating an action-selector that reasons over the uncertainties with the freedom to re-arrange, skip, and repeat the planned actions.

2 Algorithm

The OLISIPO algorithm is an extension of the algorithm presented in [21]. The flow from planning to execution of OLISIPO can be visualised in Figure 1. The approach is essentially composed of an offline and an online phase. The offline phase of OLISIPO remains unchanged from [21], while the online phase is completely reformulated.

In summary, the offline phase, performed before starting plan execution, is responsible for converting a totally-ordered plan $\prod^{tt}$, generated by an external planner, to an adaptable partially-ordered plan $\prod'$. The adaptable partially-ordered plan is passed on to the online phase. The offline generation of the partially-ordered plan allows for a more flexible execution, namely by skipping and reordering actions during plan execution.

PDDL 2.1 planners output is commonly represented as time-triggered plans (e.g. [8, 26]). A time-triggered plan is a set of tuples $\langle t, a, d\rangle$, where a is an action, t is the time at which the action should start execution and d is the prescribed duration. An adaptable partially-ordered plan is defined below, and a complete description can be found in [21].

▶ **Definition 1.** *An **Adaptable Partially-Ordered Plan** $\prod'$ is the graph $\langle N, C\rangle$, where each node $n \in N$ represents either the plan start, an instantaneous action, or the start or end of a durative action; and each edge $c \in C$ represents a temporal relation: $x < time(n_1) - time(n_2) < y$ for $n_1, n_2 \in N$ and $x, y \in R$. Each edge is labelled as either **causal**, **interference**, or **action duration**. Action duration edges express the temporal constraints between the start and end of durative actions. Causal edges express temporal relationships inferred from the causal support between actions. Similarly, interference edges express the temporal relationships inferred from the interference between actions.*

The online phase has two sub-phases: the Plan Generation Sub-Phase (PGen) and the Dynamic Plan Execution Sub-Phase (PExec). PGen is used to generate the totally-ordered plan Γ with the highest probability of a successful execution, taking into consideration the current state of the world, say S_0, and the partially-ordered plan $\prod'$. In our approach, each fact in S_0 is represented by a probability value referring to the degree of belief of that fact being present in the world. In addition to this, the first action of Γ is dispatched for execution. PExec concerns the dynamic and robust execution of the plan Γ. These 2 sub-phases are further explained in the next sections.

The online phase of OLISIPO controls the action deliberation depending on the observations made by the agent. As shown in Figure 1, the agent reads the state of the world S_0 and checks whether a totally-ordered plan Γ has already been determined or not. At the start, no Γ has been determined so the algorithm enters PGen. In PGen, the algorithm uses the *Total-Order Extractor* to build a set of totally-ordered plans. Then, the *Plan Selector* chooses the totally-ordered plan with the highest success probability and sets it as Γ, the

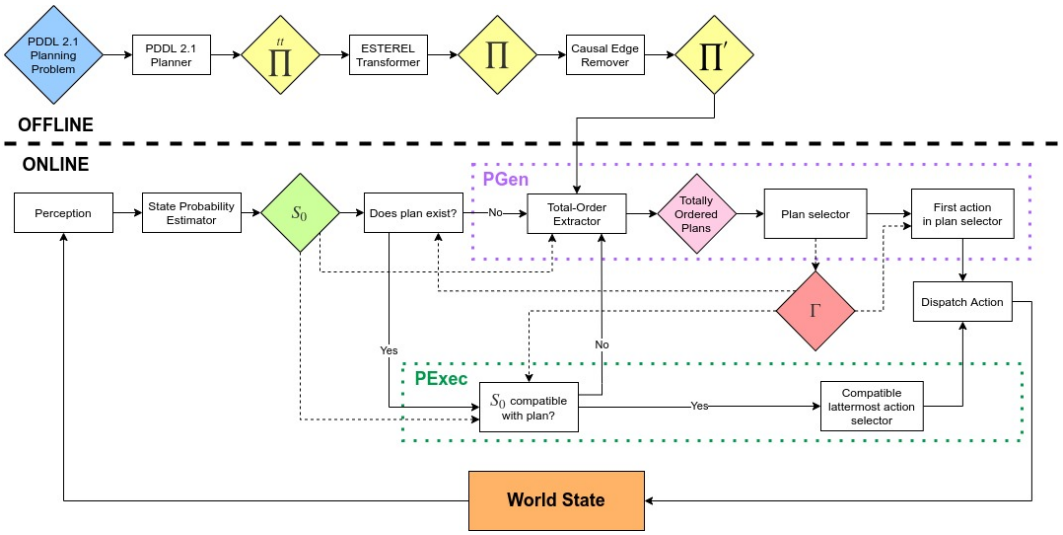


Figure 1 High-level overview of Olisipo's flow from planning to execution. The offline phase consists of converting a totally-ordered plan $\prod^{tt}$ into the partially-ordered plan $\prod'$. The online phase consists of two sub-phases, PGen and PExec. The PGen consists of finding the totally-ordered plan with the highest success probability, Γ, and dispatching its first action for execution. The PExec occurs when Γ has already been found and is compatible with the state of the world S_0. If Γ is compatible with S_0, then the compatible lattermost action of Γ is dispatched for execution.

totally-ordered plan to be used. Lastly, the first action of Γ is dispatched for execution. After executing this action, the system updates its world state representation and checks whether a totally-ordered plan has already been found. Since Γ has already been found, the algorithm advances to PExec. In this sub-phase, it first checks whether S_0 is compatible with Γ. If it is not compatible, then PGen is repeated and a new Γ is computed. If it is compatible, then the compatible lattermost action is dispatched for execution. The lattermost, as opposed to first, compatible action is chosen so that it becomes possible to skip actions that no longer contribute to achieving the goals. The agent will keep on reading the state of the world, and executing PGen and PExec, until the state of the world matches the goal.

2.1 Offline Phase: Generating an Adaptable Partially-Ordered Plan

The totally-ordered plan $\prod^{tt}$, generated by an external planner, is converted into a partially-ordered plan $\prod$ by creating a node for each instantaneous action and each durative action start and end. Then, temporal, durative and interference relations are generated between the nodes. Before starting plan execution, the causal support edges of $\prod$ are removed to give more flexibility in action selection, therefore resulting in the partially-ordered plan $\prod'$. A more detailed description of this process can be found on [21].

By first relaxing the problem to become deterministic, we are changing the problem's state-space and enabling the use of deterministic planners, which are able to scale for larger problem instances than undeterministic planners. This relaxation also means that we are moving the probability reasoning to the execution phase. Inspiration for this was taken from FF-Replan [30], which was the winner of the 2004 International Probabilistic Planning Competion (IPPC-04) and a top performer on IPPC-06. FF-Replan consisted of using a deterministic planner to solve a carefully constructed deterministic variant of the planning problem and replanning when an unexpected effect is observed. Similarly, we first solve a deterministic variant of the planning problem and obtain the plan $\prod^{tt}$, which is then converted to $\prod'$.

2.2 Online Phase

2.2.1 PGen: Building the Set of Totally-Ordered Plans and Calculating a Plan's Success Probability

The set of totally-ordered plans is built using a Branch and Bound search [19] through the nodes of $\prod'$, excluding the action start nodes which have already been dispatched but their respective action end nodes have not. This enables the repetition of an already executed action but prevents the algorithm from dispatching the action start of an action that has already started but has not finished yet.

Essentially, the search takes every node in $\prod'$ and orders them sequentially, hence building a totally-ordered plan. During search a Dynamic Bayesian Network (DBN) [10] is used to compute the joint probability of all actions being successful. When a solution is found, its probability is calculated as the joint probability of all actions being successful and of the facts of the goal being present in the solution. This solution probability is then used as reference value to prune the remaining search.

The search is pruned at a certain state S_i when one of the following conditions is verified:

- If the state S_i matches the goal;
- If the sequence of actions leading to S_i violates the temporal constraints;
- If the joint probability of the actions leading to S_i is lower than the probability of the best previously found solution (as Propositions 2 and 3 later explain, the joint probability of the actions is monotonically decreasing and is an upper bound to the solution probability);
- If there are no more applicable actions to S_i.

Our intention is to compute the success probability of a totally-ordered plan. We assume that it is possible to determine whether the execution of an action was successful. In addition to this, we say that a plan is successful if all the prescribed actions are applicable (i.e. the plan can be simulated) and if the goal condition is achieved in the final state of the plan.

A DBN, such as the one presented in Figure 2, is built to calculate a plan's success probability. This DBN contains two types of nodes: propositional nodes, which represent propositions at a certain time instant; and action nodes, which represent actions executed at a certain time instant. As it can be seen by the structure of the example DBN from Figure 2, the propositional nodes are divided into layers with one action node between each layer. Propositional nodes are propagated from layer to layer, representing their probability of spontaneously changing value with time. Action nodes have as parents and children their corresponding conditions and effects, respectively. We use the time and causal model of PDDL 2.1 [14], where durative actions are split into two instantaneous actions, corresponding to the start and end of a durative action. In the DBN of Figure 2, the action node *goto_waypoint_start(r0, w0, m2)* corresponds to the start of the durative action *goto_waypoint(r0, w0, m2)*, with its parents being the nodes corresponding to the *at start* conditions of the durative action, in the previous layer, and its children being the nodes corresponding to the *at start* effects, in the following layer. The node corresponding to the start of an action is a parent of the node corresponding to the end of an action. The nodes, in the layers between the action start and action end nodes, corresponding to the *over all* conditions of an action are parents of the action end node.

The uncertainty in the environment is represented in the Conditional Probability Distribution (CPD) of each node. Propositional nodes, from the first layer, do not have any parents so their CPD is represented by a single value, being 1 or 0 if they are *True* or *False* in the initial state, respectively. Propositional nodes with another propositional node as the only parent have a probability of spontaneously becoming *True* or *False*. Propositional

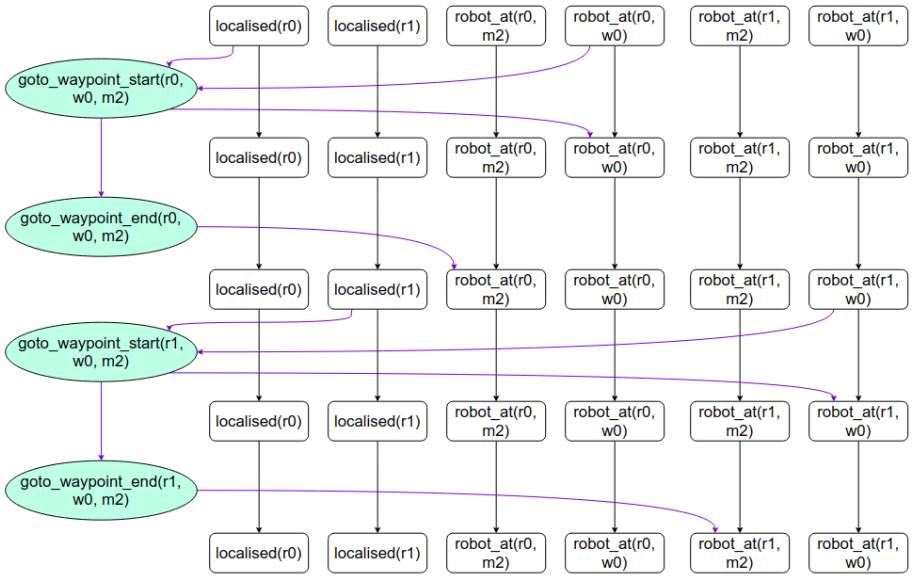


Figure 2 Example of the DBN's structure. Rectangular nodes symbolise proposition nodes and ellipsoidal nodes represent action nodes.

nodes with an action node and another propositional node as parents, have a probability of spontaneously becoming *True* or *False*, if the action node is *False*, and have a probability of being *True* corresponding to the effects of the action, if the action node is *True*. Action start nodes have a probability of being *True*, corresponding to the success probability of the action if all its preconditions are met, and have a probability of 0 of being *True* otherwise. Action end nodes have a probability of 1 of being *True* if all its preconditons are met and the action start node is *True*, and have a probability of 0 of being *True* otherwise.

The success probability of each action, of the effects of each action and of facts spontaneously changing in the world are an input from the user.

With the DBN fully built, we can compute the success probability of a plan. This success probability is defined as the conjunction of all actions in the plan being successful and of the goal facts being in the final state, $P(\mathcal{A} \cap \mathcal{G})$. In other words, it is defined as the joint probability of all action nodes, in the DBN, being *True* and of the facts from the goal being *True* in the last layer, marginalised over all other nodes. This calculation is represented in Eq. (1), with $\mathcal{A}$ being the set of action nodes and $\mathcal{G}$ being the set of nodes corresponding to the goal propositions in the last layer of the DBN. In addition to this, we also calculate the joint probability of all action nodes being *True*, $P(\mathcal{A})$, as displayed in Eq. (2), to prune the search.

$$P(\mathcal{A} \cap \mathcal{G}) = \sum_{\mathcal{S}_0,\ldots,\mathcal{S}_{N-1}} \sum_{\mathcal{S}_N \setminus \mathcal{G}} \left[\prod_{S \in \mathcal{S}_0} \rho(\pi(S)) \prod_{T \in \mathcal{A}} P(T|Pa(T)) \prod_{S \in \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_N} P(S|Pa(S)) \right] \quad (1)$$

$$P(\mathcal{A}) = \sum_{\mathcal{S}_0,\ldots,\mathcal{S}_{N-1}} \sum_{\mathcal{S}_N} \left[\prod_{S \in \mathcal{S}_0} \rho(\pi(S)) \prod_{T \in \mathcal{A}} P(T|Pa(T)) \prod_{S \in \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_N} P(S|Pa(S)) \right] \quad (2)$$

In Eqs. (1) and (2), we calculate $P(\mathcal{A} \cap \mathcal{G})$ and $P(\mathcal{A})$, respectively, by marginalising over all other nodes in the network. The term $\prod_{S \in \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_N} P(S|Pa(S))$ represents the marginalisation of proposition nodes over its parents, The term $\prod_{T \in \mathcal{A}} P(T|Pa(T))$ represents marginalising an action node T over its parents $Pa(T)$ and, lastly, the term $\prod_{S \in \mathcal{S}_0} \rho(\pi(S))$ represents marginalising over the nodes of S_0 (initial state).

From Equations (2) and (1), it is possible to infer the following two propositions.

▶ **Proposition 2.** *$P(\mathcal{A})$ is monotonically decreasing as more actions are added to the DBN.*

Proof. As more actions are considered in $\mathcal{A}$, more factors are considered in $\prod_{T\in\mathcal{A}} P(T|Pa(T))$ and, since $P(T|Pa(T)) \leq 1$, the computed value will reduce or stay the same. ◀

▶ **Proposition 3.** *$P(\mathcal{A})$ is an upper bound for $P(\mathcal{A}\cap\mathcal{G})$, meaning $P(\mathcal{A}) \geq P(\mathcal{A}\cap\mathcal{G})$.*

Proof. This proposition comes from the fact that Equation (2) marginalises over a greater number of nodes than Equation (1). This can can be seen on the second sum of each Equation, since the sum $\sum_{S_N}$, from Equation (2), yields a greater or equal value than the sum $\sum_{S_N\setminus\mathcal{G}}$, from Equation (1). ◀

These two propositions enable the use of $P(\mathcal{A})$ to prune the Branch and Bound search, since it is an upper bound for $P(\mathcal{A}\cap\mathcal{G})$ and is monotonically decreasing as more actions are added to $\mathcal{A}$.

The DBN is then simplified by the iterative application of the 3 pruning rules defined below; these rules remove nodes and marginalisations which do not affect the probabilities calculation.

1. A node that is not the goal, is not an action and is not a parent of any other node can be discarded by marginalisation.
   Proof: We marginalise over all nodes which are neither the goal nor the actions. If a marginalised node has no children, then no other node depends on its value and it does not affect the probability calculation. The example below shows the effect of marginalising over node a, which has no children.

$$\sum_{a,b,c,\ldots} P(a|Pa(a)) \cdot \lambda(b,c,..) = \sum_{b,c} \lambda(b,c,..) \tag{3}$$

2. The marginalisation sum corresponding to a node that is precondition of an action can be removed.
   Proof: The marginalisation of an action's parent includes two terms, when it is *True* and when it is *False*. Only one of these terms satisfies the precondition of the action. The term which does not satisfy the action's precondition corresponds to the action having a null success probability, according to its CPD. Hence, we can discard the term which does not satisfy the action's preconditions and not marginalise over the parents of an action.
3. For all proposition nodes with an action node T as parent, the edges from any other parent can be removed.
   Proof: In our model, we have defined that the probability of a propositional node, with both another propositional node and an action node as parents, does not depend on the value of the propositional parent if the action parent is *True*, according to its CPD. Since we are only interested in calculating the case where all action nodes are *True*, then the edges from any other parents can be removed and the action node can be considered as the single parent of the propositional node.

Figure 3 illustrates which nodes can be removed, according to the 3 pruning rules, from the DBN of Figure 2.

Since we are building a DBN and calculating a probability at every step of the B&B search, the DBN is expected to be a bottleneck in terms of computing time and resources. Hence, it is vital to make this process efficient and iterative. Given this, instead of first building

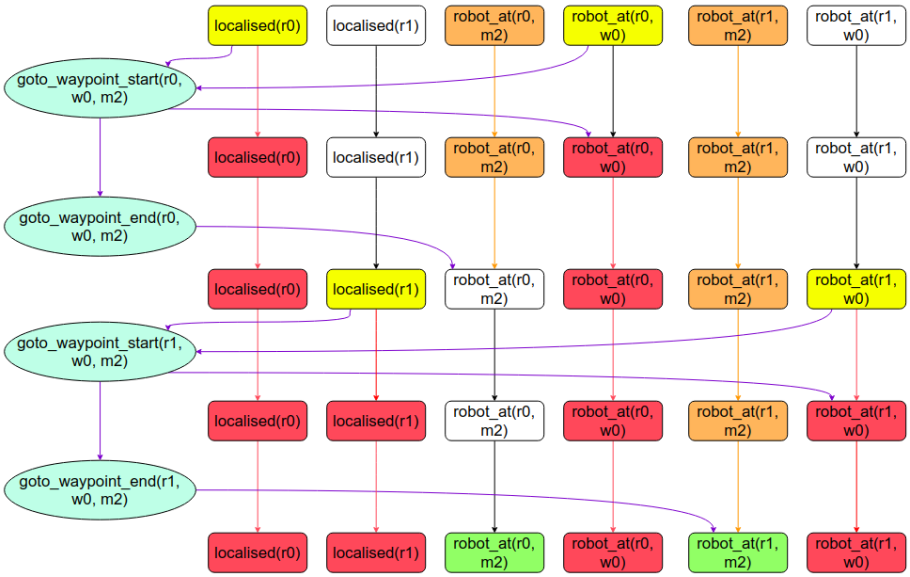


Figure 3 DBN representing which nodes can be removed from the DBN of Figure 2, by applying the 3 pruning rules. Rectangular nodes symbolise proposition nodes and ellipsoidal nodes represent action nodes. The colours of the nodes illustrate under which pruning rule they can be removed. Nodes and edges in red can be removed by Rule 1. Nodes in yellow have their marginalisation discarded, according to rule 2. Nodes and edges in orange can be removed after applying rule 3, followed by rule 1.

the entire DBN and then simplifying it, we build the already simplified DBN iteratively. In addition to this, we also use dynamic programming to calculate the values of $P(\mathcal{A})$ and $P(\mathcal{A} \cap \mathcal{G})$ iteratively.

The computation of the probabilities can be verified by resorting to any exact inference method, e.g., the well-known exact variable elimination.

2.2.2 PExec: Checking if a Plan is Compatible with the State of the World and Dispatching the Compatible Lattermost Action

In the PExec sub-phase, we wish to check if the state of the world is compatible with the plan Γ and to dispatch the lattermost action from Γ compatible with the state of the world. To achieve this, Algorithm 1 is used.

The pruning rules remove all irrelevant proposition nodes from the DBN, so the nodes remaining in the DBN, at a certain layer, must be verified in the state of the world to enable the execution of the action node that follows that layer.

Let us now consider the function *findCompatibleLattermostLayer* from Algorithm 1. This function traverses the DBN of plan Γ from the last layer to the first one, from S_N to S_0, searching for a layer which only contains propositions that match the state of the world, $\mathbb{S}$. After finding a layer i that matches this condition, it dispatches the action node after it for execution, a_i. If the action node a_i corresponds to the end of an action, then its corresponding action start must have already been dispatched. If all layers are verified and no layer satisfies these conditions, then the function returns -1, meaning that no layer is compatible with the state of the world.

As it can be seen in Figure 1, this verification will occur after the execution of each action. By considering the lattermost compatible action we are allowing for the skipping of actions. One important detail is that, due to the formalism of *PDDL*, every action that has been

Algorithm 1 Find the Lattermost Layer in the DBN Compatible with the State of the Sorld.

```
1: global 𝕊                                          ▷ Current state of the world.
2: global index_last_layer                           ▷ Index of the last DBN layer.
3: global actions_executing                          ▷ Set of executing actions.

4: function findCompatibleLattermostLayer
5:     if G ⊂ 𝕊 and actions_executing ≠ [] then
6:         return index_last_layer
7:     for i = index_last_layer-1 to 0 do
8:         if a_i is an action start node then
9:             if S_i ⊂ 𝕊 then
10:                return i
11:        else
12:            a_st = getCorrespondingActionStart(a_i)
13:            if S_i ⊂ 𝕊 and a_st ⊂ actions_executing then
14:                return i
15:    return −1                                     ▷ No layer matches 𝕊
```

started needs to be finished. So, even if the goal is already verified in the state of the world, the algorithm will first finish the actions that are still executing and only then declare that the goal has been reached. Another advantage that comes from using the pruned DBN is that we are only considering the propositions which are relevant for the execution of certain actions and for reaching the goal. If irrelevant propositions in the world change, then the same plan will remain valid. Plus, if the execution of a certain action fails, then the same plan might remain valid and the algorithm might retry the same action. Therefore, this algorithm can resist unexpected changes in the world and action failures without the need of replanning.

We allow for the repetition of actions because, in the interaction between a robot and the real world, it is very rare to have the exact same conditions. In fact, the failing of an action might be enough to slightly alter the environment. For example, a robot might fail to grasp a certain object, but in doing so change the object's pose such that it would succeed if it tried to grasp the object again. As another example, a robot might fail to navigate from room A to room B because someone stood on its way. However, if the robot tried to navigate again the person might no longer be there and the robot would navigate successfully. It is to account for situations like these that we allow for the repetition of actions.

3 Experimental Evaluation

Since the main feature of Olisipo consists in coping with unexpected changes in the world and action failures, it was necessary to develop a simulation environment capable of replicating these events. Hence, the environment developed for the original paper [21] was extended to account for simulated state perturbations.

The extended environment uses ROSPlan [5] to parse the domain and problems files. ROSPlan was chosen because it is compatible with ROS [25] and it is able to parse the PDDL language. The planner POPF [8] was used to build the totally-ordered plan $\prod^{tt}$. ROS Services were used to handle the communication between different scripts. The ROS version utilised was ROS Kinetic Kame[1].

[1] http://wiki.ros.org/kinetic

Scripts were developed to simulate perturbations in the environment. These perturbations occur after the execution of the start or end of an action and consist in randomly adding and removing propositions, as well as accounting for the failure of actions and of the effects of actions. The user defines the perturbations' probabilities in a configuration file which is then used to build and make calculations in the DBN.

Olisipo was compared to a simple dispatcher, which tries to execute a temporal plan and replans when the execution fails. Hence, the Esterel Dispatcher, from ROSPlan, was used and POPF was used to build the plan, since it was the same planner used in the offline phase of Olisipo. Ten replans was defined as the maximum amount of allowed replans both for the Esterel dispatcher and for the Olisipo algorithm.

3.1 Results

The Olisipo and Esterel Dispatchers were compared by performing 2000 trials on 10 different problems of *Simple Factory Robot Domain with 3 machines* (SF3) and 2000 trials on 8 different problems of *Advanced Factory Robot Domain with 3 machines* (AF3). The SF3 domain only has one action, where the AF3 domain has two actions. Each problem was manually generated and represents a different perturbed environment[2].

The chosen metrics to compare both dispatchers are:

- **M1**: Estimated probability of a successful execution;
- **M2**: Average number of replans;
- **M3**: Average number of actions executed.

Considering that each algorithm is either able or not able to solve a problem, its success distribution is a discrete binary distribution, namely a Bernoulli distribution. Hence, the Wilson Score Interval [29], presented in Equation (4), can be used to estimate the probability of each algorithm solving a problem, corresponding to M1.

In Equation (4), n is the total number of trials, n_s is the number of successful trials, n_f is the number of failed trials and z is the quantile function, with $z = 1.9599$ for a 95% confidence interval.

$$\hat{p} = \frac{n_s + \frac{1}{2}z^2}{n + z^2} \pm \frac{z}{n + z^2}\sqrt{\frac{n_s n_f}{n} + \frac{z^2}{4}} \tag{4}$$

Table 1 Tables showing the estimated probabilities of the Esterel dispatcher and of the Olisipo algorithm solving each problem from the SF3 and AF3 domains. These probabilities were calculated by making use of Equation (4). The last column displays the range of values where the estimated improvement, due to the use of Olisipo, lays.

SF3			
Problem	**Esterel**	**Olisipo**	**Improvement**
p1	0.13 ± 0.01	0.23 ± 0.02	$[0.07, 0.13]$
p2	0.10 ± 0.01	0.22 ± 0.02	$[0.09, 0.15]$
p3	0.07 ± 0.01	0.18 ± 0.02	$[0.08, 0.14]$
p4	0.043 ± 0.009	0.17 ± 0.02	$[0.10, 0.16]$
p5	0.024 ± 0.007	0.12 ± 0.01	$[0.08, 0.11]$
p6	0.06 ± 0.01	0.08 ± 0.01	$[0.00, 0.04]$
p7	0.020 ± 0.006	0.043 ± 0.009	$[0.008, 0, 038]$
p8	0.12 ± 0.01	0.16 ± 0.02	$[0.01, 0.07]$
p9	0.20 ± 0.02	0.23 ± 0.02	$[-0.01, 0.07]$
p10	0.025 ± 0.007	0.038 ± 0.008	$[-0.002, 0.028]$

AF3			
Problem	**Esterel**	**Olisipo**	**Improvement**
p1	0.17 ± 0.02	0.22 ± 0.02	$[0.01, 0.09]$
p2	0.15 ± 0.02	0.20 ± 0.02	$[0.01, 0.09]$
p3	0.16 ± 0.02	0.23 ± 0.02	$[0.03, 0.11]$
p4	0.17 ± 0.02	0.21 ± 0.02	$[0.00, 0.08]$
p5	0.09 ± 0.01	0.09 ± 0.01	$[-0.02, 0.02]$
p6	0.038 ± 0.008	0.047 ± 0.009	$[-0.008, 0.026]$
p7	0.029 ± 0.007	0.038 ± 0.008	$[-0.006, 0.024]$
p8	0.024 ± 0.007	0.031 ± 0.008	$[-0.008, 0.022]$

[2] Domains, problems, and configurations are available online [27].

The results referring to M1 can be seen in Table 1. From these tables, it is possible to verify that, overall, the Olisipo algorithm achieved a higher probability of success than the Esterel dispatcher. On the SF3 domain, the Olisipo algorithm presented an estimated improvement between -0.02 and 0.16. On the AF3 domain, the improvement was between -0.02 and 0.11. Despite the presence of some negative values, indicating a possible worst performance for the Olisipo algorithm, it is worth noting that on 12 of the 18 problems used, the improvement interval contained only positive values, indicating that Olisipo does present an improvement. Regarding the other problems, with the exception of problem 5 of AF3, at least 73% of the estimated interval is positive.

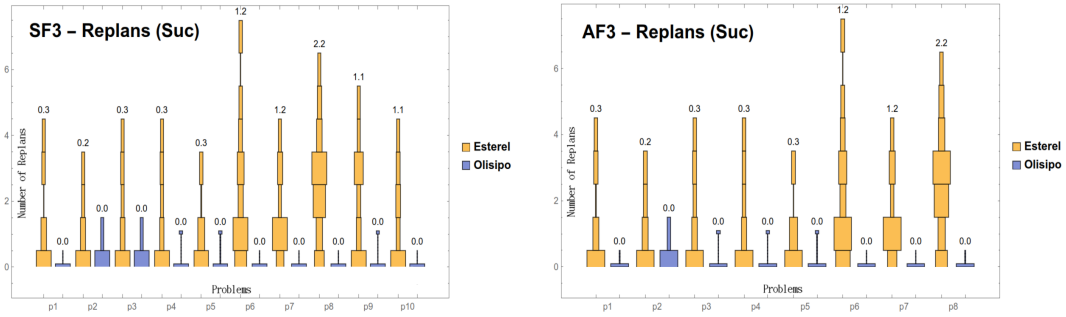


Figure 4 Distribution chart of the number of replans performed in the successful executions of the 2000 trials, for each problem of the SF3 and AF3 domains and for each dispatcher. The number on top of each distribution corresponds to its median.

Figure 4 presents the results referring to M2, namely the distribution of the number of replans, on successful executions, for each algorithm and for each problem. A successful execution occurs when a problem is successfully solved, with 10 or fewer replans performed. In these Figures, it is possible to verify that, for successful executions, the mean of the number of replans on the Olisipo algorithm is consistently lower than on the Esterel dispatcher. The mean of the number of replans in successful executions on the Olisipo algorithm was consistently zero, whereas the Esterel dispatcher often needed to replan, even achieving more than 5 replans on some problems.

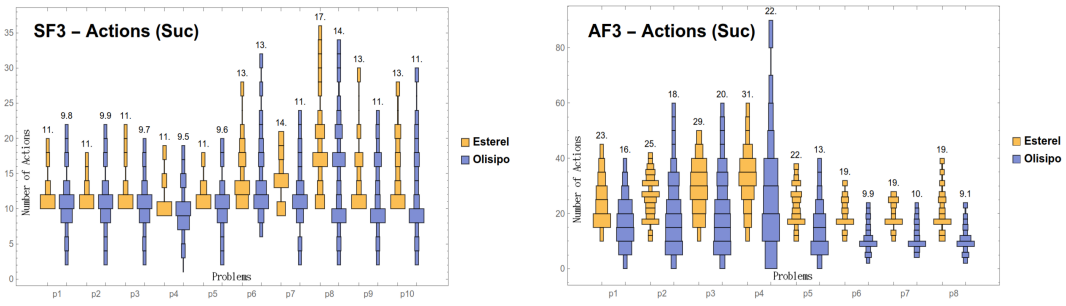


Figure 5 Distribution chart of the number of actions executed in the successful executions of the 2000 trials, for each problem of the SF3 and AF3 domains and for each dispatcher. The number on top of each distribution corresponds to its median.

Figure 5 presents the results corresponding to M3, more specifically to the distribution for the number of actions executed on successful executions. From these results, is possible to verify that the mean for the number of actions in successful executions on the Olisipo algorithm was consistently lower than on the Esterel dispatcher. For all problems of SF3,

except problem 6, the Olisipo algorithm offered a relative improvement of, at least, 10% in the mean of the number of actions for successful executions. The biggest relative improvements were achieved on problems 7 and 8, where the improvements were 21% and 18%, respectively. For all problems of AF3, the Olisipo algorithm offered an improvement of, at least, 28%. The biggest relative improvements were achieved on problems 6, 7 and 8, where the improvements were 48%, 47% and 52%, respectively.

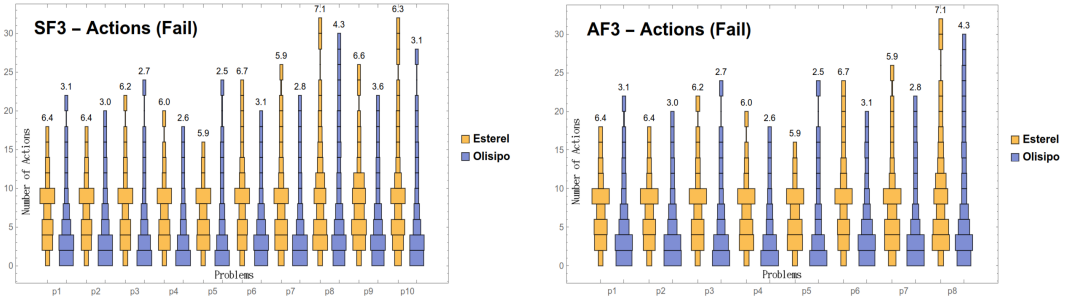


Figure 6 Distribution chart of the number of actions executed in the failed executions of the 2000 trials, for each problem of the SF3 and AF3 domains and for each dispatcher. The number on top of each distribution corresponds to its median.

Lastly, we will analyse the distribution of the number of actions on unsuccessful executions, from Figure 6, which also corresponds to M3. The mean and mode of executed actions before failing was lower on the Olisipo algorithm than on the Esterel dispatcher, for all problems of SF3 and AF3. The mean of executed actions before failure on SF3 was improved by, at least, 39% with the Olisipo algorithm. The biggest relative improvements occurred on problems 3, 4 and 5, where they were 56%, 57% and 58%, respectively. On the problems of AF3, the Olisipo algorithm also offered an improvement of at least 39% on the mean of executed actions before failure. The biggest improvements were achieved on problems 3, 4 and 5, with an improvement of 56%, 57% and 58% respectively.

Olisipo executes less actions than the Esterel dispatcher on failed executions because it requires the conditions of future actions to hold in the current state. This is achieved through the propagation of the parents of an action node to the previous layers in the DBN. Hence, Olisipo avoids the partial execution of plans which will not achieve the goal.

Regarding the execution time of the algorithm, it was found that the execution of the PGen sub-phase, which is the most resource consuming sub-phase, consistently had a duration under 2 seconds for all of the problems used.

From these results, it is possible to conclude that, in comparison to a simple dispatcher, the Olisipo algorithm has a higher probability of solving a given problem, needs to replan less times to successfully solve a problem and executes less actions on successful and failed executions. The consistency in the obtained results is especially useful to support this claim, since the results were obtained from 18 different problems from 2 different domains.

There were only two instances where the performance of Olisipo was similar to the Esterel dispatcher, which was the success probability of problem 5 of AF3 and the number of actions on successful executions of problem 6 of SF3. However, the performance of Olisipo in the other metrics, for each problem, surpassed the performance of the Esterel dispatcher. Hence, even in problems where the performance of Olisipo, in one metric, is similar to that of the Esterel dispatcher, the performance in the other metrics favour Olisipo.

Regarding the mean of executed actions in successful executions, the difference between Olisipo and Esterel was greater in AF3 than in SF3. Considering that SF3 only has one action on its domain, whereas AF3 has two actions, it is possible that this difference increases for domains with a greater number of actions.

Regarding the number of replans on successful executions, the mean for Olisipo was consistently zero for all problems. This means that Olisipo almost never needed to replan to achieve a successful execution. At the same time, it also means that Olisipo's replans rarely resulted in a successful execution. This implies that the replanning effort requested by the Esterel dispatcher was largely to recompute the same (sub)set of actions as was originally generated. In scenarios with limited computational time to spend planning, or larger problems for which full replanning is not feasible, the flexible approach of Olisipo is an obvious solution. More tests to evaluate this metric would be interesting, since this could mean that repeating the PGen sub-phase does not achieve any additional successful executions.

4 Conclusion

The robust execution of a temporal plan in a perturbed environment is a problem that remains to be solved. Perturbed environments, such as the real world, are non-deterministic and filled with uncertainty. Hence, the execution of a temporal plan presents several challenges and the employed solution often consists of replanning when the execution fails. In this paper, we propose a novel algorithm, named Olisipo, which aims to maximise the probability of a successful execution of a temporal plan in perturbed environments. To achieve this, a probabilistic model is used in the execution of the plan, instead of in the building of the plan. This approach enables Olisipo to dynamically adapt the plan to changes in the environment. In addition to this, the execution of the plan is also adapted to the probability of successfully executing each action. Olisipo was compared to a simple dispatcher and it was shown that it consistently had a higher probability of successfully solving a problem, performed fewer replans and executed fewer actions. Hence, Olisipo offers a substantial improvement in performance for disturbed environments.

Regarding the future work, there are several directions it could take. One possibility would be to save intermediate results, either from the DBN or the search tree, to cope with perturbations more efficiently. In addition to this, it would be interesting to study further the relevance of repeating the PGen sub-phase. Lastly, another possibility would be to test the Olisipo algorithm in a real world robot.

All source code and experimental setup is open source and available online [27]. Olisipo works as an add-on to any external planner and can easily be implemented by third-parties to improve execution performance.

References

1 Christer Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

2 Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

3 Michael Cashmore, Alessandro Cimatti, Daniele Magazzeni, , Andrea Micheli, and Parisa Zehtabi. Robustness envelopes for temporal plans. In *AAAI*, 2019.

4 Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Replanning for situated robots. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pages 665–673. AAAI Press, 2019. URL: `https://aaai.org/ojs/index.php/ICAPS/article/view/3534`.

5 Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *ICAPS*, 2015.

6 A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and R. Rasconi. The APSI Framework: a Planning and Scheduling Software Development Environment. In *ICAPS (Application Showcase)*, 2009.

7 Alessandro Cimatti, Minh Do, Andrea Micheli, Marco Roveri, and David E. Smith. Strong temporal planning with uncontrollable durations. *Artif. Intell.*, 256:1–34, 2018. doi:10.1016/j.artint.2017.11.006.

8 Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. *ICAPS 2010 - Proceedings of the 20th International Conference on Automated Planning and Scheduling*, pages 42–49, January 2010.

9 Patrick R. Conrad and Brian Charles Williams. Drake: An efficient executive for temporal plans with choice. *J. Artif. Intell. Res.*, 42:607–659, 2011. doi:10.1613/jair.3478.

10 Paul Dagum, Adam Galper, and Eric Horvitz. Dynamic network models for forecasting. In *Uncertainty in Artificial Intelligence*, pages 41–48. Morgan Kaufmann, 1992. doi:10.1016/B978-1-4832-8287-9.50010-4.

11 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 1991.

12 M. Do and S. Kambhampati. Improving temporal flexibility of position constrained metric temporal plans. In *ICAPS*, pages 42–51, 2003.

13 Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, pages 212–221. AAAI, 2006.

14 Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 2003.

15 J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003.

16 J. Frank and P. Morris. Bounding the resource availability of activities with linear resource impact. In *ICAPS*, pages 136–143, 2007.

17 M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

18 Félix Ingrand and Malik Ghallab. Deliberation for autonomous robots: A survey. *Artif. Intell.*, 247:10–44, 2017.

19 Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 105–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-540-68279-0_5.

20 Steven Levine and Brian Williams. Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. *Journal of Artificial Intelligence Research*, 63, 2018.

21 Oscar Lima, Michael Cashmore, Daniele Magazzeni, Andrea Micheli, and Rodrigo Ventura. Robust plan execution with unexpected observations. *CoRR*, abs/2003.09401, 2020. arXiv:2003.09401.

22 Paul Morris. Dynamic controllability and dispatchability relationships. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 464–479, Cham, 2014. Springer International Publishing.

23 Paul H. Morris and Nicola Muscettola. Execution of temporal plans with uncertainty. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 491–496, 2000.

24 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), Trento, Italy, June 2-5, 1998.*, pages 444–452, 1998.

25 Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, 2009.

26 Masood Feyzbakhsh Rankooh and Gholamreza Ghassem-Sani. Itsat: An efficient sat-based temporal planner. *Journal of Artificial Intelligence Research*, 53(1):541–632, 2015.

27 T. Ribeiro, O. Lima, M. Cashmore, A.Micheli, and R. Ventura. Experimental material for "A Probabilistic Approach to the Adaptable Execution of Deterministic Temporal Plans". URL: `https://github.com/TomasRibeiro96/Olisipo-planner`.

28 Ioannis Tsamardinos and Martha E. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artif. Intell.*, 151(1-2):43–89, 2003. `doi:10.1016/S0004-3702(03)00113-9`.

29 Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. `doi:10.1080/01621459.1927.10502953`.

30 Sung Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. *ICAPS 2007*, pages 352–, 2007.

A One-Pass Tree-Shaped Tableau for Defeasible LTL

Anasse Chafik
CRIL, University of Artois & CNRS, Arras, France

Fahima Cheikh-Alili
CRIL, University of Artois & CNRS, Arras, France

Jean-François Condotta
CRIL, University of Artois & CNRS, Arras, France

Ivan Varzinczak
CRIL, University of Artois & CNRS, Arras, France

Abstract

Defeasible Linear Temporal Logic is a defeasible temporal formalism for representing and verifying exception-tolerant systems. It is based on Linear Temporal Logic (LTL) and builds on the preferential approach of Kraus et al. for non-monotonic reasoning, which allows us to formalize and reason with exceptions. In this paper, we tackle the satisfiability checking problem for defeasible LTL. One of the methods for satisfiability checking in LTL is the one-pass tree shaped analytic tableau proposed by Reynolds. We adapt his tableau to defeasible LTL by integrating the preferential semantics to the method. The novelty of this work is in showing how the preferential semantics works in a tableau method for defeasible linear temporal logic. We introduce a sound and complete tableau method for a fragment that can serve as the basis for further exploring tableau methods for this logic.

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

Keywords and phrases Temporal logic, Non-monotonic reasoning, Tableau Calculi

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.16

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Linear temporal logic (LTL) was introduced by Pnueli [13] as a formal tool for reasoning about programs execution. Many properties that an execution should have can be expressed elegantly using this formalism. The logic LTL is used for systems verification [16]. With advances in technologies, systems became more and more complex, displaying new features and behaviours. One of these behaviours is tolerating exceptions. In more general terms, if an error occurs, within an execution of a program, at certain points of time where it is tolerated, the program can still function properly.

Let us say, for the sake of argument, that there is an execution of a program in which a parameter cannot have a certain value. We notice that, at some given points of time, the execution produces the invalid value in the aforementioned parameter. Nevertheless, we do not mind that the program produces the error at these time points deemed to be harmless. The crucial point is that this behaviour is not present in other, more important, points of time. We want to be sure that the execution still continues and the program functions properly even in the presence of such benign time points.

We want a formalism for verifying properties of executions that can, on one hand, be strictly required at some points of time, and on the other hand, be missing in other points of time. That is why we introduced an extended formalism of LTL, called defeasible

linear temporal logic ($LTL^{\sim}$) [6]. It uses the preferential approach of Kraus et al. to non-monotonic reasoning [11] (a.k.a. the KLM approach). The defeasible aspect of $LTL^{\sim}$ adds a new dimension to the verification of a program's execution. We can order time points from the important ones, which we call *normal*, to the lesser and lesser ones. *Normality* in LTL indicates the importance of a time point within an execution compared to others.

We also introduced defeasible versions of the modalities *always* and *eventually*. With these defeasible modalities, we can express properties similar to their classical counterparts, targeting the most normal time points within the execution.

The main goal of this paper is to establish a *satisfiability checking* method for our logic, in particular, for a fragment thereof. In the case of LTL, many tableau methods were proposed in the literature. There are two types of tableau methods: *multi-pass* and *one-pass* tableaux. Multi-pass tableau methods [22, 12, 10] go through an initial phase of building a tree-shaped structure by putting the sentence in the root node and expanding the tableau via a systematic application of a set of rules. The second phase is a *culling* phase, which uses an auxiliary structure built from the tableau, and checks for the satisfiability of the input sentence in this structure. Whereas in *one-pass* tableau methods [17, 14], the construction and the verification are done simultaneously. Reynolds' tableau for LTL [15, 14] is a tree-shaped one-pass tableau where each branch is independent from the others. Moreover, each successful branch by itself is a representation of an interpretation that satisfies the sentence.

As for the KLM approach, tableau methods were developed for the preferential approach of Kraus et al. logic [11] and formalisms extending the preferential approach [9, 4, 5]. In the case of preferential modal logic, Britz and Varzinczak [4] proposed a tree-shaped tableau that builds the ordering relation on worlds at the same time as the tableau is expanded. The tableau method in this paper is based on both the one-pass tableau of Reynolds [14] and the tableau for preferential modal logic by Britz and Varzinczak [4]. The novelty of this paper is in showing how preferential semantics works in a tableau for a fragment of $LTL^{\sim}$.

The plan of this paper goes the following way: We talk briefly about LTL and $LTL^{\sim}$ in Section 2. We then describe a tableau method for a fragment of $LTL^{\sim}$ in Section 3. We show soundness, and completeness of our method in Section 4. Section 5 concludes the paper.

2 Preliminaries

Linear Temporal Logic [1] is a modal logic in which modalities are considered to be temporal operators that describe events happening in different time points over a linearly ordered time-line. Let $\mathscr{P}$ be a finite set of *propositional atoms*. The set of operators in LTL can be split into two parts: the set of *Boolean connectives* ($\neg, \wedge, \vee$), and that of *temporal operators* ($\Box, \Diamond, \bigcirc$), where $\Box$ reads as *always*, $\Diamond$ as *eventually*, and $\bigcirc$ as *next*. Let $p \in \mathscr{P}$, sentences in LTL are built up according to the following grammar: $\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \Box\alpha \mid \Diamond\alpha \mid \bigcirc\alpha$.

Standard abbreviations are included in LTL, such as: $\top \stackrel{\text{def}}{=} p \vee \neg p$, $\bot \stackrel{\text{def}}{=} p \wedge \neg p$, $\alpha \rightarrow \beta \stackrel{\text{def}}{=} \neg\alpha \vee \beta$ and $\alpha \leftrightarrow \beta \stackrel{\text{def}}{=} (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$. There are other temporal operators such as $\mathscr{U}$ (until operator) and $\mathscr{R}$ (release operator) in LTL, but we chose to omit them in this paper.

The temporal semantics structure is a chronological linear succession of time points. We use the set of natural numbers in order to label each of these time points i.e., $(\mathbb{N}, <)$. Hence, a temporal interpretation associates each time point t with a truth assignment of all propositional atoms. A temporal interpretation is defined as follows:

▶ **Definition 1** (Temporal interpretation). *A* temporal interpretation I *is a mapping function* $V : \mathbb{N} \longrightarrow 2^{\mathscr{P}}$ *which associates each time point* $t \in \mathbb{N}$ *with a set of propositional atoms* $V(t)$ *corresponding to the set of propositions that are true in* t*. (Propositions not belonging to* $V(t)$ *are assumed to be false at the given time point.)*

The truth value of a sentence in an interpretation I at a time point $t \in \mathbb{N}$, denoted by $I, t \models \alpha$, is recursively defined as follows:

- $I, t \models p$ if $p \in V(t)$; $I, t \models \neg\alpha$ if $I, t \not\models \alpha$;
- $I, t \models \alpha \wedge \alpha'$ if $I, t \models \alpha$ and $I, t \models \alpha'$; $I, t \models \alpha \vee \alpha'$ if $I, t \models \alpha$ or $I, t \models \alpha'$;
- $I, t \models \Box\alpha$ if $I, t' \models \alpha$ for all $t' \in \mathbb{N}$ s.t. $t' \geq t$; $I, t \models \Diamond\alpha$ if $I, t' \models \alpha$ for some $t' \in \mathbb{N}$ s.t. $t' \geq t$;
- $I, t \models \bigcirc\alpha$ if $I, t+1 \models \alpha$.

In previous work [6], we introduced a new formalism called preferential linear temporal logic. The motivation is to provide a formalism for the specification and verification of systems where exceptions can be tolerated.

Let $p \in \mathscr{P}$, sentences of the logic $LTL^{\sim}$ are built up according to the following grammar:

$$\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \Box\alpha \mid \Diamond\alpha \mid \bigcirc\alpha \mid \boxdot\alpha \mid \blacklozenge\alpha$$

The intuition behind the new temporal operators is the following: $\boxdot$ reads as *non-monotonic always* and $\blacklozenge$ reads as *non-monotonic eventually*. The set of all well-formed $LTL^{\sim}$ sentences is denoted by $\mathcal{L}^{\sim}$. It is worth to mention that any well-formed sentence α in LTL is a sentence of $\mathcal{L}^{\sim}$.

A sentence such as $\boxdot\alpha$ reads as: in all normal future time points, α is true. A sentence of the form $\blacklozenge\alpha$ reads as: in some normal future time point, α is true. We can even express properties using a mix of classical and non-monotonic operators. A sentence $\Box\blacklozenge\alpha$ reads as: always, there is a normal future time point where α is true.

The preferential component of the interpretation of our language is directly inspired by the preferential semantics proposed by Shoham [19] and used in the KLM approach [11]. The ordering relation, denoted by $\prec$, is a strict partial order on points of time. Following Kraus et al. [11], $t \prec t'$ means that t is more preferred than t'. We use the pair notation $(t, t') \in \prec$ to indicate that t is more normal than t' w.r.t. $\prec$.

▶ **Definition 2** (Minimality w.r.t. $\prec$). *Let $\prec$ be a strict partial order on a set $\mathbb{N}$ and $N \subseteq \mathbb{N}$. The set of the minimal elements of N w.r.t. $\prec$, denoted by $min_{\prec}(N)$, is defined by $min_{\prec}(N) \stackrel{\text{def}}{=} \{t \in N \mid$ there is no $t' \in N$ such that $(t', t) \in \prec\}$.*

▶ **Definition 3** (Well-founded set). *Let $\prec$ be a strict partial order on a set $\mathbb{N}$. We say $\mathbb{N}$ is* well-founded w.r.t. $\prec$ *iff $min_{\prec}(N) \neq \emptyset$ for every $\emptyset \neq N \subseteq \mathbb{N}$.*

In what follows, given a relation $\prec$ and a time point $t \in \mathbb{N}$, the set of *preferred time points relative to t* is the set $min_{\prec}([t, \infty[)$ which is denoted in short by $min_{\prec}(t)$.

▶ **Definition 4** (Preferential temporal interpretation). *An $LTL^{\sim}$ interpretation on a set of propositional atoms $\mathscr{P}$, also called preferential temporal interpretation on $\mathscr{P}$, is a pair $I \stackrel{\text{def}}{=} (V, \prec)$ where V is a mapping function $V : \mathbb{N} \longrightarrow 2^{\mathscr{P}}$, and $\prec \subseteq \mathbb{N} \times \mathbb{N}$ is a strict partial order on $\mathbb{N}$ such that $\mathbb{N}$ is well-founded w.r.t. $\prec$. We denote the set of preferential temporal interpretations by $\mathfrak{I}$.*

Preferential temporal interpretations provide us with an intuitive way of interpreting sentences of $\mathcal{L}^{\sim}$. Let $\alpha \in \mathcal{L}^{\sim}$, let $I = (V, \prec)$ be a preferential temporal interpretation, and let t be a time point in I in $\mathbb{N}$. Satisfaction of α at t in I, denoted $I, t \models \alpha$, is defined as follows:

- The truth values of Boolean connectives and classical modalities are defined as in LTL.
- $I, t \models \boxdot\alpha$ if $I, t' \models \alpha$ for *all* $t' \in min_{\prec}(t)$;
- $I, t \models \blacklozenge\alpha$ if $I, t' \models \alpha$ for *some* $t' \in min_{\prec}(t)$.

We say $\alpha \in \mathcal{L}^{\sim}$ is *satisfiable* if there is a preferential temporal interpretation I and a time point t in $\mathbb{N}$ such that $I, t \models \alpha$. We can show that $\alpha \in \mathcal{L}^{\sim}$ is *satisfiable* iff there is a preferential temporal interpretation I s.t. $I, 0 \models \alpha$.

3 A one-pass tableau for $LTL^{\sim}$

In this paper, we address the computational task of *satisfiability checking* in $LTL^{\sim}$. That is, given a sentence α in $LTL^{\sim}$, decide whether or not there is an interpretation I that satisfies the sentence α. As mentioned in the Introduction, we propose a one-pass tree-shaped tableau for a fragment of $LTL^{\sim}$ based on Reynolds' tableau [15] and inspired by the semantic rules for defeasible modalities in modal logic proposed by Britz and Varzinczak [4]. This fragment, denoted by $\mathcal{L}_1$, serves as a starting point for showing how the ordering $\prec$ is built for preferential interpretations in $LTL^{\sim}$.

3.1 The fragment $\mathcal{L}_1$

The fragment $\mathcal{L}_1$ considers that sentences are in NNF (negation is only allowed on the level of atomic propositions). On the other hand, the non-monotonic operator $\bowtie$ is omitted from $\mathcal{L}_1$. Furthermore, only Boolean sentences are permitted within the scope of $\Box$ sentences. In what follows, we define formally well formed sentences of $\mathcal{L}_1$. In order to do that, we introduce first the set of Boolean sentences $\mathcal{L}_{bool}$. Let $p \in \mathscr{P}$, sentences $\alpha_{bool} \in \mathcal{L}_{bool}$ are defined recursively as such:

$$\alpha_{bool} ::= p \mid \neg p \mid \alpha_{bool} \wedge \alpha_{bool} \mid \alpha_{bool} \vee \alpha_{bool}$$

Next, let $\alpha_{bool} \in \mathcal{L}_{bool}$, sentences in $\mathcal{L}_1$ are recursively defined as such:

$$\alpha ::= \alpha_{bool} \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \Diamond\alpha \mid \Box\alpha_{bool} \mid \bigcirc\alpha \mid \mathbin{\tilde{\Diamond}}\alpha$$

Sentences of the form $\Diamond\alpha$ are called *eventualities*, because its truth depends on α being true in the future. Similarly, sentences of the form $\mathbin{\tilde{\Diamond}}\alpha$ are called *non-monotonic eventualities*. Their truth depends not only on α being true in some future, but it depends also on this future being preferred to the other future time points. Sentences of the form $\bigcirc\Diamond\alpha$ are called $\bigcirc$-eventuality.

3.2 Tableau method for $\mathcal{L}_1$

A tableau for $\alpha \in \mathcal{L}_1$ is a tree of nodes. Each node has a positive integer n as a *label*. It has also two sets of sentences: one we denote as Γ and the other as *une* (which stands for unfulfilled non-monotonic eventualities, a notion to be detailed below). The set Γ is a subset of $\mathcal{L}_1$ which contains the sentences in the node. The set *une* is a set of pairs $(n_k, \mathbin{\tilde{\Diamond}}\alpha_k)$, where n_k is a label and $\mathbin{\tilde{\Diamond}}\alpha_k$ is a non-monotonic eventuality.

▶ **Definition 5** (Labelled node). *A labelled node is a triple of the form* $n : (\Gamma, une)$ *where* $n \in \mathbb{N}$, $\Gamma \subseteq \mathcal{L}_1$ *and* $une \subseteq [0, n] \times \mathcal{L}_1$.

It is worth to mention that different nodes can have the same label. Intuitively, the nodes labelled by a same integer n represent the set of sentences that are satisfied at the time point associated with n. Hence, these nodes correspond with a given temporal state.

A branch B is a sequence of nodes, we introduce also a strict partial ordering relation $\prec_B$ on the labels of the nodes within the branch. The branch B has also a set of pairs of labels denoted by min_B. The relation $\prec_B$ represents a preference relation on the temporal

states of the branch B. On the other hand, the set min_B represents some constraints that the final preference relation issued from B must satisfy. More precisely, each pair (n, n') in min_B indicates that n' represents a preferred temporal state compared to all $n'' \geq n$.

▶ **Definition 6** (Branch). *A branch is a tuple* $B \stackrel{\text{def}}{=} (\langle x_0, x_1, x_2, \dots \rangle, \prec_B, min_B)$ *where the first element is a sequence of labelled nodes* $x_i := n_i : (\Gamma_i, une_i)$, $\prec_B$ *is a strict partial order* ($\prec_B \subseteq \mathbb{N} \times \mathbb{N}$) *on labels within the branch, and* min_B *is a set of pairs of labels* ($min_B \subseteq \mathbb{N} \times \mathbb{N}$).

Let $B := (\langle x_0, x_1, x_2, \dots \rangle, \prec_B, min_B)$ be a branch, x_n, x_m be two labelled nodes in B. If x_m comes after x_n in the sequence, then x_m is a *successor* of x_n, and x_n is a *predecessor* of x_m. We denote it by $x_n \leq x_m$. Moreover, if x_m is not the same labelled node as x_n, we say that x_m is a proper successor of x_n (same goes for a proper predecessor). We denote it by $x_n < x_m$. The last node of a branch is called a *leaf node*. When a leaf node is ticked with ✓, we say that the branch is a successful branch. On the other hand, when a leaf node is crossed with ✗, we say that the branch is a failed branch.

A tree is a set of branches $\mathcal{T} \stackrel{\text{def}}{=} \{B_0, B_1, B_2, B_3, \dots, B_k\}$ where $k \geq 0$. A tableau $\mathscr{T}$ for α is the limit of a sequence of trees $\langle \mathcal{T}^0, \mathcal{T}^1, \mathcal{T}^2, \dots \rangle$ where the initial tree is $\mathcal{T}^0 := \{(\langle 0 : (\alpha, \emptyset) \rangle, \emptyset, \emptyset)\}$ and every $\mathcal{T}^{i+1}$ is obtained from $\mathcal{T}^i$ by applying a rule on one of its branches. We say that a tableau $\mathscr{T}$ for α is *saturated* if no more rules can be applied after a tree $\mathcal{T}$.

We have two types of rules, static and dynamic rules. We introduce static rules first. Let $\mathcal{T}$ be a tree, and let B be a branch of $\mathcal{T}$ that has a leaf $n : (\Gamma, une)$. We say that a static rule (ρ) is applicable at the leaf $n : (\Gamma, une)$ if a sentence in Γ or a pair in une instantiates the pattern ρ. A static rule is a rule of the form:

$$(\rho)\ \frac{n : (\Gamma, une), \prec_B, min_B}{n : (\Gamma_1, une_1), \prec_{B_1}, min_{B_1} \mid \dots \mid n : (\Gamma_k, une_k), \prec_{B_k}, min_{B_k}}$$

In a tree $\mathcal{T}^i$, after applying the static rule (ρ), we obtain the tree $\mathcal{T}^{i+1}$ by replacing the branch $B := (\langle x_0, x_1, x_2, \dots, n : (\Gamma, une) \rangle, \prec_B, min_B)$ by the branches $B_1 := (\langle x_0, x_1, x_2, \dots, n : (\Gamma, une), n : (\Gamma_1, une_1) \rangle, \prec_{B_1}, min_{B_1})$, $B_2 := (\langle x_0, x_1, x_2, \dots, n : (\Gamma, une), n : (\Gamma_2, une_2) \rangle, \prec_{B_2}, min_{B_2})$, and so on. The symbol "|" indicates the occurrence of a split in the branch, i.e., a non-deterministic choice of possible outcomes, each of which needs to be explored. It is worth to mention that after applying a static rule on $n : (\Gamma, une)$, the leaf nodes of all the new branches keep the same label n.

In what follows, we show the rules for Boolean and the operators $(\Box, \Diamond)$. We also show two stopping conditions, namely, **Empty** and **Contradiction**. We chose to omit $\prec_B$ and min_B to lighten these rules. The crucial detail to remember is that they do not change after applying the rules below, i.e., $\prec_{B_i} = \prec_B$ and $min_{B_i} = min_B$ for all resulting branches. The symbol $\cup$ is the union of two sets. The symbol $\uplus$ represents the union between disjoint sets.

$$\text{(Contradiction)}\ \frac{n : (\{\alpha, \neg\alpha\} \uplus \Sigma\}, une)}{(\text{✗})} \qquad \text{(Empty)}\ \frac{n : (\emptyset, \emptyset)}{(\text{✓})}$$

$$(\wedge)\ \frac{n : (\{\alpha_1 \wedge \alpha_2\} \uplus \Sigma, une)}{n : (\{\alpha_1, \alpha_2\} \cup \Sigma, une)} \qquad (\vee)\ \frac{n : (\{\alpha_1 \vee \alpha_2\} \uplus \Sigma, une)}{n : (\{\alpha_1\} \cup \Sigma, une) \mid n : (\{\alpha_2\} \cup \Sigma, une)}$$

$$(\Box)\ \frac{n : (\{\Box\alpha_1\} \uplus \Sigma, une)}{n : (\{\alpha_1, \bigcirc\Box\alpha_1\} \cup \Sigma, une)} \qquad (\Diamond)\ \frac{n : (\{\Diamond\alpha_1\} \uplus \Sigma, une)}{n : (\{\alpha_1\} \cup \Sigma\}, une) \mid n : (\{\bigcirc\Diamond\alpha_1\} \cup \Sigma\}, une)}$$

Before introducing the rule for the non-monotonic operator $\Diamond\!\!\!\!-$, we discuss firsthand the notion of *fulfillment* for classical and non-monotonic eventualities. Following Reynolds' tableau, let an eventuality $\Diamond\alpha$ be in a node with a label n. If the sentence α appears in a proper successor node x with the label $m \geq n$, we say that $\Diamond\beta$ at the position n is *fulfilled* in m. In a similar fashion, we define the fulfillment for non-monotonic eventualities as follows:

▶ **Definition 7** (Fulfillment of non-monotonic eventualities)**.** *Let a non-monotonic eventuality $\Diamond\!\!\!\!-\alpha$ be in a node with a label n in a branch B. If α appears in a proper successor node x with a label $m \geq n$, and $(n, m) \in min_B$, we say $\Diamond\!\!\!\!-\alpha$ at the position n is fulfilled in m.*

The truth value $\Diamond\!\!\!\!-\alpha$ in a temporal state n depends on α being true on a future temporal state m and m being minimal to all temporal states that come after n w.r.t. $\prec_B$. We say m is minimal to n as shorter way to say that m is minimal to all temporal states that come after n. Unfulfilled non-monotonic eventualities in a node x with the label n are represented by the set $une \stackrel{\text{def}}{=} \{(n_1, \Diamond\!\!\!\!-\alpha_1), (n_2, \Diamond\!\!\!\!-\alpha_2), \dots\}$, each pair $(n_k, \Diamond\!\!\!\!-\alpha_k)$ represents a non-monotonic eventuality $\Diamond\!\!\!\!-\alpha_k$ at a position n_k that needs to be fulfilled. Therefore each node x has three components: n is a label indicating the temporal state, Γ is the set of sentences within the node and *une* is the set of non-monotonic eventualities at x that need to be fulfilled. With all of our notions introduced, here is the rule for the $\Diamond\!\!\!\!-$ operator:

$$(\Diamond\!\!\!\!-) \quad \frac{n : (\{\Diamond\!\!\!\!-\alpha_1\} \uplus \Sigma, une), \prec_B, min_B}{n : (\{\alpha_1\} \cup \Sigma, une), \prec_B, min_B \cup \{(n, n)\} \mid n : (\Sigma, une \cup \{(n, \Diamond\!\!\!\!-\alpha_1)\}), \prec_B, min_B}$$

For the rule $(\Diamond\!\!\!\!-)$, we explore two outcomes. The first outcome is when the non-monotonic eventuality $\Diamond\!\!\!\!-\alpha_1$ at n is fulfilled in n. We then add α_1 to the set of sentences Γ of the leaf node and add $(n, n) \in min$ of the branch. The second outcome is when $\Diamond\!\!\!\!-\alpha_1$ is not fulfilled in n, then we add the pair to $(n, \Diamond\!\!\!\!-\alpha_1)$ to *une* of the leaf node as a non-monotonic eventuality that needs to be fulfilled. Example 8 shows the application of $[\Diamond\!\!\!\!-]$ rule.

▶ **Example 8.** Let a branch B have $\prec_B$, min_B and a leaf node $5 : (\{p, q, \Box(p \wedge q), \Diamond\!\!\!\!-r\}, \emptyset)$. After applying $(\Diamond\!\!\!\!-)$ rule on $\Diamond\!\!\!\!-r$, we have two new branches B_1 and B_2. The branch B_1 has a leaf node where the sentence r is in Γ of the leaf node and $(5, 5) \in min_{B_1}$. The branch B_2 has $(5, \Diamond\!\!\!\!-r)$ in *une* of the leaf node.

$$\begin{array}{c} 5 : (\{p, q, \Box(p \wedge q), \Diamond\!\!\!\!-r\}, \emptyset), \prec_B, min_B \\ \diagup \qquad\qquad \diagdown \\ 5 : (\{p, q, \Box(p \wedge q), r\}, \emptyset), \prec_B, min_B \cup \{(5, 5)\} \qquad 5 : (\{p, q, \Box(p \wedge q)\}, \{(5, \Diamond\!\!\!\!-r)\}), \prec_B, min_B \end{array}$$

The next static rule we discuss is the rule $(\boldsymbol{une})$. Let n, n' be two labels such that $n' < n$, for each label n and a pair $(n', \Diamond\!\!\!\!-\alpha_1)$, the rule (une) is applied one and only one time. The rule goes as follows:

$$(une) \quad \frac{n : (\Gamma, \{(n', \Diamond\!\!\!\!-\alpha_1)\} \uplus U), \prec_B, min_B}{n : (\{\alpha_1\} \cup \Gamma, U), \prec_B, min_B \cup \{(n', n)\} \mid}$$

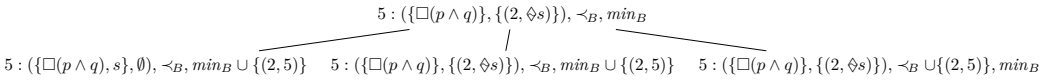
$$\frac{}{n : (\Gamma, \{(n', \Diamond\!\!\!\!-\alpha_1)\} \cup U), \prec_B, min_B \cup \{(n', n)\} \mid}$$

$$\frac{}{n : (\Gamma, \{(n', \Diamond\!\!\!\!-\alpha_1)\} \cup U), \prec_B \cup \{(n', n)\}, min_B}$$

For the rule $(\boldsymbol{une})$, we explore three outcomes. The first outcome is when $\Diamond\!\!\!\!-\alpha_1$ at the position n' is fulfilled at n. We remove $(n', \Diamond\!\!\!\!-\alpha_1)$ from *une*, then we add α in Γ of the leaf node and (n', n) in *min* of the branch. In the second and third branches, we explore the

outcome of $\Diamond\alpha_1$ not being fulfilled yet in n, we keep the pair $(n', \Diamond\alpha_1)$ on the leaves of two branches. The second branch explore the outcome of n being minimal to n' w.r.t. to $\prec$ of the branch. We then add (n', n) to the *min* of the branch. In the third branch, we explore the outcome of n not being minimal to n' w.r.t. $\prec$ of the branch. It means that there exists a temporal state m' that come after n' where m' is preferred to n w.r.t. to $\prec$ of the branch, we add the pair (n', n) in $\prec$ of the branch to represent this case. It is worth to mention that the rule (*une*) does not apply when the label of the node n is the same as $(n, \Diamond\alpha_1)$. The reason behind this is that we have already explored the case when the eventuality is fulfilled in n thanks to ($\Diamond$) rule. Example 9 shows the application of (*une*) rule.

▶ **Example 9.** Let a branch B have $\prec_B$, min_B and a leaf node $5 : (\{\Box(p \wedge q)\}, \{(2, \Diamond s)\})$. After the application of *une* on $(2, \Diamond s)$, we have three branches B_1, B_2 and B_3. B_1 has the sentence s in Γ of its leaf node, it has also $(2, 5)$ in min_{B_1}. B_2 keeps $(2, \Diamond s)$ in the *une* of its leaf node, with $(2, 5) \in min_{B_2}$. B_3 keeps also $(2, \Diamond s)$ in *une* of its leaf node, with $(2, 5) \in \prec_{B_3}$.

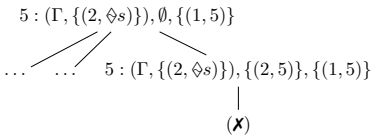


With the (*une*) and ($\Diamond$) introduced, we need to check the consistency of $\prec$ of all the new branches. We apply this check each time we apply (*une*) or ($\Diamond$) rule. Let $B := (\langle x_0, x_1, x_2, \dots \rangle, \prec_B, min_B)$ be a branch, the rule goes as follows:

- [$\prec$**-inconsistency**] If $(n, n') \in min_B$ and there exists $n'' \geq n$ s.t. $(n'', n') \in \prec_B$, then the branch is crossed (✗).

In a branch B, if $(n, n') \in min_B$, then we are currently exploring a branch where n' is minimal to n w.r.t. $\prec_B$. Therefore there should be no $n'' \geq n$ where $(n'', n) \in \prec_B$. Each time we explore a branch where this inconsistency arises, we close the branch.

▶ **Example 10.** Let B be a branch where $\prec_B$ is empty, min_B has $(1, 5)$ in it, and a leaf node $5 : (\Gamma, \{(2, \Diamond s)\})$. After applying *une* rule on $(2, \Diamond s)$, we have three branches B_1, B_2 and B_3. The relation $\prec_{B_1}$ is empty, and min_{B_1} has the pairs $(1, 5)$ and $(2, 5)$. In this case, there is no inconsistency w.r.t. $\prec_{B_1}$ so far. The same goes for B_2. However, we add $(2, 5)$ to $\prec_{B_3}$. Since we already have $(1, 5) \in min_{B_3}$, we then cannot have $(2, 5) \in \prec_{B_3}$. We close B_3.



In a branch B of a tree $\mathcal{T}$ with a leaf node x_i, after applying every static rule aforementioned (the order of application these rules is non-deterministic) that can be applied, all leaf nodes of the generated branches contain only sentences of the form $p, \neg p$ or $\bigcirc\alpha$ in their Γ. When no more static rules can be applied in a node, this node is called a *state-labelled node*. State-labelled nodes mark the full expansion of all sentences that hold in a state n.

Once we are in a state-labelled node, in order to go from a temporal state to the next, we need a transition rule (a rule to go from a temporal state n to the next $n + 1$). In a branch B with a leaf state-labelled node, the rule **transition** goes the following way:

$$\text{(Transition)} \quad \frac{n : (\{\bigcirc\alpha_1, \bigcirc\alpha_2, \bigcirc\alpha_3, \ldots, \bigcirc\alpha_k\} \uplus \Sigma, \mathit{une}), \prec_B, \mathit{min}_B}{n+1 : (\{\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_k\}, \mathit{une}), \prec_B, \mathit{min}_B}$$

After the **transition** rule is applied to a state-labelled node $n : (\Gamma, \mathit{une})$, we add a node with the label $n+1$. It marks the start of a new temporal state $n+1$. We carry over to $n+1$ only sentences within the scope of $\bigcirc\alpha_i$ sentences. The set une gets transferred as well to the next temporal state. Any pair $(n', \Diamond\!\!\!\!-\alpha_1) \in \mathit{une}$ remaining in the state node with the label n indicates that the rule (une) was applied on the temporal state n and the current branch explores an outcome where $\Diamond\!\!\!\!-\alpha_1$ is not yet fulfilled in n. Therefore, these non-monotonic eventualities need to be fulfilled in $n'' \geq n+1$.

Before applying the **transition** rule, we need to add a set of checks to prevent branches from expanding indefinitely. These checks are called **loop** and **prune** rules. These rules, together with the **transition** rule, are called *dynamic rules*.

Let $B := (\langle x_0, x_1, x_2, \ldots, v\rangle, \prec_B, \mathit{min}_B)$ be a branch where v is a state-labelled node $n : (\Gamma_v, \mathit{une}_v)$. Let u be the last state-labelled node $n-1 : (\Gamma_u, \mathit{une}_u)$ that comes before v in the branch B. Before applying the transition rule on v, we check for these rules:

- **[Loop]** Let v be a state-labelled node such that it has at least one sentence of the form $\bigcirc\Box\alpha_{bool}$ in Γ_v but has no $\bigcirc\alpha_{bool}, \bigcirc\Diamond\beta, \bigcirc\Diamond\!\!\!\!-\beta$ in Γ_v and $\mathit{une}_v = \emptyset$. If for all $\bigcirc\Box\alpha_{bool}$ in Γ_v, there exists $u < s \leq v$ such that $\Box\alpha_{bool} \in \Gamma_s$, then the branch B is ticked (✓).

Notice that once an eventuality is fulfilled, it does not appear any longer in the successors of the node. In this case, we say that the sentence is *consumed*. On the other hand, sentences of the form $\Box\alpha_{bool}$ never get consumed and get replicated indefinitely. Once a branch has no eventuality left, $\Box\alpha_{bool}$ sentences give rise to an infinite tableau with repetitive nodes. Nevertheless, we can represent this by looping nodes of the last temporal state. We can, in this case, stop the branch from ever going infinite. The **loop** rule states that when the leaf state node v has no eventualities (classical or non-monotonic), has only $\bigcirc\Box\alpha_{bool}$ as sentences with the pattern $\bigcirc$, and each $\bigcirc\Box\alpha_{bool}$ is a result from applying the $\Box$ rule to a node in B with label n, the branch is ticked and marked as a successful branch.

- **[Prune]** Let $u < v$ be two consecutive state-labelled nodes s.t. $\Gamma_v = \Gamma_u$ and $\mathit{une}_v = \mathit{une}_u$ and that there is at least one eventuality in x_u (either $\bigcirc\Diamond\beta \in \Gamma_u$ or $(n', \Diamond\!\!\!\!-\beta) \in \mathit{une}_u$), then the branch is crossed (✗).

The **prune** rule states that when the last two state nodes u and v have the same set of classical and non-monotonic eventualities that need to be fulfilled, and there is at least one eventuality in u, the branch is then crossed and marked as an unsuccessful branch. Any branch that does not fulfill at least one eventuality between the current and the last temporal state is closed, to prioritize the exploration of branches that fulfill one or more eventuality of the last temporal state. If neither **prune** or **loop** apply on v, we apply the **transition** rule on the node v. Note that the **loop** and **prune** rules are fundamentally different from the ones proposed in Reynolds' tableau [14]. These rules are tailored to the restrictions of the fragment $\mathcal{L}_1$, in particular, the restriction of not allowing temporal sentences inside the $\Box$ operator. We argue in this paper that when eventualities (either classical or non-monotonic) are not infinitely replicated inside *globally* operators, we only need to check the current state node with the last one that comes beforehand. It is the reason why we also omit also the operator $\mathcal{U}$, since the right part of a $\mathcal{U}$-sentence can also replicate eventualities.

Once we are in a state-labelled node, we check for the loop and prune within the branch before applying the transition rule. If the transition rule is applied on a state node with a label n, we obtain a new node with the label $n+1$. We can then expand the tree from this

node by applying static rules until we find ticked branches (thanks to the **empty** rule), closed branches (thanks to the **contradiction** or $\prec$**-inconsistency** rules), or branches with a state node that has the label $n+1$. We then repeat the cycle between static and dynamic rules. We can see that the tableau method does not go indefinitely. Thanks to prune rule, we close any branch (✗) that does not fulfill any eventuality in the current temporal state. Anytime we apply a transition rule (from n to $n+1$), we need to fulfill at least one eventuality in n. Therefore, as long as a branch is not closed with prune rule, eventuality sentences (either classical or non-monotonic) get consumed one by one over the execution of the method. Thus any branch that is not closed with prune has no eventualities left to fulfill. Note that if a branch contains at least one sentence of the form $\Box\alpha_{bool}$, it is then ticked thanks to the loop rule ($\Box\alpha_{bool}$ sentences do not get consumed). Otherwise, it is ticked thanks to the empty rule. Therefore any tableau $\mathscr{T}$ for a sentence in $\mathcal{L}_1$ is a *saturated* tableau.

4 Soundness and completeness

4.1 Soundness

Here we prove that the tableau method is sound, that is, when a tableau $\mathscr{T}$ of a sentence $\alpha \in \mathcal{L}_1$ has a successful branch, then α is satisfiable. As a first step, we show that we can extract an interpretation $I \in \mathfrak{I}$ from the successful branch. Let $B := (\langle x_0, x_1, x_2, \ldots, x_n, (\checkmark)\rangle, \prec_B, min_B)$ be a successful branch of a tableau $\mathscr{T}$ for α, the sequence of nodes contains normal and state-labelled nodes. Each state-labelled node, denoted by x_{j_i}, within this sequence has a distinct label i. Figure 1 shows an example of the branch B.

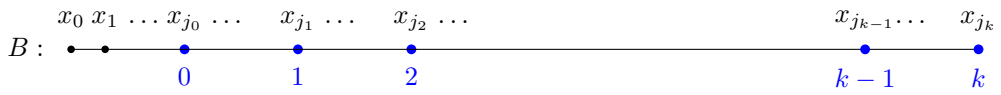


Figure 1 Illustration of the branch B.

From the aforementioned branch B, we can build an interpretation $I_B = (V, \prec)$. In this section, k denotes the label of the last state node. The function V is defined as follows:

$$V(i) := \begin{cases} \{p \in \mathscr{P} \mid p \in \Gamma_{x_{j_i}}\}, & \text{if } 0 \leq i \leq k; \\ V(k), & \text{otherwise.} \end{cases}$$

The ordering relation $\prec$ is defined as follows $\prec := \{(n, n') \mid (n, n') \in \prec_B\}$. We can see that $\prec$ is irreflexive, since there is no $(n, n) \in \prec_B$. The relation $\prec$ is also transitive, since for all (n_1, n_2) and (n_2, n_3) in $\prec_B$, there is no $(n_3, n_1) \in \prec_B$. Finally, since $\prec_B$ has no infinitely descending chains, then we can conclude that $\prec$ preserves the well-foundness condition over $\mathbb{N}$. Therefore the interpretation $I_B \in \mathfrak{I}$.

With the model construction introduced, we can move on to the second part of the proof of soundness. We need to show that the model I satisfies the sentence α. In order to do so, we introduce a mapping function, denoted by Δ_B, that links each time point $i \in \mathbb{N}$ to a set of sentences that are true in said i. These sentences come from the branch B. Depending on how the branch is ticked, the function Δ_B is defined in the following way.

If the branch was ticked with the empty rule:

$$\Delta_B(i) := \begin{cases} \bigcup_{x_0 \leq x \leq x_{j_0}} \Gamma_x, & \text{if } i = 0; \\ \bigcup_{x_{j_{i-1}} < x \leq x_{j_i}} \Gamma_x, & \text{if } 1 \leq i \leq k-1; \\ \{\}, & \text{otherwise.} \end{cases}$$

If the branch was ticked with the loop rule:

$$\Delta_B(i) := \begin{cases} \bigcup_{x_0 \leq x \leq x_{j_0}} \Gamma_x, \text{ if } i = 0; \\ \bigcup_{x_{j_{i-1}} < x \leq x_{j_i}} \Gamma_x, \text{ if } 1 \leq i \leq k; \\ \Delta_B(k), \text{ otherwise.} \end{cases}$$

For a time point $0 \leq i \leq k$, $\Delta_B(i)$ contains the set of all sentences in Γ of the node between the two consecutive state nodes $x_{j_{i-1}}$ and x_{j_i}, $x_{j_{i-1}}$ not included. If B is ticked thanks to the empty rule, then $\Delta_B(i)$ is empty for all $i \geq k$. If B is ticked thanks to the loop rule, then $\Delta_B(i)$ has the same set of sentences as $\Delta_B(k)$ for all $i \geq k$. We can show next that if a sentence α_1 is in $\Delta_B(i)$, then $I_B, i \models \alpha_1$. In what follows, let B be a successful branch of a tableau $\mathscr{T}$, let k be the label of the last state node in B, and let I_B, Δ_B be the interpretation and the mapping function of sentences extracted from B.

▶ **Lemma 11.** *Let B be a successful branch, and $i \in \mathbb{N}$. If $\bigcirc\alpha_1 \in \Delta_B(i)$, then $\alpha_1 \in \Delta_B(i+1)$.*

▶ **Lemma 12.** *Let B be a successful branch, and $i \in \mathbb{N}$. If $\Box\alpha_1 \in \Delta_B(i)$, then for all $f \geq i$, we have $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(f)$.*

▶ **Lemma 13.** *Let B be a successful branch, and $i \in \mathbb{N}$. If $\Diamond\alpha_1 \in \Delta_B(i)$, then there exists $d \geq i$ s.t. $\alpha_1 \in \Delta_B(d)$ and for all $i \leq f < d$, we have $\{\Diamond\alpha_1, \bigcirc\Diamond\alpha_1\} \subseteq \Delta_B(f)$.*

Lemma 11 to 13 are analogous to Reynolds' method [14]. Their proof are in Appendix A.

▶ **Proposition 14.** *Let B be a successful branch. If $(i, i') \in min_B$, then there is no $i \leq i''$ where $(i'', i') \in \prec_B$.*

Proof. Let B be a successful branch s.t. $(i, i') \in min_B$. Since the branch is successful, then it is not closed with $\prec$-inconsistency and therefore there is no $i \leq i''$ where $(i'', i') \in \prec_B$. ◀

▶ **Lemma 15.** *Let B be a successful branch and $0 \leq i \leq k$. If $\Diamond\!\!\!\!-\alpha_1 \in \Delta_B(i)$, then there exists $d \geq i$ s.t. $(i, d) \in min_B$ and $\alpha_1 \in \Delta_B(d)$.*

Proof. Let B be a ticked branch of the tableau, k be the label of the last state node and $i \in \mathbb{N}$. We discuss two possibilities:

- When the branch B is ticked with empty rule, whenever $\Diamond\!\!\!\!-\alpha_1 \in \Delta_B(i)$, then we have $0 \leq i \leq k-1$. Since $\Diamond\!\!\!\!-\alpha_1 \in \Delta_B(i)$, then $\Diamond\!\!\!\!-\alpha_1 \in \Gamma_x$ where $x_{j_{i-1}} < x \leq x_{j_i}$. Let x be the node where we apply the rule $(\Diamond\!\!\!\!-)$ on $\Diamond\!\!\!\!-\alpha_1$, then we either have α_1 in Γ of the next node with $(i, i) \in min_B$ or we have $(i, \Diamond\!\!\!\!-\alpha_1) \in une$ of the next node. If α_1 is in Γ of the next node, then the lemma holds. If $(i, \Diamond\!\!\!\!-\alpha_1) \in une$ of the next node, then we find $(i, \Diamond\!\!\!\!-\alpha_1) \in une_{x_{j_i}}$. Thanks to the transition rule, we have $(i, \Diamond\!\!\!\!-\alpha_1) \in une_{x_{j_i+1}}$. By applying the rule *une* on a node with the label $i+1$,then we either have α_1 in Γ of the next node with $(i, i+1) \in min_B$ or we have $(i, \Diamond\!\!\!\!-\alpha_1) \in une$ (the two remaining branches) of the next node. In a similar way as in i, we can conclude that either $\alpha_1 \in \Delta_B(i+1)$ with $(i, i+1) \in min_B$ (the lemma holds) or $(i, \Diamond\!\!\!\!-\alpha_1) \in une_{x_{j_{i+1}}}$. Without loss of generality, $(i, \Diamond\!\!\!\!-\alpha_1)$ is in $une_{x_{j_f}}$ for $i \leq f \leq k-1$ unless we find $i \leq d \leq f$ with $\alpha_1 \in \Delta_B(d)$ and $(i, d) \in min_B$. Since the branch is closed thanks to the empty rule, it means that $(i, \Diamond\!\!\!\!-\alpha_1) \notin une_{x_{j_{k-1}}}$. Therefore, there is a state $i \leq d \leq k-1$ where $\alpha_1 \in \Delta_B(d)$ with $(i, d) \in min_B$.
- When the branch B is ticked with loop rule, the proof is analogous to the case of the empty rule (notice that we also have $(i, \Diamond\!\!\!\!-\alpha_1) \notin une_{x_{j_k}}$). ◀

▶ **Theorem 16.** *Let B be a ticked branch from a saturated tableau, and $I_B = (V, \prec)$ be the model built from the branch B. For all $\alpha \in \mathcal{L}_1$, for all $i \geq 0$, if $\alpha \in \Delta_B(i)$ then $I_B, i \models \alpha$.*

Proof. We prove this lemma using structural induction on the size of the sentence α. Let B be a successful branch for a tableau $\mathscr{T}$, and $I_B = (V, \prec)$ be the model built from B.

- $\alpha = p$. Let $p \in \Delta_B(i)$. By construction of the model I_B, we have $p \in V(i)$. Therefore, we have $I_B, i \models p$.
- $\alpha = \neg p$. Let $\neg p \in \Delta_B(i)$. Since B is a ticked branch, then it was not closed with the contradiction rule, therefore we have $p \notin V(i)$. Therefore, we have $I_B, i \models \neg p$.
- $\alpha = \alpha_1 \wedge \alpha_2$. Let $\alpha_1 \wedge \alpha_2 \in \Delta_B(i)$. By $\wedge$-rule, we have $\alpha_1, \alpha_2 \in \Delta_B(i)$. By induction hypothesis on α_1, α_2, we have $I_B, i \models \alpha_1$ and $I_B, i \models \alpha_2$. Thus, we have $I_B, i \models \alpha_1 \wedge \alpha_2$.
- $\alpha = \alpha_1 \vee \alpha_2$. Let $\alpha_1 \vee \alpha_2 \in \Delta_B(i)$. By $\vee$-rule, we either have α_1 or α_2 in $\Delta_B(i)$. Suppose that $\alpha_1 \in \Delta_B(i)$, by induction hypothesis on α_1, we have $I_B, i \models \alpha_1$. Therefore, we have $I_B, i \models \alpha_1 \vee \alpha_2$. Same reasoning applies when $\alpha_2 \in \Delta_B(i)$.
- $\alpha = \bigcirc\alpha_1$. Let $\bigcirc\alpha_1 \in \Delta_B(i)$. Thanks to Lemma 11, we have $\alpha_1 \in \Delta_B(i+1)$. By induction hypothesis on α_1, we have $I_B, i+1 \models \alpha_1$. Therefore, we have $I_B, i \models \bigcirc\alpha_1$.
- $\alpha = \Box\alpha_1$. Let $\Box\alpha_1 \in \Delta_B(i)$. Thanks to Lemma 12, we have $\alpha_1 \in \Delta_B(f)$ for all $f \geq i$. By induction hypothesis on α_1, we have $I_B, f \models \alpha_1$ for all $f \geq i$. Therefore, we have $I_B, i \models \Box\alpha_1$.
- $\alpha = \Diamond\alpha_1$. Let $\Diamond\alpha_1 \in \Delta_B(i)$. Thanks to Lemma 13, we have $\alpha_1 \in \Delta_B(d)$ for some $d \geq i$. By induction hypothesis on α_1, we have $I_B, d \models \alpha_1$. Therefore, we have $I_B, i \models \Diamond\alpha_1$.
- $\alpha = \diamondsuit\alpha_1$. Let $\diamondsuit\alpha_1 \in \Delta_B(i)$. Depending on where i is, we have two cases:
  - In the case of $i > k$, since $\diamondsuit\alpha_1 \in \Delta_B(i)$, then we have $\diamondsuit\alpha_1 \in \Delta_B(k)$. Furthermore, since the branch is ticked with loop rule, we know that $(i, \diamondsuit\alpha_1) \notin une_{x_{j_k}}$. Therefore $\alpha_1 \in \Delta_B(k)$, thus $\alpha_1 \in \Delta_B(i)$. Furthermore, since $\prec := \prec_B$, and there is no $f \geq i$ such $(f, i) \in \prec_B$, then $i \in min_{\prec}(i)$, and therefore, $I_B, i \models \diamondsuit\alpha_1$.
  - $0 \leq i \leq k$. Thanks to Lemma 15, there exists $d \geq i$ s.t. $\alpha_1 \in \Delta_B(d)$ and $(i, d) \in min_B$. By induction hypothesis on α_1, we have $I_B, d \models \alpha_1$. Thanks to Proposition 14, there is no $i \leq f \leq k$ where $(f, d) \in \prec_B$ and therefore there is no $i \leq f \leq k$ where $(f, d) \in \prec$. Furthermore, by the construction of the model I_B, there is no $f \geq k$ where $(f, d) \in \prec$. Therefore, we have $d \in \min_{\prec}(i)$. Thus, we have $I_B, i \models \diamondsuit\alpha_1$. ◀

Let $\alpha \in \mathcal{L}_1$, B be a ticked branch from a saturated tableau for α, $I_B = (V, \prec)$ be a model built from B. Since we have $\alpha \in \Delta_B(0)$, then we have $I_B, 0 \models \alpha$.

4.2 Completeness

We conclude this paper by proving the completeness of the tableau method for sentences in $\mathcal{L}_1$ i.e., if a sentence α is satisfiable, then any tableau for α has a successful branch, no matter the order of applying the rules. We use a model I for α to find a ticked node.

▶ **Theorem 17.** *Let $\alpha \in \mathcal{L}_1$ be a satisfiable sentence of $LTL^{\sim}$. Then any tableau for α has a successful branch.*

The idea behind this proof is to have an intermediate sequence s that serves as a link between an interpretation I that satisfies the sentence α and a tableau $\mathscr{T}$ for α. The sequence s is a tuple $s := (\langle x_0, x_1, x_2, \dots \rangle, \prec_s, min_s)$ where each x_i is a pair (Γ, une), $\prec_s, min_s$ are the set of constraints that the sequence s must follow in order to be coherent with $\prec$ of the interpretation. The set $\prec_s$ is not an ordering relation, it records instances of points of time not being minimal to other points of time w.r.t. the ordering relation $\prec$. Remember

that each when we apply the *une* rule, we add a pair (n', n) to $\prec$ in order to symbolize the outcome of n not being minimal to n'. The set of min_s records the instances of points of points of time being minimal to other points of time w.r.t. the ordering relation $\prec$.

We link each node of the sequence x_i to a time point $J(x_i)$ of the interpretation I and a labelled node $f(x_i)$ of the tableau $\mathscr{T}$. Depending on I, we can build the sequence s using the tableau, we then show the sequence s ends up with a tick (✓). We make sure that for each node x_i with the index time point $J(x_i)$ of the sequence, we have the following invariant:

$$Inv(x_i, J(x_i)) \begin{cases} \text{For each } \alpha \in \Gamma_{x_i}, \text{ we have } I, J(x_i) \models \alpha; \\ \text{For each } (J_1, \Diamond\alpha_1) \in une_{x_i}, \text{ there exists } J_2 \geq J(x_i) \text{ where} \\ J_2 \in min_{\prec}(J_1) \text{ and } I, J_2 \models \alpha_1; \\ \text{For each } (J_1, J_2) \in min_s, \text{ we have } J_2 \in min_{\prec}(J_1); \\ \text{For each } (J_1, J_2) \in \prec_s, \text{ there exists } J_3 \geq J_1 \text{ s.t. } (J_3, J_2) \in \prec \\ \text{(in other words } J_2 \notin min_{\prec}(J_1)). \end{cases}$$

We start by putting the root node $0 : (\{\alpha\}, \emptyset)$ with the index time point $J(x_0) := 0$ at the start of the sequence. For the first node x_0 with the index time point 0 (since there is no rule applied before the root node, the sets min_s and $\prec_s$ are empty at the start), we have $I, 0 \models \alpha$. Therefore the invariant $Inv(x_0, 0)$ holds. Suppose that the invariant holds up to x_i, and a rule was applied to x_i, we then add a new node x_{i+1} to the sequence depending on which outcome of the rule represents the interpretation I. We then move to the outcome node in the tableau, and see which rule is applied to it, and so on and so forth. Each time we add a new node x_{i+1} to the sequence s, we need to make sure that the invariant $Inv(x_{i+1}, J(x_{i+1}))$ holds. In general, the sequence will head from the parent node to a child node but it might occasionally jump backwards (only in the case of the parent being a prune node, more on that later). It is worth to point out that since we might be jumping back and forth between nodes of $\mathscr{T}$, each time we are add a new node x_{i+1} to the sequence s, we are going to rename labels within the sets une_x, $\prec_B$ and min_B by their respective indexed time points J. The function f links each node x_i of the sequence s to a labelled node $f(x_i)$ of the tableau $\mathscr{T}$. It is worth to mention that, since we are only renaming labels of other sets, then we have $\Gamma_{x_i} = \Gamma_{f(x_i)}$. In Appendix B, we discuss the case of each rule that is applied to x_i.

5 Conclusion

We introduced the basis for a tableau method for $LTL^{\sim}$. We showed how preferential semantics work in a one-pass tree-shaped tableau. We also established semantic rules for the $\Diamond$ operator. We showed how to handle non-monotonic eventualities using *une*, $\prec_B$ and min_B. In the end, we proved that our method is sound and complete. The loop/prune checkers proposed in this paper are specific to $\mathcal{L}_1$, and work well under these restrictions.

With the foundation laid in this work, the next step is to establish semantic rules for the $\boxminus$ operator. The next fragment of $LTL^{\sim}$ that we are investigating is the sub-language that allows only Boolean sentences within $\Box$ and $\boxminus$. We conjecture that the satisfiability of this fragment is decidable and has an upper bound model property similar to one that we published in [6].

References

1 M. Ben-Ari. *Mathematical Logic for Computer Science, third edition*. Springer, 2012.

2 K. Britz, T. Meyer, and I. Varzinczak. Preferential reasoning for modal logics. *Electronic Notes in Theoretical Computer Science*, 278:55–69, 2011. doi:10.1016/j.entcs.2011.10.006.

3 K. Britz, T. Meyer, and I. Varzinczak. Semantic foundation for preferential description logics. In *AI 2011: Advances in Artificial Intelligence*, pages 491–500, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

4 K. Britz and I. Varzinczak. From KLM-style conditionals to defeasible modalities, and back. *Journal of Applied Non-Classical Logics*, 28(1):92–121, 2018. doi:10.1080/11663081.2017.1397325.

5 K. Britz and I. Varzinczak. Preferential tableaux for contextual defeasible $\mathcal{ALC}$. In *Proceedings of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, number 11714 in LNCS, pages 39–57, 2019.

6 A. Chafik, F. Cheikh Alili, J.-F. Condotta, and I. Varzinczak. On the decidability of a fragment of preferential LTL. In *27th International Symposium on Temporal Representation and Reasoning, TIME 2020*, volume 178 of *LIPIcs*, pages 19:1–19:19, 2020. doi:10.4230/LIPIcs.TIME.2020.19.

7 D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer, 1987. doi:10.1007/3-540-51803-7_36.

8 L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato. Preferential description logics. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, number 4790 in LNAI, pages 257–272. Springer, 2007.

9 L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato. Analytic tableaux calculi for KLM logics of nonmonotonic reasoning. *ACM Transactions on Computational Logic*, 10(3):18:1–18:47, 2009.

10 Y. Kesten, Z. Manna, H. Mcguire, and A. Pnueli. A decision algorithm for full propositional temporal logic. *LNCS*, 697, April 1999.

11 S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.

12 Z. Manna and A. Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

13 A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 46–57, October 1977. doi:10.1109/SFCS.1977.32.

14 M. Reynolds. A new rule for LTL tableaux. *Electronic Proceedings in Theoretical Computer Science*, 226:287–301, September 2016. doi:10.4204/eptcs.226.20.

15 M. Reynolds. A traditional tree-style tableau for LTL, 2016. arXiv:1604.03962.

16 K. Y. Rozier and M. Y. Vardi. LTL satisfiability checking. In *Model Checking Software*, pages 149–167, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

17 S. Schwendimann. A new one-pass tableau calculus for PLTL. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 277–291. Springer Berlin Heidelberg, 1998.

18 Y. Shoham. *A Semantical Approach to Nonmonotonic Logics*, page 227–250. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

19 Y. Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, 1988.

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

21 P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.

22 P. Wolper. The tableau method for temporal logic. *Logique Et Analyse*, 28:110–111, January 1985.

A Soundness proof

▶ **Lemma 11.** *Let B be a successful branch, and $i \in \mathbb{N}$. If $\bigcirc\alpha_1 \in \Delta_B(i)$, then $\alpha_1 \in \Delta_B(i+1)$.*

Proof. Let B be a ticked branch of the tableau, k be the label of the last node and $i \in \mathbb{N}$. We discuss two possibilities:

- When the branch B is ticked with empty rule. We can see that when $i \geq k$, $\Delta_B(i) = \{\}$ and therefore $\bigcirc\alpha_1 \notin \Delta_B(i)$. We also know that since $\Delta_B(k) = \{\}$, then there is no $\bigcirc\alpha_1 \in \Gamma_{x_{j_{k-1}}}$. Furthermore, there is no static rule that removes $\bigcirc\alpha_1$, we can conclude that there is no $\bigcirc\alpha_1 \in \Delta_B(k-1)$.
  Otherwise, in the case of $0 \leq i < k-1$, if $\bigcirc\alpha_1 \in \Delta_B(i)$, then $\bigcirc\alpha_1 \in \Gamma_x$ where $x_{j_{i-1}} < x \leq x_{j_i}$. Since there is no static rule that removes $\bigcirc\alpha_1$, we have $\bigcirc\alpha_1 \in \Gamma_{x_{j_i}}$. Furthermore, after applying the transition rule on the node x_{j_i}, we have $\alpha_1 \in \Gamma_{x_{j_i+1}}$. Thus, we have $\alpha_1 \in \Delta_B(i+1)$.
- When the branch B is ticked with loop rule. In the case of $0 \leq i < k$, the proof is analogous to the case of empty rule. When $i = k$, if $\bigcirc\alpha_1 \in \Delta_B(k)$, then $\bigcirc\alpha_1$ is subsequently in $\Gamma_{x_{j_k}}$. Since B is ticked with loop, then α_1 is a sentence of the form $\Box\alpha_{bool}$ and $\Box\alpha_{bool} \in \Gamma_x$ $(x_{j_{k-1}} < x \leq x_{j_k})$ and therefore $\Box\alpha_{bool} \in \Delta_B(k)$. Moreover, we have $\Delta_B(k) = \Delta_B(k+1)$. Therefore, we have $\Box\alpha_{bool} \in \Delta_B(k+1)$ and thus $\alpha_1 \in \Delta_B(k+1)$.
  In the case where $i \geq k$. If $\bigcirc\alpha_1 \in \Delta_B(i)$, then $\bigcirc\alpha_1 \in \Delta_B(k-1)$. As mentioned before, since $\bigcirc\alpha_1 \in \Delta_B(k-1)$, then α_1 is $\Box\alpha_2$ and $\Box\alpha_2 \in \Delta_B(k-1)$. Since $\Box\alpha_2 \in \Delta_B(k-1)$, then $\Box\alpha_2 \in \Delta_B(i+1)$ and therefore $\alpha_1 \in \Delta_B(i+1)$. ◀

▶ **Lemma 12.** *Let B be a successful branch, and $i \in \mathbb{N}$. If $\Box\alpha_1 \in \Delta_B(i)$, then for all $f \geq i$, we have $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(f)$.*

Proof. Let B be a ticked branch of the tableau, k be the label of the last node and $i \in \mathbb{N}$.

For all $0 \leq i \leq k$, whenever $\Box\alpha_1 \in \Delta_B(i)$, then both α_1 and $\bigcirc\Box\alpha_1$ is in $\Delta_B(i)$. By Lemma 11, since $\bigcirc\Box\alpha_1 \in \Delta_B(i)$, then we have $\Box\alpha_1 \in \Delta_B(i+1)$. By successive applications of Lemma 11, we have $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(f)$ for all $i \leq f \leq k$. Note that in the case of a branch ticked with empty rule, since $\Delta_B(k) = \{\}$, $\Box\alpha_1$ cannot be in any $\Delta_B(i)$ where $0 \leq i \leq k$. In other words, if a branch contains $\Box\alpha_1$, it can only be ticked with loop rule.

Since $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(k)$, and for all $f \geq k$, we have $\Delta_B(f) = \Delta_B(k)$, then $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(f)$. Thus, the lemma holds when $0 \leq i \leq k$.

In the case of $i > k$, since $\Box\alpha_1 \in \Delta_B(i)$ and $\Delta_B(i) = \Delta_B(k-1)$. Thanks to $\Box$-rule, $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(k-1)$. Thus, we have $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(f)$ for all $f \geq k$ and subsequently $\{\alpha_1, \Box\alpha_1, \bigcirc\Box\alpha_1\} \subseteq \Delta_B(f)$ for all $f \geq i$. ◀

▶ **Lemma 13.** *Let B be a successful branch, and $i \in \mathbb{N}$. If $\Diamond\alpha_1 \in \Delta_B(i)$, then there exists $d \geq i$ s.t. $\alpha_1 \in \Delta_B(d)$ and for all $i \leq f < d$, we have $\{\Diamond\alpha_1, \bigcirc\Diamond\alpha_1\} \subseteq \Delta_B(f)$.*

Proof. Let B be a ticked branch of the tableau, k be the label of the last node and $i \in \mathbb{N}$. We discuss two possibilities:

- When the branch B is ticked with empty rule. In the case of $0 \leq i \leq k-1$, whenever $\Diamond\alpha_1 \in \Delta_B(i)$, then either α_1 is in $\Delta_B(i)$ or $\bigcirc\Diamond\alpha_1$ is in $\Delta_B(i)$. If $\alpha_1 \in \Delta_B(i)$, the lemma holds. Otherwise, by Lemma 11, if $\bigcirc\Diamond\alpha_1 \in \Delta_B(i)$ then $\Diamond\alpha_1 \in \Delta_B(i+1)$. By successive applications of Lemma 11, $\{\Diamond\alpha_1, \bigcirc\Diamond\alpha_1\}$ is in $\Delta_B(f)$ for $i \leq f \leq k-1$, unless we find $i \leq d \leq f$ with $\alpha_1 \in \Delta_B(d)$.
  It remains to show that there is a time point d where $\alpha_1 \in \Delta_B(d)$. Since the branch is closed thanks to the empty rule, it means that $\bigcirc\Diamond\alpha_1 \notin \Delta_B(k-1)$. Therefore, there is a state $i \leq d \leq k-1$ where $\alpha_1 \in \Delta_B(d)$.

- When the branch B is ticked with loop rule and in the case of $0 \leq i \leq k$, the proof is analogous to the case of empty rule (notice that $\bigcirc\Diamond\alpha_1 \notin \Delta_B(k)$ also in the case of branches ticked with loop).
  In the case of $i > k$, since $\Diamond\alpha_1 \in \Delta_B(i)$, then we have $\Diamond\alpha_1 \in \Delta_B(k-1)$. Furthermore, since the branch is ticked with loop rule, we know that $\bigcirc\Diamond\alpha_1 \notin \Delta_B(k)$. Therefore $\alpha_1 \in \Delta_B(k)$, thus $\alpha_1 \in \Delta_B(i)$. ◀

B Completeness proof

Proof. In this section, suppose that we build the sequence s up to x_i and the invariant holds for all the nodes in the sequence.

[**Empty, Loop**]: If we end up with a ticked node in the sequence s, the theorem holds.

[**Contradiction**]: If the sequence s is closed, then we have p and $\neg p$ in Γ_{x_i}. Since we have $Inv(x_i, J(x_i))$, then we $I, J(x_i) \models p$ and $I, J(x_i) \models \neg p$. This cannot happen in a interpretation $I \in \mathfrak{I}$.

[$\wedge$]: Suppose that the rule $\wedge$ is applied to the sentence $\alpha_1 \wedge \alpha_2$ on the node $f(x_i)$ of the tableau $\mathscr{T}$. Let y be the child node of the node $f(x_i)$ in the branch. We have $\Gamma_y = (\Gamma_{f(x_i)} \setminus \{\alpha_1 \wedge \alpha_2\}) \cup \{\alpha_1, \alpha_2\}$. We define the next node in the sequence x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_y$, $une_{x_{i+1}} = une_{x_i}$, and the sets $min_s, \prec_s$ remain unchanged. Since we have $Inv(x_i, J(x_i))$ and $\alpha_1 \wedge \alpha_2 \in \Gamma_{x_i}$, then $I, J(x_i) \models \alpha_1$ and $I, J(x_i) \models \alpha_2$. For the node x_{i+1}, we have $\Gamma_{x_{i+1}} = (\Gamma_{x_i} \setminus \{\alpha_1 \wedge \alpha_2\}) \cup \{\alpha_1, \alpha_2\}$ and $une_{x_{i+1}} = une_{x_i}$. Therefore the first and second conditions of $Inv(x_{i+1}, J(x_i))$ are met. Moreover, since $min_s, \prec_s$ remain unchanged and we have $Inv(x_i, J(x_i))$, then the third and forth conditions of $Inv(x_{i+1}, J(x_i))$ are met. Consider that $J(x_{i+1}) = J(x_i)$, the invariant $Inv(x_{i+1}, J(x_i))$ holds.

We can see that by applying a static rule of the from $(\wedge, \vee, \Box, \Diamond)$ on the node $f(x_i)$, we do not add in either une, $\prec_B$ or min_B while applying these rules nor add a new non-monotonic eventuality to be fulfilled in the outcome nodes. In order to lighten the proof, we skip the check for the second, third and fourth conditions of Inv up until $\Diamond$ and une rules.

[$\vee$]: Suppose that the rule $\vee$ is applied to the sentence $\alpha_1 \vee \alpha_2$ on the node $f(x_i)$ of the tableau $\mathscr{T}$. We obtain two children nodes y and z of $f(x_i)$. We have $\Gamma_y = (\Gamma_{f(x_i)} \setminus \{\alpha_1 \vee \alpha_2\}) \cup \{\alpha_1\}$ and $\Gamma_z = (\Gamma_{f(x_i)} \setminus \{\alpha_1 \vee \alpha_2\}) \cup \{\alpha_2\}$. Since we have $Inv(x_i, J(x_i))$, and $\alpha_1 \vee \alpha_2 \in \Gamma_{x_i}$, then we either have $I, J(x_i) \models \alpha_1$ or $I, J(x_i) \models \alpha_2$. Consider that $J(x_{i+1}) = J(x_i)$, we discuss two cases:

- Case 1: If $I, J(x_i) \models \alpha_1$, then we define the next node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_y$ and $une_{x_{i+1}} = une_{x_i}$. We know that $\Gamma_{x_{i+1}} = (\Gamma_{x_i} \setminus \{\alpha_1 \vee \alpha_2\}) \cup \{\alpha_1\}$. Therefore for all $\gamma \in \Gamma_{x_{i+1}}$, we have $I, J(x_i) \models \gamma$. Thus, the invariant $Inv(x_{i+1}, J(x_i))$ holds.
- Case 2: Otherwise, when $I, J(x_i) \models \alpha_2$, then we define the node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_z$ and $une_{x_{i+1}} = une_{x_i}$. We know that $\Gamma_{x_{i+1}} = (\Gamma_{x_i} \setminus \{\alpha_1 \vee \alpha_2\}) \cup \{\alpha_2\}$. Therefore for all $\gamma \in \Gamma_{x_{i+1}}$, we have $I, J(x_i) \models \gamma$. Thus, the invariant $Inv(x_{i+1}, J(x_i))$ holds.

[$\Diamond$]: Suppose that the rule $\Diamond$ is applied to the sentence $\Diamond\alpha_1$ on the node $f(x_i)$ of the tableau $\mathscr{T}$. We obtain two children nodes y and z of $f(x_i)$. We have $\Gamma_y = (\Gamma_{f(x_i)} \setminus \{\Diamond\alpha_1\}) \cup \{\alpha_1\}$ and $\Gamma_z = (\Gamma_{f(x_i)} \setminus \{\Diamond\alpha_1\}) \cup \{\bigcirc\Diamond\alpha_1\}$. Since we have $Inv(x_i, J(x_i))$, and $I, J(x_i) \models \Diamond\alpha_1$, then we have $I, J(x_i) \models \alpha_1 \vee \bigcirc\Diamond\alpha_1$. Therefore, we either have $I, J(x_i) \models \alpha_1$ or $I, J(x_i) \models \bigcirc\Diamond\alpha_1$. Consider that $J(x_{i+1}) = J(x_i)$, we discuss two cases:

- Case 1: If $I, J(x_i) \models \alpha_1$, then we define the next node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_y$ and $une_{x_{i+1}} = une_{x_i}$. We know that $\Gamma_{x_{i+1}} = (\Gamma_{x_i} \setminus \{\Diamond\alpha_1\}) \cup \{\alpha_1\}$. Therefore for all $\gamma \in \Gamma_{x_{i+1}}$, we have $I, J(x_i) \models \gamma$. Thus, the invariant $Inv(x_{i+1}, J(x_i))$ holds.

- Case 2: When $I, J(x_i) \models \bigcirc\Diamond\alpha_1$, then we define the next node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_z$ and $une_{x_{i+1}} = une_{x_i}$. We know that $\Gamma_{x_{i+1}} = (\Gamma_{x_i} \setminus \{\Diamond\alpha_1\}) \cup \{\bigcirc\Diamond\alpha_1\}$. Therefore for all $\gamma \in \Gamma_{x_{i+1}}$, we have $I, J(x_i) \models \gamma$. Thus, the invariant $Inv(x_{i+1}, J(x_i))$ holds.

[$\Box$]: Suppose that the rule $\Box$ is applied to the sentence $\Box\alpha_1$ on the node $f(x_i)$ of the tableau $\mathscr{T}$. Let y be the child node of the node $f(x_i)$ in the branch. We have $\Gamma_y = (\Gamma_{f(x_i)} \setminus \{\Box\alpha_1\}) \cup \{\alpha_1, \bigcirc\Box\alpha_1\}$. We define the next node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_y$ and $une_{x_{i+1}} = une_{x_i}$ and $I, J(x_i) \models \Box\alpha_1$, then we have $I, J(x_i) \models \alpha_1 \wedge \bigcirc\Box\alpha_1$. Therefore, we have $I, J(x_i) \models \alpha_1$ and $I, J(x_i) \models \bigcirc\Box\alpha_1$. We know that $\Gamma_{x_{i+1}} = (\Gamma_{x_i} \setminus \{\Box\alpha_1\}) \cup \{\alpha_1, \bigcirc\Box\alpha_1\}$. Therefore for all $\gamma \in \Gamma_{x_{i+1}}$, we have $I, J(x_i) \models \gamma$. Consider that $J(x_{i+1}) = J(x_i)$, the invariant $Inv(x_{i+1}, J(x_i))$ holds.

[$\Diamond$]: When the rule [$\Diamond$] is applied to $\Diamond\alpha_1$ on the node $f(x_i)$ of $\mathscr{T}$, we explore two outcomes. Let n be the label of the node $f(x_i)$ in the branch. In the first outcome, we have a child y with $\Gamma_y = (\Gamma_{f(x_i)} \setminus \{\Diamond\alpha_1\}) \cup \{\alpha_1\}$ and (n, n) in min of the branch. In the second outcome, we have a child node z with $\Gamma_z = (\Gamma_{f(x_i)} \setminus \{\Diamond\alpha_1\})$ and $une_z = une_{f(x_i)} \cup (n, \Diamond\alpha_1)$. Since we have $Inv(x_i, J(x_i))$, and $\Diamond\alpha_1 \in \Gamma_{x_i}$, then we have $I, J(x_i) \models \Diamond\alpha_1$. It means that there exists $J_1 \geq J(x_i)$ s.t. $J_1 \in min_{\prec}(J(x_i))$ and $I, J_1 \models \alpha_1$. Consider that $J(x_{i+1}) = J(x_i)$, we discuss two cases:

- Case 1: If $J_1 = J(x_i)$, then we have $I, J(x_i) \models \alpha_1$ and $J(x_i) \in min_{\prec}(J(x_i))$. We then define the next node x_{i+1} of the sequence with $\Gamma_{x_{i+1}} = \Gamma_y$, $une_{x_{i+1}} = une_{x_i}$ and add the pair $(J(x_i), J(x_i))$ to min_s. Notice that we swap the labels of nodes with the position of their indexed time point $J(x_i)$, we will be using indexed time point J instead of labels throughout this proof. We know that $\Gamma_{x_{i+1}} = (\Gamma_{x_i} \setminus \{\Diamond\alpha_1\}) \cup \{\alpha_1\}$ with $I, J(x_i) \models \alpha_1$. Additionally, we have $min_s := min_s \cup \{(J(x_i), J(x_i))\}$ with $J(x_i) \in min_{\prec}(J(x_i))$. The sets $une_{x_{i+1}}, \prec_s$ remains unchanged. Therefore, the invariant $Inv(x_{i+1}, J(x_i))$ holds.
- Case 2: when $J_1 > J(x_i)$, then we define the next node x_{i+1} of the sequence with $\Gamma_{x_{i+1}} = \Gamma_z$, $une_{x_{i+1}} = une_{x_i} \cup \{(J(x_i), \Diamond\alpha_1)\}$. We also know that $J_1 > J(x_i)$ and $J_1 \in min_{\prec}(J(x_i))$ and $I, J_1 \models \alpha_1$. Therefore, the second condition of $Inv(x_{i+1}, J(x_i))$ holds on the pair $(J(x_i), \Diamond\alpha_1)$. The sets min_s and $\prec_s$ remain unchanged. The invariant $Inv(x_{i+1}, J(x_i))$ holds.

[***une***]: When the rule [une] is applied on a pair $(n_1, \Diamond\alpha_1)$ in une of $f(x_i)$. Let n be the label of the node $f(x_i)$. Let x be the predecessor of x_i in s where the rule [$\Diamond$] was applied on $\Diamond\alpha_1$, let $J(x)$ be the indexed time point of x. Note that the label of $f(x)$ is n_1. In the first outcome, we have a child y where $\Gamma_y = \Gamma_{f(x_i)} \cup \{\alpha_1\}$, $une_y = une_{f(x_i)} \setminus \{(n_1, \Diamond\alpha_1)\}$ and (n_1, n) in min of the branch. In the second outcome, we have a child z where $\Gamma_z = \Gamma_{f(x_i)}$, $une_z = une_{f(x_i)}$ and (n_1, n) in min of the branch. In the third outcome, we have a child v where $\Gamma_v = \Gamma_{f(x_i)}$, $une_v = une_{f(x_i)}$ and (n_1, n) in $\prec$ of the branch.

On the other hand, since x is a predecessor of x_i in s, then we have $Inv(x, J(x))$. Furthermore, since we have $(n_1, \Diamond\alpha_1) \in une_{f(x_i)}$, it means that when the rule [$\Diamond$] is applied on the node $f(x)$, the branch where $(n_1, \Diamond\alpha_1) \in une_{f(x+1)}$ is the path that corresponds with the interpretation I. By [$\Diamond$] rule, since we have $Inv(x+1, J(x+1))$, $(n_1, \Diamond\alpha_1) \in une_{f(x+1)}$ and we know that $J(x+1) = J(x)$, then we have $(J(x), \Diamond\alpha_1) \in une_{x+1}$. Furthermore, since no rule application consumed $(n_1, \Diamond\alpha_1)$ up to $f(x_i)$, then the pair $(J(x), \Diamond\alpha_1)$ remains also in une_{x_i}. Also, we have $Inv(x_i, J(x_i))$, then there is $J' \geq J(x_i)$ where $J' \in min_{\prec}(J(x))$ and $I, J' \models \alpha_1$. Consider that $J(x_{i+1}) = J(x_i)$, we discuss all possibilities below:

- Case 1: If $J' = J(x_i)$, then we have $J(x_i) \in min_{\prec}(J(x))$ and $I, J(x_i) \models \alpha_1$. We define the next node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_y$, $une_{x_{i+1}} = une_{x_i} \setminus \{(J(x), \Diamond\alpha_1)\}$ and add $(J(x), J(x_i))$ to min_s. We have $\Gamma_{x_{i+1}} = \Gamma_{x_i} \cup \{\alpha_1\}$ with $I, J(x_i) \models \alpha_1$. Additionally, we have $(J(x), J(x_i)) \in min_s$ with $J(x_i) \in min_{\prec}(J(x))$. The set $\prec_s$ remains unchanged. Thus, the invariant $Inv(x_{i+1}, J(x_i))$ holds.

- Case 2: when $J' > J(x_i)$, we have two possibilities:
  - Case 2.1: If $J(x_i) \in min_{\prec}(J(x))$, then we define the next node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_z$, $une_{x_{i+1}} = une_{x_i}$ and add $(J(x), J(x_i))$ to min_s. We have $(J(x), J(x_i)) \in min_s$ with $J(x_i) \in min_{\prec}(J(x))$. The sets $\Gamma_{x_{i+1}}$, $une_{x_{i+1}}$ and $\prec_s$ remain unchanged. Thus, the invariant $Inv(x_{i+1}, J(x_i))$ holds.
  - Case 2.2: If $J(x_i) \notin min_{\prec}(n_1)$, then there exists $J'' \geq J(x)$ s.t. $(J'', J(x_i)) \in \prec$. We define the next node x_{i+1} with $\Gamma_{x_{i+1}} = \Gamma_v$, $une_{x_{i+1}} = une_{x_i}$ and add $(J(x), J(x_i))$ to $\prec_s$. We have $(J(x), J(x_i)) \in \prec_s$ with $J(x_i) \notin min_{\prec}(n_1)$. The sets $\Gamma_{x_{i+1}}$, $une_{x_{i+1}}$ and min_s remain unchanged. Thus, the invariant $Inv(x_{i+1}, J(x_i))$ holds.

[**Transition**]: Suppose that the transition rule is applied on the state node $f(x_i)$. Let y be the child node of the node x_i in the branch. We have $\Gamma_y = \{\alpha_1 \mid \bigcirc\alpha_1 \in \Gamma_{f(x_i)}\}$ and $une_y = une_{f(x_i)}$. We define the next node x_{i+1} in s with $\Gamma_{x_{i+1}} = \Gamma_y$ and $une_{x_{i+1}} = une_{x_i}$. We consider that $J(x_{i+1}) = J(x_i) + 1$.

Since we have $Inv(x_i, J(x_i))$, then for all $\bigcirc\alpha_1 \in \Gamma_{x_i}$, we have $I, J(x_i) \models \bigcirc\alpha_1$ and therefore $I, J(x_i) + 1 \models \alpha_1$. The first condition of the invariant $Inv(x_{i+1}, J(x_i) + 1)$ is met.

Secondly, since x_i is a state node, then for each remaining $(n_1, \Diamond\alpha_1) \in une_{f(x_i)}$, either the rule $[\Diamond]$ or $[une]$ was applied to a node $f(x_i')$ with the index $J(x_i') = J(x_i)$ and $(n_1, \Diamond\alpha_1)$ was carried over to $f(x_i)$. In both rules, for each $(n_1, \Diamond\alpha_1) \in une_{f(x_i)}$, we have $(J(x_1), \Diamond\alpha_1) \in une_{x_i}$ s.t. $f(x_1)$ is the node where the rule $[\Diamond]$ was applied to $\Diamond\alpha_1$ (see Case 2 for $[\Diamond]$ and $[une]$ rules). Furthermore, since we have $Inv(x_i, J(x_i))$ and $f(x_i)$ is a state node, then for each $(J(x_1), \Diamond\alpha_1) \in une_{x_i}$, there exists $J_2 > J(x_i)$ where $J_2 \in min_{\prec}(J(x_1))$ and $I, J_2 \models \alpha_1$. Without loss of generality, there exists $J_2 \geq J(x_i)+1$ where $J_2 \in min_{\prec}(J(x_1))$ and $I, J_2 \models \alpha_1$. The second condition of the invariant $Inv(x_{i+1}, J(x_i)+1)$ is met. Since min_s and $\prec_s$ remain unchanged, the invariant $Inv(x_{i+1}, J(x_i) + 1)$ holds.

[$\prec$**-inconsistency**]: Suppose that the $\prec$-inconsistency rise on the node $f(x_i)$, and let n be the label of the $f(x_i)$ on the branch B. If this inconsistency rises, we have (n_1, n) in min_B and (n_2, n) in $\prec_B$ where $n_1 \leq n_2 \leq n$. These two pairs come from applying $[\Diamond]$ or $[une]$ rule on two predecessors $f(x), f(x')$ of $f(x_i)$ with the same label n and the same indexed time point $J(x) = J(x') = J(x_i)$.

Let J_1 be the time point corresponding to the node $f(x_1)$ with the label n_1, and let J_2 be the time point corresponding to the node $f(x_2)$ with the label n_2. It is worth to mention that $J_1 \leq J_2 \leq J(x_i)$. Since x, x' are predecessors of x, we have $Inv(x, J(x))$, $Inv(x', J(x'))$ and $Inv(x_i, J(x_i))$. Therefore, we the rules are applied on x and x', we end up with $(J_1, J(x_i)) \in min_s$ and $(J_2, J(x_i)) \in \prec_s$. Since $(J_1, J(x_i)) \in min_s$, then we have $J(x_i) \in min_{\prec}(J_1)$. On the other hand, since $(J_2, J(x_i)) \in \prec_s$, then there exists $J_3 \geq J_2$ s.t. $(J_3, J(x_i)) \in \prec$. Moreover, we have $J_1 \leq J_2$, this entails that there exists $J_3 \geq J_1$ s.t. $(J_3, J(x_i)) \in \prec$. This contradicts Definition 4 of minimality w.r.t. to the relation $\prec$. Therefore this cannot happen in a interpretation $I \in \mathfrak{I}$.

[**Prune**]: Let $f(x_i)$ be a state node where the prune condition is met. There is a sequence within s that goes the following way, $x_h = u, x_{h+1}, x_{h+2}, \ldots, v = x_i$. The node u or x_h is the state node that comes before x_i and the node v is the current state node. Since v is a prune node, we have $\Gamma_v = \Gamma_u$ and $une_u = une_v$. We can see that if we apply the transition rule to the node x_i, we will have $\Gamma_{x_{i+1}} = \Gamma_{x_{h+1}}$ and $une_{x_{i+1}} = une_{x_{h+1}}$. Therefore, we can proceed with the construction of s as if x_i was linked to $f(u)$ instead of $f(v)$. Thanks to the transition, since we have $Inv(x_u, J(x_u))$, then we have $Inv(x_{i+1}, J(x_i) + 1)$.

Each time we find a pair (u, v) in the sequence s, we call it a *jump*. These jumps may occur once or many times (and it may go infinite) in s. In a sequence s, if a pair (u, v) jumps repeatedly in succession, we call the pair a *recurring jump*. It is worth to point out that,

each time we jump backwards because of a node closed with prune, we return to the state labelled node that comes before. In general, the sequence s explores one branch B of $\mathscr{T}$, and it deviates sometime to a prune node and goes back to B. Furthermore, since no eventuality is fulfilled within a prune loop, eventualities and their fulfillment are in the same branch B.

What we showed so far is that for an interpretation I and its corresponding sequence s, we have $Inv(x_i, J(x_i))$ for each $i \geq 0$. Going back to the start of the proof, we need to prove that the sequence finishes with a ticked node (such is the case when we end up in [loop] or [empty] node). We can see that if the sequence s is on a [prune] node, we jump back to the state node that comes before it. Theoretically, this jump can recur infinitely many times. This means that sequence goes infinite on this case (and never find a ticked node). We need to prove that this case cannot happen in the sequence s of I. Suppose that is the case, that means the last jump (u_k, v_k) in the sequence s is a recurring jump that goes infinitely many times. The jumps (u_j, v_j) that come before may recur many times but not infinitely many times (otherwise, (u_k, v_k) would not be the last jump). In the recurring jump (u_k, v_k), no eventuality is fulfilled (whether it is classical or non-monotonic). This entails that when we are in a parent node $u_k < x_l < v_k$ that applies either $[\Diamond]$ or $[une]$ rule, we move to the child node that delays the propagation of the eventuality (we are in Case 2 for both rules).

It is worth to point out that we have at least one eventuality in u_k. Let us take $\bigcirc\Diamond\alpha_1 \in \Gamma_{u_k}$ for example, since we have $Inv(u_k, J(u_k))$, that means that $I, J(u_k) \models \bigcirc\Diamond\alpha_1$. Thus, we take the *first* time point $J_{\alpha_1} > J(u_k)$ s.t. $I, J_{\alpha_1} \models \alpha_1$. We also have $I, J_{\alpha_1} \models \Diamond\alpha_1$. On the other hand, for all $J(u_k) < J < J_{\alpha_1}$, we have $I, J \models \Diamond\alpha_1$ $I, J \models \bigcirc\Diamond\alpha_1$. In other words, each time we encounter $\Diamond\alpha_1 \in \Gamma_{x_{l-1}}$ within our jumps (keep in mind we have $Inv(x_{l-1}, J)$), we pick the node in Case 2 of the $[\Diamond]$ rule i.e., $\bigcirc\Diamond\alpha_1 \in \Gamma_{x_l}$. However, in the node indexed with J_{α_1}, when we encounter $\Diamond\alpha_1 \in \Gamma_{x_{l'-1}}$ (keep in mind we have $Inv(x_{l'-1}, J_{\alpha_1})$), we pick the node in Case 1 of the $[\Diamond]$ rule i.e., $\alpha_1 \in \Gamma_{x'_l}$. This raises a contradiction, because the node $x_{l'}$ is not present within the jump (u_k, v_k). Thus breaking the infinite recurring jump (u_k, v_k).

If the eventuality is a non-monotonic one, namely $(J_1, \Diamond\alpha_1) \in une_{u_k}$. Since we have $Inv(u_k, J(u_k))$ with u_k being a state node, there exists $J' > J(u_k)$ s.t. $J' \in min_{\prec}(J_1)$ and $I, J' \models \alpha_1$. Let J_{α_1} be the first time point that met these criteria. For all $J(u_k) < J < J_{\alpha_1}$, each time we encounter $(J_1, \Diamond\alpha_1) \in une_{x_{l-1}}$ with the index J, we have $J_{\alpha_1} > J$, $J_{\alpha_1} \in min_{\prec}(J_1)$ and $I, J_{\alpha_1} \models \alpha_1$. Therefore, we pick Case 2 of $[une]$ rule i.e., $(J_1, \Diamond\alpha_1) \in une_{x_l}$. However, when we encounter $(J_1, \Diamond\alpha_1) \in une_{x_{l'-1}}$ with the index J_{α_1}, we have $J_{\alpha_1} \in min_{\prec}(J_1)$ and $I, J_{\alpha_1} \models \alpha_1$, then we pick the node in Case 1 of $[une]$ rule i.e., $\alpha_1 \in x_{l'}$. This raises a contradiction, because the node $x_{l'}$ is not present within the jump (u_k, v_k).

We proved that since $I, 0 \models \alpha$, then the corresponding sequence s cannot finish on a contradiction, $\prec$-inconsistency or a prune jump. Therefore it must finish with a ticked node. Hence, the tableau $\mathscr{T}$ of α has a ticked node and therefore a successful branch. ◀

$1\frac{1}{2}$-Player Stochastic StopWatch Games

Sparsa Roychowdhury
Indian Institute of Technology Bombay, Mumbai, India

Abstract

Stochastic timed games (STGs), introduced by Bouyer and Forejt, generalize continuous-time Markov chains and timed automata. Depending on the number of players – 2, 1, or 0 – subclasses of stochastic timed games are classified as $2\frac{1}{2}$-player, $1\frac{1}{2}$-player, and $\frac{1}{2}$-player games where the $\frac{1}{2}$ symbolizes the presence of the stochastic player. The qualitative and quantitative reachability problem for STGs was studied in [10] and [1]. In this paper, we introduce stochastic stopwatch games (SSG), an extension of (STG) from clocks to stopwatches. We focus on $1\frac{1}{2}$-player SSGs and prove that with two variables which can be either a clock or a stopwatch, qualitative reachability is decidable, whereas, if we increase the number of variables to three, with at least one stopwatch, the problem becomes undecidable.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases Timed Automata, Stopwatches, Stochastic Timed Games

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.17

Acknowledgements The author thanks Prof. Krishna S. of IIT Bombay, India for insightful discussions, suggestions and encouragement towards this work. The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Two-player zero-sum games are well studied for controller synthesis of discrete event systems. But, they are not sufficient to model real-time probabilistic systems. To model real-time systems, one needs to capture the semantics of time. Timed automata [4] are a well-known and extensively studied formalism widely used to model real timed systems. Timed automata models time with a finite set of real valued variables known as clocks. The reachability problem of timed automata is showed to be PSPACE-Complete using a special abstraction known as region automata [4]. But, only clocks are not sufficient to model stochastic behaviors of a system. In [7] probabilistic semantics were added in timed automata where choices of time and transitions are randomized. The probabilistic notion was mostly used to solve the "almost sure model checking" [8] of timed automata i.e., to check if a property is satisfied with some certainty or not. Different formalisms like probabilistic timed automata [18], continuous probabilistic timed automata [17], continuous timed Markov chains [6], and stochastic timed automata [9] have been proposed that capture both of the real-time and stochastic nature of the system. Timed games [5] are a natural extension of timed automata to model interactive systems in a more robust manner. Stochastic timed games (STG in short) was proposed in [10], which extends timed games with probabilities. Just like timed games, the locations are partitioned among players but in STG there is a special player known as the "environment". A player can only process their move if she is in a location that belongs to her. The player "environment" is different from other players in the sense that it can choose delays and transitions stochastically based on a distribution. Hybrid automata [3] is a powerful formalism that uses more generalized real valued variables to model hybrid systems. Unlike clocks, the value of the variables in a hybrid automaton changes depending on a function defined on the locations. These functions can be linear as well as non-linear. But, most of the interesting problems of hybrid automata like reachability are

undecidable [14]. Hybrid automata have been studied extensively under different constraints like time-bounded hybrid automata [12], initialized hybrid automata [20], singular hybrid automata [16], etc. A hybrid automaton is linear if its constraints can be expressed as linear expressions over the set of variables [2]. A stopwatch [13] is a real-valued variable that can either track time like clocks or can choose to stay in the current value depending on the current location. It is shown in [13] that a minor upgrade from timed automata to stopwatch automata immediately yields the full expressive power of linear hybrid automata.

In case of stochastic systems reachability, one can associate a probability parameter p with reachability query to ask, "is a state reachable from another state with probability p?". The parameter p is called threshold probability. Depending on the value of threshold probability, we can have two different types of reachability queries,

Quantitative Reachability: When the constraint on the threshold is $0 < p < 1$,

Qualitative Reachability: When the constraint on the threshold is $p \in \{0, 1\}$.

It is known that [10], the qualitative reachability problem is decidable for $1\frac{1}{2}$-player stochastic timed games with one clock, and quantitative reachability is undecidable for $2\frac{1}{2}$-player stochastic timed games with ≥ 3 clocks. These results were further refined in [1], where it was shown that the qualitative reachability problem is undecidable for $1\frac{1}{2}$-STGs with four or more clocks, and the same problem is undecidable for $2\frac{1}{2}$-STGs for three or more clocks.

Just as stopwatches generalizes clocks, we generalize stochastic timed games to stochastic stopwatch games (SSG in short) by replacing clocks with stopwatches. We solve the qualitative reachability problem on this extended model. Our focus in this paper is only on qualitative reachability and $1\frac{1}{2}$ player games on SSG. We keep the case of two and a half player qualitative reachability for future work. Our main results are,

(1) The qualitative reachability problem is EXPTIME-Complete for $1\frac{1}{2}$ player stochastic stopwatch games with two stopwatches.

(2) The qualitative reachability problem for $1\frac{1}{2}$ player stochastic stopwatch games is undecidable (Π^0_1 hard) with three stopwatches.

Our results give a tight demarcation between decidability and undecidability in the case of $1\frac{1}{2}$ player SSGs.

2 Preliminaries

We use standard notations for the set of reals ($\mathbb{R}$), rationals ($\mathbb{Q}$), and natural numbers ($\mathbb{N}$), and add subscripts to indicate additional constraints (for instance $\mathbb{R}_{\geq 0}$ is for the set of non-negative reals). Let $\mathcal{X}$ be a finite set of real-valued variables called *clocks*. A *valuation* on $\mathcal{X}$ is a function $\nu : \mathcal{X} \to \mathbb{R}_{\geq 0}$. We assume an arbitrary but fixed ordering on the clocks and write x_i for the clock with order i. This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{R}^{|\mathcal{X}|}_{\geq 0}$. For a subset of clocks $R \subseteq \mathcal{X}$ and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, we write $\nu[R]$ for the valuation where $\nu[R](x) = 0$ if $x \in R$, and $\nu[R](x) = \nu(x)$ otherwise. For $t \in \mathbb{R}_{\geq 0}$, write $\nu + t$ for the valuation defined by $\nu(x) + t$ for all $x \in \mathcal{X}$. The valuation $\mathbf{0} \in \mathbb{R}^{|\mathcal{X}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{X}$. A constraint (or guard) over $\mathcal{X}$ is a subset of $\mathbb{R}^{|\mathcal{X}|}_{\geq 0}$ defined by a (finite) conjunction of constraints of the form $x \bowtie k$, where $k \in \mathbb{N}$, $x \in \mathcal{X}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. We write $\mathsf{rect}(\mathcal{X})$ for the set of constraints on $\mathcal{X}$. For a constraint $\varphi \in \mathsf{rect}(\mathcal{X})$, and a valuation ν, we write $\nu \models \varphi$ to represent the fact that valuation ν satisfies constraint φ (defined in a natural way).

▶ **Definition 1** (Timed Automata [4]). *A timed automaton is a tuple $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{X}, \Delta, F)$ where $\mathcal{Q}$ is a finite set of locations, $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is a set of initial locations, $\mathcal{X}$ is a finite set of clocks, $F \subseteq \mathcal{Q}$ is a set of accepting locations and Δ is a set of transitions of the form (l_1, φ, R, l_2) where, $l_1, l_2 \in \mathcal{Q}$, $R \subseteq \mathcal{X}$ is known as the set of reset clocks, and $\varphi \in \mathsf{rect}(\mathcal{X})$.*

A state s of timed automata is a pair $s = (l, \nu) \in (\mathcal{Q} \times \mathbb{R}^{|\mathcal{X}|}_{\geq 0})$ consists of a location and valuation. A transition (t, e) from a state $s = (l, \nu)$ to a state $s' = (l', \nu')$ is written as $s \xrightarrow{t,e} s'$ if $e = (l, \varphi, R, l') \in \Delta$, such that $\nu + t \models \varphi$, and $\nu' = (\nu + t)[R](x)$.

A run is a finite or infinite sequence of transitions $\rho = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \ldots$ of states and transitions. An edge e is enabled from s whenever there is a state s' such that $s \xrightarrow{0,e} s'$. Given a state s of $\mathcal{A}$ and an edge e, we define $\mathcal{E}(s, e) = \{t \in \mathbb{R}_{\geq 0} \mid s \xrightarrow{t,e} s'\}$ for some s' and $\mathcal{E}(s) = \bigcup_{e \in \Delta} \mathcal{E}(s, e)$. We say that $\mathcal{A}$ is non-blocking if and only if for all states s, $\mathcal{E}(s) \neq \emptyset$.

▶ **Definition 2** (Singular Stopwatch Automata). *A Singular Stopwatch automaton is a tuple* $\mathcal{H} = (\mathcal{Q}, \mathcal{Q}_0, V, \Delta, \mathfrak{R}, F)$ *where,*

- $\mathcal{Q}$ *is a finite set of locations including a distinguished initial set of locations* $\mathcal{Q}_0 \subseteq \mathcal{Q}$,
- V *is an (ordered) set of variables called stopwatches,*
- $\Delta \subseteq \mathcal{Q} \times \mathit{rect}(V) \times 2^V \times \mathcal{Q}$ *is the set of* transitions *of the form* (l, φ, R, l') *such that,* $l, l' \in \mathcal{Q}$, $\varphi \in \mathit{rect}(V)$, *and* $R \subseteq V$ *subset of variables that are reset during the transition.*
- $\mathfrak{R} : \mathcal{Q} \to \{0, 1\}^{|V|}$ *is the location dependent* flow function *characterizing the rate of each variable in each location.*
- $F \subseteq \mathcal{Q}$ *is the set of final locations.*

A variable $x \in V$ is a clock when for all locations $l \in \mathcal{Q}$, $\mathfrak{R}(l)[x] = 1$, and a variable is a stopwatch when, for all locations $l \in \mathcal{Q}$, $\mathfrak{R}(l)[x] \in \{0, 1\}$. Just like timed automata the map $\nu : V \to \mathbb{R}_{\geq 0}$ represents the current valuation of the variables, we define a state of $\mathcal{H}$ as a pair of locations and valuations $(l, \nu) \in (\mathcal{Q} \times \mathbb{R}^{|V|}_{\geq 0})$. For a state $s = (l, \nu)$ of $\mathcal{H}$ and $t \in \mathbb{R}_{\geq 0}$ we define $s + t = (l, \nu + t)$ where, $(\nu + t)(x) = \nu(x) + \mathfrak{R}(l)[x] \cdot t, \forall x \in V$. A transition (t, e) of $\mathcal{H}$ from a state $s = (l, \nu)$ to a state $s' = (l', \nu')$ is written as $s \xrightarrow{t,e} s'$ if $e = (l, \varphi, R, l') \in \Delta$, such that $\nu + t \models \varphi$, and $\nu' = (\nu + t)[R](x)$. A run of $\mathcal{H}$ is $\rho = (l_0, \nu_0) \xrightarrow{t_1,e_1} (l_1, \nu_1) \xrightarrow{t_2,e_2} \ldots$ of states and transitions. The notions of non blocking states, and $\mathcal{E}(s) = \bigcup_{e \in \Delta} \mathcal{E}(s, e)$ for all states carry over as in timed automata.

Singular Stopwatch automata are a special case of singular hybrid automata [16], when the variables are stopwatches. We now formally define the stochastic stopwatch games in the same line of stochastic timed games as defined in [10].

▶ **Definition 3** ($1\frac{1}{2}$-Stochastic Stopwatch Games ($1\frac{1}{2}$-SSG)). *A* $1\frac{1}{2}$*-player stochastic stopwatch game is a tuple* $\mathcal{SG} = (\mathcal{H}, \mathcal{Q}_\Diamond, \mathcal{Q}_\bigcirc, w, \mu)$ *where* $\mathcal{H} = (\mathcal{Q}, \mathcal{Q}_0, V, \Delta, \mathfrak{R}, F)$ *is a singular stopwatch automaton,* $(\mathcal{Q}_\Diamond, \mathcal{Q}_\bigcirc)$ *is a partition of* $\mathcal{Q}$ *such that they are controlled by players* $\Diamond$ *and* $\bigcirc$ *respectively,* w *is a map that assigns weight to each transition leaving* $\mathcal{Q}_\bigcirc$*, and* μ *is a function that assigns a measure over* $\mathcal{E}(s)$ *for every state* $s \in \mathcal{Q}_\bigcirc \times \mathbb{R}^{|\mathcal{X}|}_{\geq 0}$*. The function* $\mu(s)$ *satisfies the following properties,*

(1) $\mu(s)(\mathcal{E}(s)) = 1$ *(Law of total probability)*

(2) *Let* λ *be the Lebesgue measure, if* $\lambda(\mathcal{E}(s)) > 0$ *then for each measurable set* $B \subseteq \mathcal{E}(s)$ *we have* $\lambda(B) = 0$ *if and only if* $\mu(s)(B) = 0$*. The choice of measures is such that the measures evolve smoothly while moving from one state to another.*

The singular stopwatch automaton $\mathcal{H}$ is equipped with uniform distributions over delays if for every state s, $\mathcal{E}(s)$ is bounded, and $\mu(s)$ is the uniform distribution over $\mathcal{E}(s)$. $\mathcal{H}$ is equipped with exponential distributions over delays whenever, for every state s, either $\mathcal{E}(s)$ has Lebesgue measure zero, or $\mathcal{E}(s) = \mathbb{R}_{\geq 0}$ and for every location l, there is a positive rational α_l such that $\mu(s)(\mathcal{E}) = \int_{t \in \mathcal{E}} \alpha_l e^{-\alpha_l t} dt$. We assume $\alpha_l = 1$ for all locations l. The locations in $\mathcal{Q}_\Diamond$ are controlled by player $\Diamond$. The locations in $\mathcal{Q}_\bigcirc$ are governed by probabilistic laws. For $s \in \mathcal{Q}_\bigcirc \times \mathbb{R}^{|V|}_{\geq 0}$, both delays and discrete moves will be chosen probabilistically: from s, a delay d is chosen following the probability distribution over delays $\mu(s)$. Then, from state

$s+d$, an enabled edge is selected following a discrete probability distribution that is given in a usual way with the weight function w: in state $s+t$, the probability of edge e (if enabled), denoted $p(s+t)(e)$ is $\frac{w(e)}{\sum_{e'\{w(e')|e' \text{ is enabled in } s+t\}}}$. This way of probabilizing behaviours in timed automata has been presented in [10]. We refer to $\ell \in \mathcal{Q}_{\bigcirc}$ as stochastic nodes and $\ell \in \mathcal{Q}_{\Diamond}$ as diamond ($\Diamond$) nodes.

Strategies. Let $\rho = (l_0,\nu_0) \xrightarrow{t_1,e_1} (l_1,\nu_1) \dots \xrightarrow{t_n,e_n} (l_n,\nu_n)$ be a finite run of SSG $\mathcal{SG}$. A strategy (for $\Diamond$) is a function that maps a finite run $\rho = (l_0,\nu_0) \xrightarrow{t_1,e_1} (l_1,\nu_1) \dots \xrightarrow{t_n,e_n} (l_n,\nu_n)$ to a pair (t,e) such that $(l_n,\nu_n) \xrightarrow{t,e} (l',\nu')$ for some (l',ν'), whenever $l_n \in \mathcal{Q}_{\Diamond}$. In order to measure probabilities of certain sets of runs, the following measurability condition is imposed on strategy $\lambda_{\Diamond}$: for every finite sequence of edges $e_1,\dots,e_n$ and every state s, the function $\chi : (t_1,\dots,t_n) \to (t,e)$ is such that $\chi(t_1,\dots,t_n) = (d,e)$ if and only if $\lambda_{\Diamond}(s_0 \overset{t_1,e_1}{\to} s_1 \overset{t_2,e_2}{\to} s_2 \dots \overset{t_n,e_n}{\to} s_n) = (t,e)$ is measurable.

Given SSG $\mathcal{SG}$, a finite run ρ ending in state s_0, and a strategy $\lambda_{\Diamond}$, we define $Runs(\mathcal{SG},\rho,\lambda_{\Diamond})$ to be the set of all runs generated by $\lambda_{\Diamond}$ after prefix ρ; that is, the set of all runs of the automaton satisfying the following condition: If $s_i = (l_i,\nu)$ and $l \in \mathcal{Q}_{\Diamond}$, then $\lambda_{\Diamond}$ returns (t_{i+1},e_{i+1}) when applied to $\rho \overset{t_1,e_1}{\to} s_1 \overset{t_2,e_2}{\to} \dots \overset{t_i,e_i}{\to} s_i$. Given a finite sequence $e_1,\dots,e_n$ of edges, a symbolic path $\pi_{\lambda_{\Diamond}}(\rho,e_1\dots e_n)$ is defined as $\pi_{\lambda_{\Diamond}}(\rho,e_1\dots e_n) = \{\rho' \in Runs(\mathcal{SG},\rho,\lambda_{\Diamond}) \mid \rho' = s_0 \overset{t_1,e_1}{\to} s_1 \overset{t_2,e_2}{\to} s_2 \dots \overset{t_n,e_n}{\to} s_n, \text{ with } t_i \in \mathbb{R}_{\geq 0}\}$. When $\lambda_{\Diamond}$ is clear, we simply write $\pi(\rho,e_1\dots e_n)$.

Given a strategy $\lambda_{\Diamond}$, and a finite run ρ ending in state $s=(l,\nu)$, a probability measure $\mathcal{P}_{\lambda_{\Diamond}}$ can be defined on the set $Runs(\mathcal{G},\rho,\lambda_{\Diamond})$, following [10]. First, define $\mathcal{P}_{\lambda_{\Diamond}}$ on symbolic paths starting with ρ, $\mathcal{P}_{\lambda_{\Diamond}}(\pi(\rho)) = 1$. Then:

If $\ell \in \mathcal{Q}_{\Diamond}$, and $\lambda_{\Diamond}(\rho) = (t,e)$,

$$\mathcal{P}_{\lambda_{\Diamond}}(\pi(\rho,e_1\dots e_n)) = \begin{cases} 0 & \text{if } e_1 \neq e \\ \mathcal{P}_{\lambda_{\Diamond}}(\pi(\rho \xrightarrow{t,e} s', e_2\dots e_n)) & \text{otherwise} \end{cases}$$

If $\ell \in \mathcal{Q}_{\bigcirc}$

$$\mathcal{P}_{\lambda_{\Diamond}}(\pi(\rho,e_1\dots e_n)) = \int_{t\in\mathcal{E}(s,e_1)} p(s+t)(e_1) \cdot \mathcal{P}_{\lambda_{\Diamond}}(\pi(\rho \xrightarrow{t,e_1} s', e_2\dots e_n))\ t\mu(s)(t)$$

where $s \xrightarrow{t,e_1} s'$ for every $d \in \mathcal{E}(s,e_1)$.

The correctness of these integrals is done as in [10], assuming some measurability conditions. When $\mathcal{Q}_{\Diamond} = \emptyset$ we call this game a $\frac{1}{2}$-player stochastic stopwatch games.

▶ **Example 4.** We give here an example to explain the method of computing probabilities in the ($\frac{1}{2}$-SSG) $\mathcal{SG}$ in the Figure 1. There is a single stopwatch x.

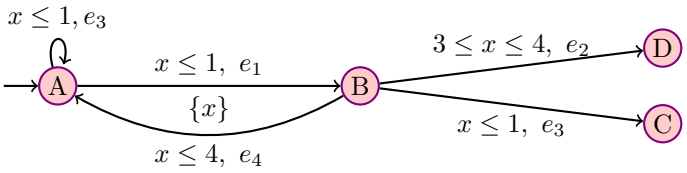


Figure 1 An example of $\frac{1}{2}$ SSG.

We consider two cases. First, assume the rate of x $\mathfrak{R}[l](x) = 0$ when, $l = B$, and $\mathfrak{R}[l](x) = 1$ when $l \neq B$. The initial state is $s_0 = (A, 0)$. Then:

$$\mathcal{P}(\pi((A,0), e_1e_3)) = \int_0^1 \frac{\mathcal{P}(\pi((B,0), e_3))}{2} d\mu_{(A,0)}(t) = \int_0^1 \frac{1}{2} \int_0^\infty \frac{1}{3} \cdot d\mu_{(B,0)}(t_1) \cdot d\mu_{(A,0)}(t)$$
$$= \int_0^1 \frac{1}{2}\frac{1}{3} \int_0^\infty e^{-t_1} dt_1 dt = \frac{1}{6}$$

$d\mu_{(A,0)}$ is the uniform distribution over [0,1]. Note that since the rate of x is 0 at B, an unbounded time can be spent, enabling transition e_3. Hence, $d\mu_{(B,0)}$ is the exponential distribution. Second, if we assume the rate of x is 1 in all locations (x is a clock), then we have the uniform distribution of all delays.

$$\mathcal{P}(\pi((A,0), e_1e_2)) = \int_0^1 \frac{\mathcal{P}(\pi((B,0), e_2))}{2} d\mu_{(A,0)}(t) = \int_0^1 \frac{1}{2} \int_3^4 \frac{1}{3} \cdot d\mu_{(B,0)}(t_1) \cdot d\mu_{(A,0)}(t)$$
$$= \int_0^1 \frac{1}{2}\frac{1}{3} \int_3^4 \frac{1}{4-0} dt_1 dt = \frac{1}{24}$$

Note that SSGs are defined on top of a stopwatch automata. In general, reachability is undecidable for stopwatch automata [13]. On restricting the number of stopwatches, we find the fine boundary between decidability and undecidability for $1\frac{1}{2}$-SSG.

Qualitative Reachability. We study the qualitative reachability problem for SSGs, stated as follows. Given a SSG $\mathcal{SG}$ with a set T of target locations, an initial state s_0 and $p \in \{0, 1\}$, decide whether there is a strategy $\lambda_\Diamond$ for Player $\Diamond$ such that $\mathcal{P}_{\lambda_\Diamond}(\{\rho \in Runs(\mathcal{SG}, s_0, \lambda_\Diamond) \mid \rho \text{ visits } T\}) \bowtie p$. Now we state our main theorem,

▶ **Theorem 5.** *The qualitative reachability problem of $1\frac{1}{2}$-SSG is*
(1) *decidable with two stopwatches moreover, it is EXPTIME complete.*
(2) *undecidable with three variables (with at least one stopwatch, and other two are clocks).*

3 Qualitative Reachability of $1\frac{1}{2}$ SSG : the two stopwatches case

The qualitative reachability of $1\frac{1}{2}$ SSG $\mathcal{SG}$ consists of two major objectives, reaching a desired set of locations $\mathcal{F}$ of $\mathcal{SG}$, with probability greater than 0, and equal to 1, represented by $\mathsf{Prob\_Reach}_{>0}(\mathcal{F})$ and $\mathsf{Prob\_Reach}_{=1}(\mathcal{F})$ respectively. All other objectives can be achieved using these two. All the edges of $\mathcal{SG}$ have some positive probability greater than 0 because we can remove the negligible edges effectively [10]. Since objectives under consideration are $\mathsf{Prob\_Reach}_{>0}(\mathcal{F})$ and $\mathsf{Prob\_Reach}_{=1}(\mathcal{F})$, exact probability does not matter. Thus, we only need to check if there is some clock valuation which allows us to make a move, or if all valuations are good. Hence, it is sufficient to work with regions.

▶ **Lemma 6.** *Given a 2 variable $1\frac{1}{2}$ SSG $\mathcal{SG}$, and desired set of target locations $\mathcal{F}$, we can compute the set of states from which player $\Diamond$ has a strategy to attain the objective of $\mathsf{Prob\_Reach}_{>0}(\mathcal{F})$.*

Proof. The first thing to do is to work with the underlying singular stopwatch automaton $\mathcal{A}$ of the $\mathcal{SG}$. Since $\mathcal{A}$ has only 2 stopwatches, with some care, the region construction applies to $\mathcal{A}$ (Appendix A.1), and we can construct the region automaton $\mathcal{R}(\mathcal{A})$ corresponding to $\mathcal{A}$, such that there is a run ρ in $\mathcal{A}$ reaching a target location $T \in \mathcal{F}$ if and only if there is a run ρ' in $\mathcal{R}(\mathcal{A})$ reaching a corresponding target location T'. $\mathcal{R}(\mathcal{A})$ is an untimed automaton,

and the locations of $\mathcal{R}(\mathcal{A})$ have the form (l, α) where l is a location of $\mathcal{A}$ and $\alpha \in \mathsf{Reg}(V)$ is a region over the variables V of $\mathcal{A}$. Thus, $T' = \{(T, \alpha) \mid \alpha \in \mathsf{Reg}(V), T \in \mathcal{F})\}$ is the set of target locations in $\mathcal{R}(\mathcal{A})$. Note that the region construction for singular stopwatch automata does not extend when there are 3 variables (see Appendix A.2).

Given a location $(l, \alpha) \in \mathcal{R}(\mathcal{A})$, we say that it belongs to $\mathcal{Q}_\Diamond$ if $l \in \mathcal{Q}_\Diamond$; likewise, $(l, \alpha) \in \mathcal{Q}_\bigcirc$ if $l \in \mathcal{Q}_\bigcirc$.

For brevity, in the following, we use ℓ to denote locations of $\mathcal{R}(\mathcal{A})$. Likewise we use $\mathcal{F}$ to denote target locations in $\mathcal{R}(\mathcal{A})$. Our reachability algorithm operates on the region automaton $\mathcal{R}(\mathcal{A})$. We do backward reachability, starting from the target $\mathcal{F}$. We construct a set Y as follows.

1. Initialize: Y_0=$\mathcal{F}$
2. Repeatedly add locations ℓ to Y_i, to construct Y_{i+1} as follows, $Y_{i+1} = Y_i \cup \{\ell\}$,
   a. If $\ell \in \mathcal{Q}_\bigcirc$, and has at least one enabled edge going into the set Y_i.
   b. If $\ell \in \mathcal{Q}_\Diamond$, and has at least one enabled edge going into the set Y_i.

We will repeat Step-2 until a fixpoint is reached.

Now, we claim that: a location $\ell \in Y$ if and only if there exists a strategy of player $\Diamond$ from ℓ to attain the objective $\mathsf{Prob\_Reach}_{>0}(\mathcal{F})$.

($\Rightarrow$) Let $\ell \in Y$. The rank of a location ℓ is i if it is added to Y_i. We prove that if $\ell \in Y$, then player $\Diamond$ has a strategy such that $\mathcal{F}$ is reached with positive probability by inducting on the rank of locations. The base case is trivial when $i = 0$ since target locations have rank 0. Assume the result holds for ranks $\leqslant i$. Now, we will prove for rank $i + 1$. There can be two different cases depending on the type of location.

Case $\ell \in \mathcal{Q}_\Diamond$: If location ℓ belongs to player $\Diamond$, then the probability of reaching $\mathcal{F}$ from ℓ is equal to the probability of reaching some location ℓ' of rank i (because of which ℓ was added to Y) which can be reached from ℓ according to the strategy of player $\Diamond$.

Case $\ell \in \mathcal{Q}_\bigcirc$: If location ℓ is probabilistic, then there exists an enabled out-going edge from ℓ to $\ell' \in Y_i$ whose probability is greater than 0 (say p_1). Since $\ell \in Y_i$, it reaches $\mathcal{F}$ with some probability (say p_2). Then the probability of reaching $\mathcal{F}$ from ℓ is $p_1.p_2$, which is greater than 0.

($\Leftarrow$) If $\ell \notin Y$, then the probability of reaching $\mathcal{F}$ from ℓ is equal to 0. To prove this we will consider the following,

Case $\ell \in \mathcal{Q}_\bigcirc \cup \mathcal{Q}_\Diamond$: If ℓ belongs to player $\Diamond$, then player $\Diamond$ has no strategy to reach set Y (because of the way Y is constructed). The case of probabilistic location is also the same. Hence, the probability of reaching Y from ℓ is equal to zero. ◀

▶ **Lemma 7.** *Given a 2 variable $1\frac{1}{2}$ SSG $\mathcal{SG}$, and a desired set of target locations $\mathcal{F}$, we can compute the set of states from which player $\Diamond$ has a strategy to attain objective $\mathsf{Prob\_Reach}_{=1}(\mathcal{F})$.*

Proof. First, we construct the set Y from which player $\Diamond$ has a strategy to reach $\mathcal{F}$ with probability greater than 0 (using algorithm given in Lemma 6). From set Y, we will construct a set Z as follows,

1. Initialize: Z_0=Y
2. Repeatedly remove locations ℓ from Z depending on their type, $Z_{i+1} = Z_i \setminus \ell$ until a fix-point is reached.
   a. If $\ell \in \mathcal{Q}_\bigcirc$, and has any enabled edge going out of the set Z_i.
   b. If $\ell \in \mathcal{Q}_\Diamond$, and has no enabled edge going into the set Z_i.

Now, we claim: a location $\ell \in Z$ if and only if player $\Diamond$ has a strategy to attain the objective $\mathsf{Prob\_Reach}_{=1}(\mathcal{F})$ from ℓ.

Let us consider the case that $\ell \notin Z$. We show that player $\Diamond$ does not have a strategy to reach a location in $\mathcal{F}$ with probability 1 from ℓ.

1. If $\ell \notin Y$. Then by Lemma 6, the probability of reaching $\mathcal{F}$ from ℓ is 0.
2. If $\ell \in Y \setminus Z$. Since $\ell \notin Z$, there exists some $i \geq 0$ such that $\ell \in Z_i \subseteq Y$, but $\ell \notin Z_{i+1} \subset Z_i$, and as $\ell \in Y$, atleast one outgoing edge of ℓ must be in Y.
   a. If $\ell \in \mathcal{Q}_{\bigcirc}$, then there exists an outgoing edge from ℓ to a node $\ell' \notin Z_i$.
   b. If $\ell \in \mathcal{Q}_{\Diamond}$, then all outgoing edges from ℓ are to nodes $\ell' \notin Z_i$.

Continuing backward from these nodes ℓ', we eventually reach nodes ℓ_{bad} such that $\ell_{bad} \notin Z_0 = Y$. $\ell_{bad} \notin Y$ implies that the probability of reaching $\mathcal{F}$ is 0. Since ℓ_{bad} is reachable from $\ell \in Y \setminus Z$, we conclude that the probability of reaching $\mathcal{F}$ from ℓ is not 1. The converse case, that is, if ℓ is a location such that player $\Diamond$ has no strategy to reach a location in $\mathcal{F}$ with probability 1, then $\ell \notin Z$ can be proved in a similar way. ◀

▶ **Theorem 8.** *The qualitative reachability problem is decidable for $1\frac{1}{2}$ SSG with two variables. Moreover, it is EXPTIME complete.*

Proof. We give the EXPTIME membership and hardness.

Membership. First, construct the region game graph from the given SSG (Appendix A.1). As seen in Lemma 6, a location (l, α) in the region graph is in $\mathcal{Q}_{\bigcirc}$ if and only if $l \in \mathcal{Q}_{\bigcirc}$; likewise, it is in $\mathcal{Q}_{\Diamond}$ if $l \in \mathcal{Q}_{\Diamond}$. Given the region game graph, we solve the qualitative reachability question using a backward fixpoint algorithm that iteratively refines the probability computation starting from the target locations. We know that the fixed point computation is polynomial time with respect to the size of the underlying graph (as this can be solved using BFS on the underlying graph). The size of the region graph (Appendix A.1) is exponential in the number of the variables V, when $|V| > 1$. Given the polytime algorithm for the fixed point, this problem is in EXPTIME.

Hardness. For the hardness we use the qualitative reachability problem of probabilistic timed automata (PTA) with two clocks [15]. A probabilistic timed automata (PTA) [17] is defined as $\mathsf{T} = (\mathcal{Q}, \ell_0, \mathcal{X}, \Delta, \Delta_{prob})$ where, $\mathcal{Q}$ is a finite set of locations, $\ell_0 \in \mathcal{Q}$ is the initial location, $\mathcal{X}$ is the finite set of real valued variables called clocks, Δ is a set of transitions of the form $(\ell, \varphi, R, \ell')$ with the usual semantics as timed automata transitions, and Δ_{prob} is the set of probabilistic transitions of the form $\big(\ell, \varphi, \mathsf{Dist}(2^{\mathcal{X}} \times \mathcal{Q})\big)$ where $\ell \in \mathcal{Q}$, φ is a clock constraint in the outgoing transitions from ℓ and $\mathsf{Dist}(2^{\mathcal{X}} \times \mathcal{Q})$ is a probability distribution which assigns probabilities to (reset set, target location) pairs (R, ℓ') on outgoing transitions $(\ell, \varphi, R, \ell')$ from ℓ. W.l.o.g, in the PTA, we replace probabilities with weights (and calculate probabilities from weights in the usual way). See Figure 2 for a PTA (on the left).

A $1\frac{1}{2}$ SSG with two variables can simulate a PTA $\mathsf{T} = (\mathcal{Q}, \ell_0, \{x, y\}, \Delta, \Delta_{prob})$ with two clocks x, y. We construct a $1\frac{1}{2}$ SSG $\mathcal{SG}$ with two variables x', y' corresponding to the two clocks x, y of the T. For each location $\ell \in \mathcal{Q}$ of the T we create location $\ell_{\Diamond} \in \mathcal{Q}_{\Diamond}$. The two clocks of the T are simulated using the two variables of the $1\frac{1}{2}$ SSG whose rate is 1 $\mathfrak{R}(\ell_{\Diamond})[x'] = \mathfrak{R}(\ell_{\Diamond})[y'] = 1$ in all locations from $\mathcal{Q}_{\Diamond}$. The transitions Δ of the T are added in the $1\frac{1}{2}$-SHG (by replacing ℓ with $\ell_{\Diamond}$, and replacing x, y with x', y'). It remains to add the probabilistic transitions Δ_{prob} to $\mathcal{SG}$. Consider $t = \big(\ell^i, \varphi, \mathsf{Dist}_i\big) \in \Delta_{prob}$. We add a new stochastic location $\ell_t \in \mathcal{Q}_{\bigcirc}$ such that, the rate of x', y' are zero at ℓ_t and add the following transitions in the $\mathcal{SG}$.

- $(\ell^i_{\Diamond}, \varphi, \emptyset, \ell_t)$ i.e., a transition from $\ell^i_{\Diamond}$ to the stochastic location ℓ_t with the same constraints of the transition t.
- For each pair $(R_j, \ell^j) \in \mathsf{Dist}(2^{\mathcal{X}} \times \mathcal{Q})$ in the probability distribution $(\ell^i, \varphi, \mathsf{Dist}(2^{\mathcal{X}} \times \mathcal{Q}))$ with weight w_j, we add the transition $(\ell_t, \top, R_j, \ell^j_{\Diamond})$ with the same weight w_j.

The probability incurred to go from ℓ_t to $\ell^j_\Diamond$ is given by $\frac{w_j}{\sum w_j}\int_0^\infty e^{-t}dt$, since an unbounded delay is allowed at ℓ_t. Hence, the probability in the SSG $\mathcal{SG}$ to go from $\ell^i_\Diamond$ to $\ell^j_\Diamond$ is $\frac{w_j}{\sum w_j}$ which is the same as the probability given by the PTA T. Since we preserve all probabilities, it is easy to check that, by solving the qualitative reachability of the constructed $1\frac{1}{2}$ SSG we solve the qualitative reachability of the T with two clocks. Hence the qualitative reachability of $1\frac{1}{2}$ SSG is EXPTIME-Hard. ◀

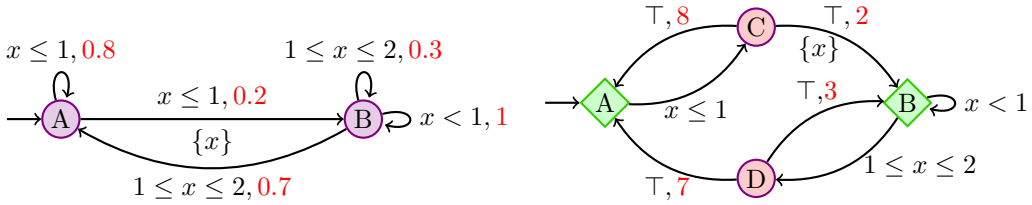


Figure 2 PTA to $1\frac{1}{2}$ SSG reduction (PTA is on left and $1\frac{1}{2}$-SSG on right). Probabilities (and weights for SSG) are in red color. From A, on $x \le 1$, there is a reset free edge and a reset edge with probabilities 0.8 and 0.2. Likewise, from B, on $1 \le x \le 2$, there is a distribution (0.3, 0.7), while for $x \le 1$, there is just one transition. The green diamond shaped nodes in the right are of Player $\Diamond$ and other nodes are stochastic.

We next show that the qualitative reachability problem becomes undecidable as soon as we have three variables. The proof goes via a reduction from the non-halting problem for Minsky two counter machines to the qualitative reachability problem for $1\frac{1}{2}$ SSG with three variables, where we have one stopwatch and two clocks.

Two-counter machine. A counter machine can be defined as a tuple $(L, \mathcal{C})$, where L is the finite state of instructions including the special instruction "HALT", and $\mathcal{C} = \{C_1, C_2\}$. The instruction can be any one of the following types,

1) (Increment the counter) $\ell_p : C_i := C_i + 1;$ goto ℓ_q; $\forall i \in \{1, 2\}$,
2) (Decrement the counter) $\ell_p : C_i := C_i - 1;$ goto ℓ_q; $\forall i \in \{1, 2\}$,
3) (Checking zero) $\ell_p :$ if $(C_i = 0)$ then goto ℓ_q else goto ℓ_r; $\forall i \in \{1, 2\}$,
4) (Halting instruction) $\ell_q :$ HALT;

Where, $C_i \in \mathcal{C}$, $\ell_p, \ell_q, \ell_r \in L$. A configuration of a two-counter machine is a tuple (ℓ, m, n) where, $\ell \in L$ and $m, n \in \mathbb{N} \cup \{0\}$ represents the current value of the counters c_1, c_2 respectively. A two-counter machine starts from the initial configuration $(\ell_0, 0, 0)$. A run of a two-counter machine is a sequence of configurations $(\ell_0, 0, 0) \to (\ell_1, m_1, n_1) \to (\ell_2, m_2, n_2) \cdots$. The transition between two configurations depends on the instruction of the first configuration. We say a run is halting if it is finite and ends with an HALT instruction, in fact the two-counter machine never progresses beyond a HALT instruction. The halting problem of a two-counter machine is checking if a given two-counter machine has a halting run or not. It is well-known that two-counter machine is Turing complete and the halting problem for two-counter machine is undecidable [19].

▶ **Theorem 9.** *The qualitative reachability problem for $1\frac{1}{2}$ SSG is $\prod_1^0$ hard with one stopwatch variable and two clocks.*

We prove the $\prod_1^0$ hardness of qualitative reachability for $1\frac{1}{2}$ SSG with one stopwatch and two clocks by reducing it to the non-halting problem for two counter machines.

Reduction to reachability in $1\frac{1}{2}$ SSG

Given a two counter machine $\mathcal{M}$, we construct a one and half player SSG $G(\mathcal{M})$, where $\Diamond$ has a strategy to reach some desired location (denoted as pink square nodes labeled T) with probability =1 if and only if $\mathcal{M}$ does not halt.

The game graph $G(\mathcal{M})$ uses 3 variables : a stopwatch x and two clocks y, z. The values i, j of the counters C_1, C_2 are encoded in the variable x as $\frac{1}{2^i 3^j}$. y, z are used as auxiliary clock variables. The stopwatch x has rate $r_x = 1$ in all the stochastic nodes. The $\Diamond$ nodes where x has rate $r_x = 0$ are colored blue. The graph $G(\mathcal{M})$ has one gadget per instruction of the two counter machine. By adjoining the gadgets appropriately, depending on the instructions of $\mathcal{M}$, we obtain the complete game graph $G(\mathcal{M})$.

Decrement C_1

Let us begin with the gadget for the instruction ℓ_i: $C_1 := C_1 - 1$; goto ℓ_j. Figure 3 depicts the gadget for decrementing counter C_1.

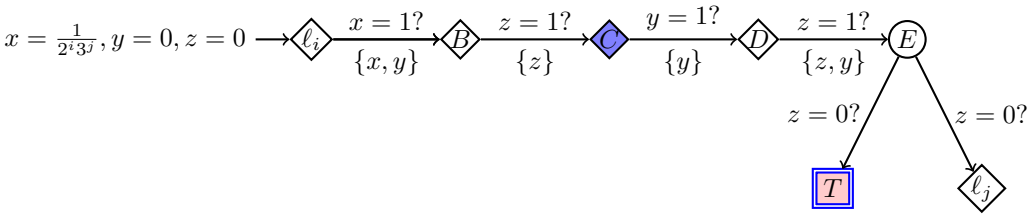


Figure 3 Gadget Dec C_1.

▶ **Lemma 10.** *On entering the gadget Dec C_1 in Figure 3, in node ℓ_i with values $x = \frac{1}{2^i 3^j}$, $y = z = 0$, the node ℓ_j is reached with probability $\frac{1}{2}$ with $x = \frac{1}{2^{i-1} 3^j}, y = z = 0$ and the target node T is reached with probability $\frac{1}{2}$.*

Proof. The proof of correctness of Lemma 10 can be seen by examining the functioning of the gadget: On entry to node ℓ_i, we have the values $x = \frac{1}{2^i 3^j}$, $y = z = 0$. A time $1 - \frac{1}{2^i 3^j}$ is spent at location ℓ_i, obtaining $x = y = 0$,$z = 1 - \frac{1}{2^i 3^j}$ on entry into node B.

Time equal to $\frac{1}{2^i 3^j}$ is spent at node B, obtaining valuations $x = y = \frac{1}{2^i 3^j}$,$z = 0$ on entry to node C. Subsequently time equal to $1 - \frac{1}{2^i 3^j}$ is spent at node C, obtaining valuations $x = \frac{1}{2^i 3^j}$,$y = 0$, $z = 1 - \frac{1}{2^i 3^j}$ on entry to node D. The value of x remains constant during this transition to node D as the node is shaded blue. Then the constraint forces to spend time equal to $\frac{1}{2^i 3^j}$ at node D and reaches the stochastic node E with valuation $x = \frac{2}{2^i 3^j} = \frac{1}{2^{i-1} 3^j}$,$y = z = 0$. Through the stochastic node E, ℓ_j and T can be reached with probability $= \frac{1}{2}$ each, with the clock valuation $x = \frac{1}{2^{i-1} 3^j}$,$y = z = 0$ as required. ◀

Increment C_1

Next, let us look at the instruction for incrementing. Figure 4 depicts the gadget for incrementing counter C_1 simulating the instruction $\ell_i : C_1 := C_1 + 1$, goto ℓ_j.

▶ **Lemma 11.** *On entering the gadget Inc C_1 in Figure 4, in node ℓ_i with values $x = \frac{1}{2^i 3^j}$, $y = z = 0$, the node ℓ_j is reached with probability $\frac{1}{2}$ and $x = \frac{1}{2^{i+1} 3^j}, y = z = 0$. The target node is reached with probability $\frac{1}{2}$.*

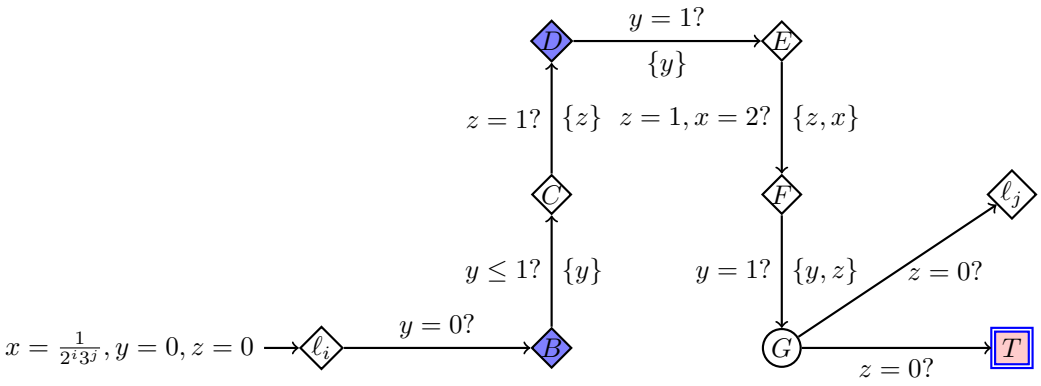


Figure 4 Gadget Inc C_1.

Proof. The proof of Lemma 11 can be seen by examining the functioning of the gadget in Figure 4. The node ℓ_i is entered with $x = \frac{1}{2^i 3^j}, y = z = 0$. The node B is entered with the same values but the stopwatch x is now stopped because the node is blue shaded. An amount of time $t \leq 1$ is spent at B obtaining $x = \frac{1}{2^i 3^j}, y = 0, z = t$ on entering node C. We spend $1 - t$ time at node C entering node D with valuations $x = 1 - t + \frac{1}{2^i 3^j}, y = 1 - t, z = 0$. Now t time is spent at node D with x paused (the node D is blue shaded) and hence we enter node E with valuations $x = 1 - t + \frac{1}{2^i 3^j}, y = 0, z = t$. Now to satisfy the first guard i.e., $z = 1$ for leaving E we need to spend $1 - t$ time at E which would mean x would now be $x = 2 - 2t + \frac{1}{2^i 3^j}$. To move to F we also need $x = 2$ which would imply that $2t = \frac{1}{2^i 3^j}$. Then we enter node F with $x = z = 0, y = 1 - t$ where $t = \frac{1}{2^{i+1} 3^j}$. We spend t time at F and through the stochastic node G. We enter node ℓ_j and target node T with probability $\frac{1}{2}$ each, with clock values $x = \frac{1}{2^{i+1} 3^j}, y = 0, z = 0$ as required. ◀

Zero Check for C_1

Next, we look at the zero check instruction, ℓ_i : if $C_1 = 0$, then goto ℓ_j, else goto ℓ_k. Figure 5 depicts the gadget for zero check of counter C_1.

▶ **Lemma 12.** *On entering the gadget Zero Check C_1 in Figure 5, in node ℓ_i with values $x = \frac{1}{2^i 3^j}$, $y = z = 0$, player $\Diamond$ has a strategy to reach the nodes l_j and the target T with probability $\frac{1}{2}$ each if and only if $i = 0$, that is, the value of $x = \frac{1}{3^j}$ on entry at ℓ_i. Similarly, player $\Diamond$ has a strategy to reach the nodes l_k and the target node T with probability $\frac{1}{2}$ each if and only if $i \neq 0$, that is, the value of $x = \frac{1}{2^i 3^j}$, $i > 0$ on entry at ℓ_i.*

Proof. The proof of Lemma 12 follows by examining the functioning of the gadget in Figure 5. The $\Diamond$ player has two possible strategies at node ℓ_i: to goto Z or NZ. The correct strategy is to go to Z when $x = \frac{1}{3^j}$ and to go to NZ when $x = \frac{1}{2^i 3^j}$, $i > 0$ at node ℓ_i. No time is elapsed at node ℓ_i. Assume the strategy is to go to Z, no time is elapsed at Z, and the stochastic node SZ is entered. Then from the stochastic node SZ, with probability $\frac{1}{2}$ each, the node ℓ_j (corresponding to the next instruction) and the gadget $Make\ C_2\ 0$ is entered. This gadget is in Figure 6. From the $\Diamond$ node SZ, we enter the starting location A of in Figure 6, and from the same node A, we can move to the location T of Figure 5 if $x = 1, z = 0$. If the strategy of choosing Z was indeed the correct one, then, decrementing C_2 some number of times would give $x = 1$. In this case, after some number of iterations of the gadget in Figure 6, we reach T in Figure 5 with probability $\frac{1}{2}$. Note that the gadget in Figure 6 has no stochastic nodes, so on successful completion we reach T with $x = 1$ incurring probability $\frac{1}{2}$.

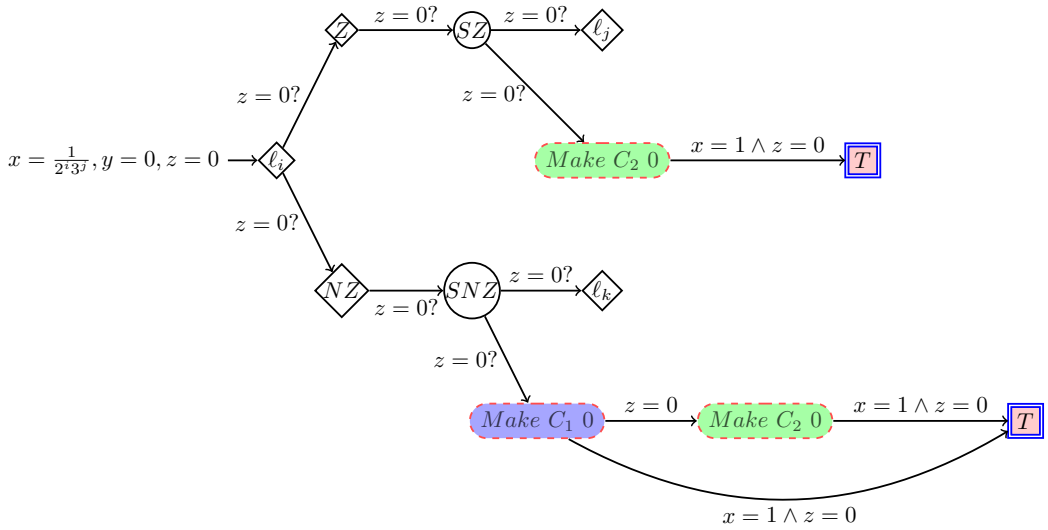


Figure 5 Gadget Zero Check C_1.

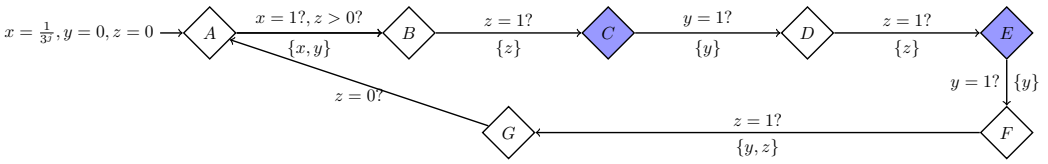


Figure 6 Make C_2 0.

Note that if the decision of choosing node Z is incorrect, that is, if $x = \frac{1}{2^i 3^j}$, $i > 0$, then the value of x will exceed 1 and we will be stuck in the gadget $Make\ C_2\ 0$.

The decision of choosing NZ from ℓ_i is similar : in this case, the gadget $Make\ C_1\ 0$ has to be visited at least once, before the target T is reached. $Make\ C_1\ 0$ can be obtained similar to $Make\ C_2\ 0$, and it also has no stochastic nodes. In particular, if we have $x = \frac{1}{2^i 3^j}$, $i > 0$, then we must iterate $Make\ C_1\ 0$ i times exactly, and in case $x < 1$ we iterate $Make\ C_2\ 0$ j times. If we iterate $Make\ C_1\ 0$ $< i$ $(> i)$ times, we will get stuck in $Make\ C_1\ 0$ $(Make\ C_2\ 0)$. Otherwise, we will reach T in Figure 5 with probability $\frac{1}{2}$. ◀

Gadgets for incrementing, decrementing and zero check gadget for C_2 are similar to the seen gadgets. We now argue that $\Diamond$ has a strategy to reach a target location T with probability 1 if and only if the two counter machine does not halt.

▶ **Theorem 13.** *Given a two counter machine $\mathcal{M}$, Player $\Diamond$ has a strategy to reach a target node T with probability 1 in the constructed game graph $G(\mathcal{M})$ if and only if the two counter machine $\mathcal{M}$ does not halt.*

Proof. The proof of Theorem 13 is obtained by putting together the lemmas above. Since HALT is also an instruction in the two counter machine, we have a gadget with location labeled *Halt* corresponding to the HALT instruction. From this node, there is no outgoing edge and hence this is a dead state. The rest of the SSG is made by appropriately stringing together the gadgets as per the design of the two counter machine. We claim that player $\Diamond$ has a strategy to reach a target location T with probability 1 if and only if he simulates all the instructions correctly and if and only if $\mathcal{M}$ does not halt.

Consider the first instruction of the two counter machine that is executed. By using all lemmas so far, whatever this instruction might be, on a correct simulation, the SSG enters a target T in some gadget with probability $\frac{1}{2}$ and continues execution from the state as specified by the instruction with probability $\frac{1}{2}$.

Therefore the probability of reaching a target T is $P_{total} = \frac{1}{2}$ (reaching T in the first gadget) $+ \frac{1}{2}P_{rest}$ where P_{rest} is the probability of reaching the target T node when continuing the simulation of $\mathcal{M}$ after the current instruction. Recall that each gadget corresponding to instructions went with probability $\frac{1}{2}$ to the next instruction ℓ_j to be simulated; the $\frac{1}{2}$ in the term $\frac{1}{2}P_{rest}$ comes from there.

We now apply the above process for finding P_{total} to find P_{rest} recursively for the next executed instruction from ℓ_j and we get $P_{total} = \frac{1}{2} + \frac{1}{2}(\frac{1}{2} + \frac{1}{2}P_{rest})$ where P_{rest} is the probability of reaching T when continuing execution from the subsequent instruction reached.

This above processes can be recursively repeated. If $\mathcal{M}$ reaches HALT, then we will reach a gadget where with probability $\frac{1}{2}$ we reach T and with probability $\frac{1}{2}$ we reach HALT. In this case, the above summation will add up to $< 1 : P_{total} = \sum_{n=1}^{I} 2^{-n} < 1$ where I is the total number of instructions executed to reach HALT. However if $\mathcal{M}$ does not halt, then the run is not finite, and we keep going one gadget after the other. Then we get the infinite sum $P_{total} = \sum_{n=1}^{\infty} 2^{-n} = 1$. Hence player $\Diamond$ will reach the target node T with probability 1 if and only if the two counter machine does not halt. ◀

4 Conclusion

In this paper, we have proposed stochastic stopwatch games, an extension of stochastic timed games and proved decidability and undecidability results for $1\frac{1}{2}$-stochastic stopwatch games. This work leads us to further open problems for e.g., how does the undecidability result change if we consider time-bounded qualitative reachability or when we consider only $\frac{1}{2}$-stochastic stopwatch games.

References

1. S. Akshay, Patricia Bouyer, Shankara Narayanan Krishna, Lakshmi Manasa, and Ashutosh Trivedi. Stochastic Timed Games Revisited. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2016.8.
2. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, February 1995. doi:10.1016/0304-3975(94)00202-t.
3. Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229. Springer Berlin Heidelberg, 1993. doi:10.1007/3-540-57318-6_30.
4. Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.
5. Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. *IFAC Proceedings Volumes*, 31(18):447–452, 1998. doi:10.1016/S1474-6670(17)42032-5.
6. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, June 2003. doi:10.1109/tse.2003.1205180.

7 Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Größer. Probabilistic and topological semantics for timed automata. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, pages 179–191. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-77050-3_15.

8 Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Grosser. Almost-sure model checking of infinite paths in one-clock timed automata. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*. IEEE, June 2008. doi:10.1109/lics.2008.25.

9 Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Groesser, and Marcin Jurdzinski. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4), December 2014. doi:10.2168/lmcs-10(4:6)2014.

10 Patricia Bouyer and Vojtech Forejt. Reachability in stochastic timed games. In *ICALP*, pages 103–114. Springer, 2009. doi:10.1007/978-3-642-02930-1_9.

11 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On model-checking timed automata with stopwatch observers. *Information and Computation*, 204(3):408–433, 2006. doi:10.1016/j.ic.2005.12.001.

12 Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joel Ouaknine, Jean-Francois Raskin, and James Worrell. Time-bounded reachability for monotonic hybrid automata: Complexity and fixed points. In *Automated Technology for Verification and Analysis*, pages 55–70. Springer International Publishing, 2013. doi:10.1007/978-3-319-02444-8_6.

13 Franck Cassez and Kim Larsen. The impressive power of stopwatches. In *CONCUR 2000 – Concurrency Theory*, pages 138–152. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-44618-4_12.

14 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, August 1998. doi:10.1006/jcss.1998.1581.

15 Marcin Jurdzinski, Jeremy Sproston, and Francois Laroussinie. Model checking probabilistic timed automata with one or two clocks. *Logical Methods in Computer Science*, 4(3), September 2008. doi:10.2168/lmcs-4(3:12)2008.

16 Shankara Narayanan Krishna, Umang Mathur, and Ashutosh Trivedi. Weak singular hybrid automata. In *Lecture Notes in Computer Science*, pages 161–175. Springer International Publishing, 2014. doi:10.1007/978-3-319-10512-3_12.

17 Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *CONCUR 2000 – Concurrency Theory*, pages 123–137. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-44618-4_11.

18 Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, June 2002. doi:10.1016/s0304-3975(01)00046-9.

19 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., USA, 1967.

20 Nima Roohi and Mahesh Viswanathan. Time-bounded reachability for initialized hybrid automata with linear differential inclusions and rectangular constraints. In *Lecture Notes in Computer Science*, pages 191–205. Springer International Publishing, 2014. doi:10.1007/978-3-319-10512-3_14.

A Singular Stopwatch Automata with Two Stopwatches

Let 2-$\mathcal{H}$ be a singular stopwatch automata with two stopwatch variables $V = \{x, y\}$. When the current location $l \in \mathcal{Q}$ is known, we represent the rate of $c \in V$ as $r_c = \mathfrak{R}(l)[c]$ i.e., $r_x, r_y \in \{0, 1\}$ for $x, y \in V$ respectively. Let $c_{\max}$ be the maximum constant used in any of the guards of 2-$\mathcal{H}$. Note that, unlike clock regions defined in [4], we can not directly define

regions for stopwatches, as the successor region depends on the current rate of change r_x, r_y of stopwatch x and y respectively. So, we need to define regions such that they agree with the successor function with different rates.

A.1 Regions and Region Automaton of 2-$\mathcal{H}$

In this section, we define a region abstraction, which is reachability preserving in the sense of the region automaton from [4]. Intuitively, a region $\Re$ is a collection of infinitely many valuations $\nu \in \mathbb{R}^2_{\geq 0}$, having some good properties preserving reachability. The number of regions must be finite.

▶ **Definition 14.** *Let $\varphi \in \mathit{rect}(V)$ be a constraint. A region $\Re$ is compatible with φ if and only if for all valuations $\nu \in \Re$, either $\nu \models \varphi$ or $\nu \models \neg\varphi$.*

We define a map $\mathsf{Res} : \Re \to \Re$ that maps a region $\Re$ to the region $\mathsf{Res}(\Re)$ obtained from $\Re$ by assigning value 0 to all variables which were reset to 0.

▶ **Definition 15.** *A set of regions $\Re$ is compatible with resets (or with the map Res) if whenever a valuation $\nu' \in \Re'$ is reachable from a valuation $\nu \in \Re$ after a reset, then $\Re'$ is reachable from any $\nu \in \Re$ by the same reset.*

Construction of Regions Reg(V)

We first construct a set of regions for 2-$\mathcal{H}$ that are compatible with resets and guards. For $z \in \{x, y\}$, we define the set of intervals

$$\mathcal{I}_z = \big\{[c] \mid 0 \leq c \leq c_{\max}\big\} \cup \big\{(c, c+1) \mid 0 \leq c < c_{\max}\big\} \cup \big\{(c_{\max}, \infty)\big\}$$

For a variable x and its current valuation $\nu[x]$, we use $I_x \in \mathcal{I}_x$ to represent its current interval, and $\mathsf{fract}(\nu[x])$ to denote the fractional part of $\nu[x]$. For example, if we have a variable x with valuation $\nu[x] = 4.5$ then it's interval $I_x = (4, 5) \in \mathcal{I}_x$ and $\mathsf{fract}(\nu[x]) = 0.5$. Let us define a relation $\sim_\Re$ between two valuation ν_1 and ν_2 such that $\nu_1 \sim_\Re \nu_2$ if and only if,

- $\forall z \in V$, $I_z \in \mathcal{I}_z$ $\nu_1[z] \in I_z \iff \nu_2[z] \in I_z$
- $x, y \in V$, $\mathsf{fract}(\nu_1[x]) \leq \mathsf{fract}(\nu_1[y]) \iff \mathsf{fract}(\nu_2[x]) \leq \mathsf{fract}(\nu_2[y])$

Clearly, the relation $\sim_\Re$ is an equivalence relation and this forms a finite partitioning of $\mathbb{R}^2_{\geq 0}$. Let us define such a partition by $\alpha = (I_x, I_y, \prec, \mathsf{Rel})$ where, I_x, I_y represents the interval of clocks x and y respectively, and $\prec$ is a total pre-order on the set $\mathsf{Rel} = \big\{x \in V \mid I_x \in \big\{(c, c+1) \mid 0 \leq c < c_{\max}\big\}\big\}$. Assume $\Re_\alpha$ represents the region defined by α.

We define $\mathsf{Reg}(V)$ to be the set of all such partitions $\Re_\alpha$. For each valuation $\nu \in \mathbb{R}^2_{\geq 0}$ of variables, the unique element $\Re$ of $\mathsf{Reg}(V)$ that contains ν is called a region, denoted $[\nu]$.

Successors of a Region

We define the successors of a region $\Re$, $\mathsf{Succ}_{(r_x, r_y)}(\Re) \subseteq \mathsf{Reg}(V)$, in the following natural way: $r_x, r_y \in \{0, 1\}, x, y \in V$,

$$\Re' \in \mathsf{Succ}_{(r_x, r_y)}(\Re) \text{ if } \exists \nu \in \Re, \exists t \in \Re \text{ such that } [\nu + (r_x, r_y)t] = \Re' \tag{1}$$

By $\nu + (r_x, r_y)t$, we represent the valuation $\big(\nu[x] + r_x \cdot t, \nu[y] + r_y \cdot t\big)$.

A finite partition $\mathsf{Reg}(V)$ of $\mathbb{R}^2_{\geq 0}$ is a *set of regions* whenever the following condition holds:

$$\Re' \in \mathsf{Succ}_{(r_x, r_y)}(\Re) \text{ if and only if } \forall \nu \in \Re, \exists t \in \Re \text{ such that } [\nu + (r_x, r_y)t] = \Re' \tag{2}$$

We now consider resets. Formally, we have

$$\Re' \in \mathsf{Res}(\Re) \rightarrow \forall \nu \in \Re, \exists \nu' \in \Re' \text{ such that } \nu' \in \mathsf{Res}(\nu) \tag{3}$$

Now, we will show that $\mathsf{Reg}(V)$ follows the conditions (2) and (3) defined above.

▶ **Lemma 16.** *For all partitions $\Re \in \mathsf{Reg}(V)$, $\Re' \in \mathit{Succ}_{(r_x,r_y)}(\Re)$ if and only if for all valuation $\nu \in \Re$, there exists $t \in \mathbb{R}_{\geq 0}$ such that $[\nu + (r_x, r_y)t] = \Re'$*

Proof. Consider a partition $\Re_\alpha$ defined by $\alpha = (I_x, I_y, \prec, \mathsf{Rel})$.

If $I_x = I_y = (c_{max}, \infty)$, then $\mathsf{Succ}_{(r_x,r_y)}(\Re_\alpha) = \{\Re_\alpha\}$ because, for all $\nu \in \Re_\alpha$, for all $t \in \mathbb{R}_{\geq 0}$, for $r_x, r_y \in \{0, 1\}$, $\nu + (r_x, r_y)t \in \Re_\alpha$.

If $r_x = r_y = 0$ then, $\mathsf{Succ}_{(r_x,r_y)}(\Re_\alpha) = \{\Re_\alpha\}$, for all $\Re_\alpha$.

If $\mathsf{Succ}_{(r_x,r_y)}(\Re_\alpha) \neq \{\Re_\alpha\}$, then there exists atleast one another region in $\mathsf{Succ}_{(r_x,r_y)}(\Re_\alpha)$ that is different from $\Re_\alpha$. Let $\Re_\beta$ denote the region that is closest to region to $\Re_\alpha$ such that, $\Re_\beta \in \mathsf{Succ}_{(r_x,r_y)}(\Re_\alpha)$, and for all $\nu \in \Re_\alpha$, for all $t \in \mathbb{R}_{\geq 0}$, if $\nu + (r_x, r_y)t \notin \Re_\alpha$, then $\exists t' \leq t$ such that $\nu + (r_x, r_y)t' \in \Re_\beta$. Assume that, such a region $\Re_\beta$ is defined by $\beta = (I'_x, I'_y, \prec', \mathsf{Rel}')$ and characterized as follows:

Let $Z = \{z \in V \mid I_z \text{ is of the form } [c]\}$, i.e., Z is the set of clocks with integer value.

1. If $Z \neq \emptyset$ and $r_x = r_y = 1$, (if $x \notin Z$ then $y \in Z$ and vice versa)

$$I'_z = \begin{cases} I_z & \text{if } z \notin Z, \\ (c, c+1) & \text{if } z \in Z, I_z = [c], \text{ and } 0 \leq c < c_{\max} \\ (c_{\max}, \infty) & \text{if } z \in Z \text{ and } I_z = [c_{\max}] \end{cases}$$

   and, $x \prec' y$ if $I_x = [c], I'_x = (c, c+1)$ with $0 \leq c < c_{\max}$ and I'_y is of the form $(d, d+1)$, $\mathsf{Rel}' = \{x, y\}$.
2. If $Z \neq \emptyset$ and atleast one of r_x, r_y is 0, (if $x \notin Z$ then $y \in Z$ and vice versa, and if $r_x = 1$ then $r_y = 0$ and vice versa.)

$$I'_z = \begin{cases} I_z & \text{if } r_z = 0, \\ [c+1] & \text{if } z \notin Z, I_z = (c, c+1), r_z = 1, \text{ and } 0 \leq c < c_{\max} \\ (c, c+1) & \text{if } z \in Z, I_z = [c], r_z = 1 \text{ and } 0 \leq c < c_{\max} \\ (c_{\max}, \infty) & \text{if } z \in Z, I_z = [c_{\max}], \text{ and } r_z = 1 \end{cases}$$

   and, $\mathsf{Rel}' = \{x, y\}, x \prec' y$ if $r_x = 1$ and $x \in Z, y \notin Z$. If ($r_x = 0$ and $x \in Z$) or ($r_x = 1$ and $x \notin Z$) or if $x, y \in Z$, then $\mathsf{Rel}' = \emptyset$.

3. If $Z = \emptyset$ and $r_x = r_y = 1$. Let M denote the set of variables with the maximum fractional part, whose interval is of the form $(c, c+1)$ for $0 \leq c < c_{\max}$. Then,

$$I'_z = \begin{cases} I_z & \text{if } z \notin M, \\ [c+1] & \text{if } z \in M \text{ and } I_z = (c, c+1) \text{ with } 0 \leq c < c_{\max} \end{cases}$$

   One variable moves to an integer value, or both variables are in $(c_{\max}, \infty)$ thus, $\mathsf{Rel}' = \emptyset$.

4. If $Z = \emptyset$ and atleast one of r_x, r_y is 0. Then,

$$I'_z = \begin{cases} I_z & \text{if } r_z = 0, \\ [c+1] & \text{if } z \in M, r_z = 1, \text{ and } I_z = (c, c+1) \text{ with } 0 \le c < c_{\max} \\ I_z & \text{if } z \notin M, r_z = 1, \text{ and } I_z = (c, c+1) \text{ with } 0 \le c < c_{\max} \end{cases}$$

and, $x \prec' y$ is same as $x \prec y$ when $r_x = r_y = 0$. Otherwise, one of the variables gets an integer value, and hence $\mathsf{Rel}' = \emptyset$.

We now claim that,

$\forall \nu \in \Re_\alpha, \exists t \in \mathbb{R}_{\ge 0}$ such that $\nu + t \in \Re_\beta$

Let ν be a valuation in $\Re_\alpha$.

1. If $Z \neq \emptyset$ and $r_x = r_y = 1$. Let $\tau = \min\{1 - \mathsf{fract}(\nu[z]) \mid I_z \text{ is of the form } (c, c+1)\}$. Then $\nu + (1,1)\frac{\tau}{2}$ is in the region R_β.
2. If $Z \neq \emptyset$ and atleast one of r_x, r_y is 0.
   - **(i)** If $x \in Z, y \notin Z$ and $r_x = 1$, then pick $\tau = \mathsf{fract}(\nu[y])$. Then $\nu + (1,0)\frac{\tau}{2}$ is in the region $\Re_\beta$.
   - **(ii)** If $r_x = 0$ and $x \in Z$, then pick $\tau = 1 - \mathsf{fract}(\nu[y])$. Then $\nu + (0,1)\tau$ is in the region $\Re_\beta$.
   - **(iii)** If $r_x = 1$ and $x \notin Z$, then pick $\tau = 1 - \mathsf{fract}(\nu[x])$. $\nu + (1,0)\tau$ is in the region $\Re_\beta$.
   - **(iv)** If $x, y \in Z$, and $r_x = 1$, then pick $\tau = 0.5$. Then $\nu + (1,0)\tau$ is in the region $\Re_\beta$.
3. If $Z = \emptyset$ and $r_x = r_y = 1$.
   Pick the variable $z \in M$. Let $\tau = 1 - \mathsf{fract}(\nu[z])$. Then $\nu + (1,1)\tau$ is in the region $\Re_\beta$.
4. If $Z = \emptyset$ and atleast one of r_x, r_y is 0.
   If $r_x = 1$ and $r_y = 0$. Pick $\tau = 1 - \mathsf{fract}(\nu[x])$. Then $\nu + (1,0)\tau$ is in the region $\Re_\beta$.

Thus we obtain that $\Re_\beta \in \mathsf{Succ}_{(r_x, r_y)}(\Re_\alpha)$ is the closest successor of $\Re_\alpha$. Inducting on $\Re_\beta$, we get the closest successor of $\Re_\beta$, which is also a successor of $\Re_\alpha$, 2 steps away, and so on. We write $\Re_\alpha \xrightarrow{n} \Re^n_\alpha$ if $\Re^n_\alpha$ is the nth closest successor of $\Re_\alpha$ with respect to some choice of rates (r_x, r_y). This clearly means that there is a sequence of regions $\Re^0_\alpha, \Re^1_\alpha, \Re^2_\alpha, \ldots, \Re^n_\alpha$ such that $\Re^0_\alpha = \Re_\alpha$, and $\Re^{i+1}_\alpha$ is the closest successor of $\Re^i_\alpha$ for all $1 \le i < n$.

In this way, we can find all successors $\Re'_\alpha$ of $\Re_\alpha$ such that $\Re'_\alpha \in \mathsf{Succ}_{(r_x, r_y)}(\Re_\alpha)$ if and only if for all $\nu \in \Re_\alpha$ there exists some $t \in \mathbb{R}_{\ge 0}$ such that $\nu + (r_x, r_y)t \in \Re'_\alpha$. Hence, $\mathsf{Reg}(V)$ is indeed a set of regions partitioning $\mathbb{R}^2_{\ge 0}$. ◀

Given two valuations $\nu_1, \nu_2 \in \Re_\alpha$ for some region $\Re_\alpha$, we say that ν_1 and ν_2 are equivalent if they lie in the same region, i.e., $[\nu_1] = [\nu_2]$.

$\mathsf{Reg}(V)$ compatible with resets and guards

▶ **Lemma 17.** $\mathsf{Reg}(V)$ *is compatible with the guards* φ *and with the resets* Res.

Proof.

(1) Let $\Re' \in \mathsf{Res}(\Re)$. Consider $\nu_1, \nu_2 \in \Re$, i.e., $[\nu_1] = [\nu_2]$. Clearly, $\nu_1[x]$ and $\nu_2[x]$ lie in the same interval; same with $\nu_1[y]$ and $\nu_2[y]$. If the operation Res resets x, then $\mathsf{Res}(\nu_1) = \big(0, \nu_1[y]\big)$ and $\mathsf{Res}(\nu_2) = \big(0, \nu_2[y]\big)$. Since $\nu_1[y]$ and $\nu_2[y]$ are in the same interval, we have $[\mathsf{Res}(\nu_1)] = [\mathsf{Res}(\nu_2)]$. Similar results are obtained when y is reset, or when both x, y are reset.

(2) Let $[\nu_1] = [\nu_2]$ be valuations in the same region $\Re$. Let φ be a guard. The result can be proved by structural induction on φ. If φ is atomic of the form $x \sim c$, clearly, $\nu_1 \models \varphi$ if and only if $\nu_2 \models \varphi$, since ν_1 and ν_2 are equivalent. Assume for guards of size $\leq n-1$. It can be seen that the inductive hypothesis can be easily extended to guards of size n. Thus, $\mathsf{Reg}(V)$ is a finite set of regions compatible with guards and resets, partitioning $\mathbb{R}^2_{\geq 0}$. ◀

Hence, we can use the region abstraction for the above set of regions to obtain a region automaton $\mathsf{Reg}(2\text{-}\mathcal{H})$ capturing the untimed language of 2-$\mathcal{H}$. The set of states of such a region automaton is the set $\mathcal{Q} \times \mathsf{Reg}(V)$, where $\mathcal{Q}$ is the set of locations of 2-$\mathcal{H}$. The initial location of $\mathsf{Reg}(2\text{-}\mathcal{H})$ is $\big(l_0, (0,0)\big)$ where $l_0 \in \mathcal{Q}_0$ is the initial location of 2-$\mathcal{H}$. The transitions of $\mathsf{Reg}(2\text{-}\mathcal{H})$ are defined as $(l, \Re) \xrightarrow{a} (l', \Re')$ if and only if there is a region $\hat{\Re}$ and a transition from l to l' on $(\varphi, a, \mathsf{Res})$ in 2-$\mathcal{H}$ such that,

(1) $\hat{\Re} \in \mathsf{Succ}_{(r_x, r_y)}(\Re)$. ($r_x, r_y$ are the rates at the location l),

(2) For all $\nu \in \hat{\Re}$, $\nu \models \varphi$, and

(3) $\mathsf{Res}(\hat{\Re}) = \Re'$

The final states of the region automaton are the states $(f, \Re)$ such that f is a final location of 2-$\mathcal{H}$. It can be seen that the language accepted by this region automaton is indeed the untimed counterpart of $\mathcal{L}(2\text{-}\mathcal{H})$. We thus have, the following result.

▶ **Theorem 18.** *The region automaton construction for singular stopwatch automata 2-$\mathcal{H}$ with two stopwatches is a correct abstraction. For each run ρ in 2-$\mathcal{H}$ from an initial state (l_0, ν_0) to a state (l_n, ν_n), if and only if we have a run ρ' in $\mathsf{Reg}(2\text{-}\mathcal{H})$ from $(l_0, (0,0))$ to $(l_n, \Re_n)$ such that $\nu_n \in \Re_n$.*

The proof is straightforward since in each step, we obtain a new region which is compatible with the constraints and resets of the transition taken.

However, this does not extend to 3 variables, as shown below.

A.2 Problem in extending the region construction to 3 variables

It is an interesting exercise to note where the above region construction fails if we try to extend it to 3 variables with at least one stopwatch. In a nutshell, the problem arises in defining the successor regions of $\Re$ i.e., in defining $\Re_\beta$. The above construction only works for 2 variables and fails if we try to extend the same to 3 variables. The following example helps to illustrate this point. Consider the natural extension of case 4 in the construction above to 3 variables

If $Z = \emptyset$ and one of r_x, $x \in V$ is 0. Here M is the set of variables with the maximum fractional part as before.

$$I'_z = \begin{cases} I_z & \text{if } r_z = 0, \\ [c+1] & \text{if } z \in M, r_z = 1, \text{ and } I_z = (c, c+1) \text{ with } 0 \leq c < c_{\max} \\ I_z & \text{if } z \notin M, r_z = 1, \text{ and } I_z = (c, c+1) \text{ with } 0 \leq c < c_{\max} \end{cases}$$

The problem here arises in constructing the total preorder Rel'. Consider the case when $z \prec x \prec y$ and $r_x = 0$. Consider two distinct valuations ν and ν' such that $\nu[y] = \nu'[y]$ and they belong to the same region $\Re$ in consideration. Now depending on the value of $t = 1 - \mathsf{fract}(\nu[y])$, the successor valuations $\nu_1 = \nu + (0,1,1)t$ and $\nu'_1 = \nu' + (0,1,1)t$ of ν, ν' respectively, might belong to different regions as they might result in different partial orders i.e., $\mathsf{fract}(\nu_1[z]) > \mathsf{fract}(\nu_1[x])$ in one case and $\mathsf{fract}(\nu'_1[z]) < \mathsf{fract}(\nu'_1[x])$

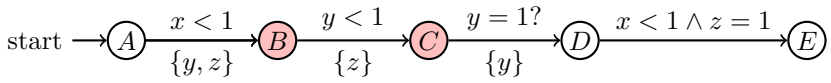


Figure 7 A singular stopwatch automaton with three stopwatches.

in the other. Coming up with such valuations is not difficult and it can be verified by the valuations $\nu[x, y, z] = (0.20, 0.75, 0)$ and $\nu'[x, y, z] = (0.30, 0.75, 0)$. The successor of $\nu'[x, y, z] = (0.30, 0.75, 0)$ after time $1 - 0.75$ is $\nu'_1[x, y, z] = (0.30, 1, 0.25)$, having $z \prec x$ as $0.25 < 0.3$ and the successor of ν after time $1 - 0.75$ is $\nu_1[x, y, z] = (0.20, 1, 0.25)$ having order $x \prec z$ as $0.2 < 0.25$. One might think that further subdividing into smaller regions might help out but as shown in [11] it can be seen that no matter how many such finite number of subdivisions are made this problem will still persist.

This idea can be materialized using a stopwatch automaton with three variables as shown in the Figure 7 and details are given in Example 19.

▶ **Example 19.** We show that any amount of partitioning (finite number) will fail to correctly bi-simulate the given timed automaton.

Note that the shaded nodes B, C have $r_x = 0$ and $r_x = 1$ elsewhere.

Suppose we have partitioned $[0, 1]$ using the n points $p_1, p_2 ..., p_n$ Lets consider the following run of the automaton. We first spend time $t < \frac{p_1}{2}$ in state A and then subsequently time t_1 in state B therefore we will enter state C with clock value $t, t_1, 0$. Due to the continuity of the real line $\exists \delta \in \mathbb{R}_{\geq 0}$, $\exists i$ st. $1 - t - \delta \in (p_i, p_{i+1}) \wedge 1 - t + \delta \in (p_i, p_{i+1})$ Now consider the two clock valuations $\nu = (t, 1 - t - \delta, 0)$ and $\nu' = (t, 1 - t + \delta, 0)$ using the above assertion clearly ν and ν' belong to the same region $\big((0, p_1), (p_i, p_{i+1}), [0]\big)$. Hence they have the same successor and should behave in exactly the same way if the bi-simulation is correct. Now say we enter state C with these clock values we then spend $t \pm \delta$ time in C and enter D with valuation $(t, 0, t \pm \delta)$ now only the valuation $(t, 0, t + \delta)$ will allow us to take the edge to enter E hence clearly these valuations differ in there behavior but the current partitioning is not fine enough to distinguish between them.