

# 27th International Symposium on Temporal Representation and Reasoning

TIME 2020, September 23–25, 2020, Bozen-Bolzano, Italy

Edited by

Emilio Muñoz-Velasco

Ana Ozaki

Martin Theobald



*Editors*

**Emilio Muñoz-Velasco** 

University of Málaga, Spain  
ejmunoz@uma.es

**Ana Ozaki** 

University of Bergen, Norway  
Ana.Ozaki@uib.no

**Martin Theobald**

University of Luxembourg, Luxembourg  
martin.theobald@uni.lu

*ACM Classification 2012*

Theory of computation → Logic; Information systems → Temporal data; Computing methodologies → Knowledge representation and reasoning

**ISBN 978-3-95977-167-2**

*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/978-3-95977-167-2>.

*Publication date*

September, 2020

*Bibliographic information published by the Deutsche Nationalbibliothek*

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

*License*

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0): <https://creativecommons.org/licenses/by/3.0/legalcode>.



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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.TIME.2020.0

**ISBN 978-3-95977-167-2**

**ISSN 1868-8969**

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

## LIPICs – Leibniz International Proceedings in Informatics

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

### *Editorial Board*

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

**ISSN 1868-8969**

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





## ■ Contents

Preface	
<i>Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald</i> .....	0:vii
Authors	
.....	0:ix–0:xi
PC Members	
.....	0:xiii

### Invited Talks

Verifying Autonomous Robots: Challenges and Reflections	
<i>Clare Dixon</i> .....	1:1–1:4
Temporal Modalities in Answer Set Programming	
<i>Pedro Cabalar</i> .....	2:1–2:5
Time and Business Process Management: Problems, Achievements, Challenges	
<i>Johann Eder and Marco Franceschetti</i> .....	3:1–3:8

### Regular Papers

Negotiating Temporal Commitments in Cross-Organizational Business Processes	
<i>Marco Franceschetti and Johann Eder</i> .....	4:1–4:15
The Horn Fragment of Branching Algebra	
<i>Alessandro Bertagnon, Marco Gavanelli, Alessandro Passantino,</i> <i>Guido Sciavicco, and Stefano Trevisani</i> .....	5:1–5:16
Temporal Logic with Recursion	
<i>Florian Bruse and Martin Lange</i> .....	6:1–6:14
Parametric Model Checking Continuous-Time Markov Chains	
<i>Catalin-Andrei Ilie and James Ben Worrell</i> .....	7:1–7:18
Universal Solutions in Temporal Data Exchange	
<i>Zehui Cheng and Phokion G. Kolaitis</i> .....	8:1–8:17
Knowledge Extraction with Interval Temporal Logic Decision Trees	
<i>Guido Sciavicco and Ionel Eduard Stan</i> .....	9:1–9:16
Window-Slicing Techniques Extended to Spanning-Event Streams	
<i>Aurélie Suzanne, Guillaume Raschia, José Martínez, and Damien Tassetti</i> .....	10:1–10:14
Mining Significant Temporal Networks Is Polynomial	
<i>Guido Sciavicco, Matteo Zavatteri, and Tiziano Villa</i> .....	11:1–11:12
Dynamic Branching in Qualitative Constraint Networks via Counting Local Models	
<i>Michael Sioutis and Diedrich Wolter</i> .....	12:1–12:15

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald

Leibniz International Proceedings in Informatics



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

Non-Simultaneity as a Design Constraint <i>Jean Guyomarc'h, François Guerret, Bilal El Mejjati, Emmanuel Ohayon, Bastien Vincke, and Alain Mérigot</i> .....	13:1–13:15
One-Pass Context-Based Tableaux Systems for CTL and ECTL <i>Alex Abuin, Alexander Bolotov, Montserrat Hermo, and Paqui Lucio</i> .....	14:1–14:20
TESL: A Model with Metric Time for Modeling and Simulation <i>Hai Nguyen Van, Frédéric Boulanger, and Burkhart Wolff</i> .....	15:1–15:15
Complexity of Qualitative Timeline-Based Planning <i>Dario Della Monica, Nicola Gigante, Salvatore La Torre, and Angelo Montanari</i> .	16:1–16:13
A Note on $C^2$ Interpreted over Finite Data-Words <i>Bartosz Bednarczyk and Piotr Witkowski</i> .....	17:1–17:14
Stab-Forests: Dynamic Data Structures for Efficient Temporal Query Processing <i>Jelle Hellings and Yuqing Wu</i> .....	18:1–18:19
On the Decidability of a Fragment of preferential LTL <i>Anasse Chafik, Fahima Cheikh-Alili, Jean-François Condotta, and Ivan Varzinczak</i>	19:1–19:19

## ■ Preface

The 27th International Symposium on Temporal Representation and Reasoning (TIME 2020) was planned to be in Bozen-Bolzano, Italy, from the 23rd to the 24th of September, 2020. However, due to the special circumstances related to COVID-19, the conference was held virtually. This year's edition was organized as a part of the Bolzano Summer of Knowledge (BOSK 2020). TIME is a well-established symposium series that brings together researchers interested in reasoning about temporal aspects of information in all areas of computer science. The symposium aims to be interdisciplinary and to attract attendees from artificial intelligence, database management, logic and verification, and beyond.

TIME 2020 received 23 research paper submissions from Austria, Canada, Germany, India, Italy, Poland, Romania, Russia, South Africa, Spain, United Kingdom, and the United States, written by 59 different authors. We would like to thank all authors for submitting outstanding contributions to the conference. The submissions were carefully evaluated by the 23 members of the program committee, who deserve warm thanks for their high-quality and timely handling of the reviews. Each submission was reviewed by at least three PC members. In the end, 16 contributions were accepted for inclusion in the conference proceedings and presentation at the conference.

The TIME 2020 scientific program includes also three keynote presentations given, respectively, by Clare Dixon (University of Manchester, United Kingdom), Pedro Cabalar (University of Corunna, Spain), and Johann Eder (Alpen-Adria Universität Klagenfurt, Austria). We are delighted that all of them accepted our invitations and we are very grateful for their scientific contribution.

Finally, we would like to acknowledge the excellent work of all the people involved in the organization of the conference, in particular, the local chairs Alessandro Artale and Johann Gamper. Their support was indispensable for a smooth organization and preparation of the conference. Deep thanks also to Dr. Wagner and the LIPICs team for the support during the preparation of the conference proceedings.

Emilio Muñoz-Velasco  
Ana Ozaki  
Martin Theobald





## ■ List of Authors

Alain Mérigot  
Université Paris-Saclay, CNRS  
France  
alain.merigot@universite  
-paris-saclay.fr

Alessandro Bertagnon  
University of Ferrara  
Italy  
alessandro.bertagnon@unife.it

Alessandro Passantino  
Università di Ferrara  
Italy  
alessandr.passantino@student.  
unife.it

Alex Abuin Yepes  
Ikerlan Technology Research Centre  
Spain  
aabuin@ikerlan.es

Alexander Bolotov  
University of Westminster  
United Kingdom  
a.bolotov@westminster.ac.uk

Anasse Chafik  
CRIL, Université d'Artois; CNRS  
France  
chafik@cril.fr

Angelo Montanari  
University of Udine  
Italy  
angelo.montanari@uniud.it

Aurélie Suzanne  
LS2N  
France  
aurelie.suzanne@etu.univ-nantes.fr

Bartosz Bednarczyk  
University of Wrocław & TU Dresden  
Poland  
bartosz.bednarczyk@cs.uni.wroc.pl

Bastien Vincke  
Université Paris-Saclay, CNRS  
France  
bastien.vincke@universite  
-paris-saclay.fr

Bilal El Mejjati  
Krono-Safe  
France  
bilal.elmejjati@krono-safe.com

Burkhard Wolff  
Université Paris-Saclay, LRI  
France  
wolff@lri.fr

Catalin-Andrei Ilie  
University of Bucharest  
Romania  
cilie@fmi.unibuc.ro

Damien Tassetti  
Polytech Nantes  
France  
damien.tassetti@etu.univ-nantes.fr

Dario Della Monica  
Università degli Studi di Udine  
Italy  
dario.dellamonica@uniud.it

Diedrich Wolter  
University of Bamberg  
Germany  
diedrich.wolter@uni-bamberg.de


Emmanuel Ohayon  
Krono-Safe  
France  
emmanuel.ohayon@krono-safe.com

Fahima Cheikh-Alili  
CRIL, Université d'Artois; CNRS  
France  
cheikh@cril.fr

Florian Bruse  
Universität Kassel  
Germany  
florian.bruse@uni-kassel.de

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald

 Leibniz International Proceedings in Informatics

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

François Guerret  
Krono-Safe  
France  
francois.guerret@krono-safe.com

Frédéric Boulanger  
CentraleSupélec, Université Paris-Saclay,  
LRI  
France  
frederic.boulanger@lri.fr

Guido Sciavicco  
University of Ferrara, Italy  
Italy  
guido.sciavicco@unife.it

Guido Sciavicco  
University of Ferrara, Italy  
Italy  
guido.sciavicco@unife.it

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

Guillaume Raschia  
LS2N  
France  
guillaume.raschia@univ-nantes.fr

Hai Nguyen Van  
Université Paris-Saclay, LRI  
France  
hai.nguyenvan.phie@gmail.com

Ionel Eduard Stan  
University of Ferrara, Italy, and  
University of Parma, Italy  
Italy  
ioneleduard.stan@unife.it

Ivan Varzinczak  
Univ. Artois and CNRS  
France  
ijv@acm.org

James Worrell  
University of Oxford  
United Kingdom  
jbw@cs.ox.ac.uk

Jean Guyomarc'H  
Krono-Safe  
France  
jean.guyomarch@krono-safe.com

Jean-François Condotta  
CRIL  
France  
j\_condotta@yahoo.fr

Jelle Hellings  
University of California, Davis  
United States  
jhellings@ucdavis.edu

Johann Eder  
Alpen-Adria-Universitaet Klagenfurt  
Austria  
johann.eder@aau.at

José Martinez  
LS2N  
France  
jose.martinez@univ-nantes.fr

Marco Franceschetti  
Alpen-Adria-Universitaet Klagenfurt  
Austria  
marco.franceschetti@aau.at

Marco Gavanelli  
University of Ferrara  
Italy  
marco.gavanelli@unife.it

Martin Lange  
University of Kassel  
Germany  
martin.lange@uni-kassel.de

Matteo Zavatteri  
Università degli Studi di Verona  
Italy  
matteo.zavatteri@gmail.com

Michael Sioutis  
University of Bamberg  
Germany  
michael\_sioutis@hotmail.com

Montserrat Hermo  
The Universtiy of the Basque Country  
(UPV/EHU)  
Spain  
`montserrat.hermo@ehu.es`

Nicola Gigante  
University of Udine, Italy  
Italy  
`nicola.gigante@uniud.it`

Paqui Lucio  
University of the Basque Country  
Spain  
`paqui.lucio@ehu.es`

Phokion Kolaitis  
UC Santa Cruz and IBM Research -  
Almaden  
United States  
`kolaitis@ucsc.edu`

Piotr Witkowski  
University of Wrocław  
Poland  
`pwit@ii.uni.wroc.pl`

Salvatore La Torre  
University of Salerno, Italy  
Italy  
`slatorre@unisa.it`

Stefano Trevisani  
University of Ferrara  
Italy  
`stefan.trevisani@student.unife.it`

Tiziano Villa  
Universita' di Verona  
Italy  
`tiziano.villa@univr.it`

Yuqing Wu  
Pomona College  
United States  
`Melanie.Wu@pomona.edu`

Zehui Cheng  
UC Santa Cruz  
United States  
`zecheng@ucsc.edu`





## ■ List of PC Members

Alexander Artikis

Klaus Berberich

Camille Bourgaux

Panagiotis Bouros

Anton Dignös

Clare Dixon

Johann Gamper

Rajeev Gore

Víctor Gutiérrez-Basulto

Fredrik Heintz

Jean Christoph Jung

Roman Kontchakov

Cláudia Nalon

Daniel Neider

Kjetil Nørvåg

Manuel Ojeda-Aciego

Hector Pomares

Dimitris Sacharidis

Stephan Schlüter

Guido Sciavicco



# Verifying Autonomous Robots: Challenges and Reflections

Clare Dixon 

Department of Computer Science, The University of Manchester, UK  
<https://www.research.manchester.ac.uk/portal/clare.dixon.html>  
clare.dixon@manchester.ac.uk

---

## Abstract

Autonomous robots such as robot assistants, healthcare robots, industrial robots, autonomous vehicles etc. are being developed to carry out a range of tasks in different environments. The robots need to be able to act autonomously, choosing between a range of activities. They may be operating close to or in collaboration with humans, or in environments hazardous to humans where the robot is hard to reach if it malfunctions. We need to ensure that such robots are reliable, safe and trustworthy. In this talk I will discuss experiences from several projects in developing and applying verification techniques to autonomous robotic systems. In particular we consider: a robot assistant in a domestic house, a robot co-worker for a cooperative manufacturing task, multiple robot systems and robots operating in hazardous environments.

**2012 ACM Subject Classification** Computer systems organization → Dependable and fault-tolerant systems and networks; Software and its engineering → Software verification and validation; Theory of computation → Logic

**Keywords and phrases** Verification, Autonomous Robots

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.1

**Category** Invited Talk

**Funding** *Clare Dixon*: This work was funded by the Engineering and Physical Sciences Research Council (EPSRC) under the grants Trustworthy Robot Systems (EP/K006193/1) and Science of Sensor Systems Software (S4 EP/N007565/1) and by the UK Industrial Strategy Challenge Fund (ISCF), delivered by UKRI and managed by EPSRC under the grants Future AI and Robotics Hub for Space (FAIR-SPACE EP/R026092/1) and Robotics and Artificial Intelligence for Nuclear (RAIN EP/R026084/1).

**Acknowledgements** The work discussed in this document was carried out collaboratively with researchers on the following funded research projects: Trustworthy Robot Systems<sup>1</sup>; Science of Sensor Systems Software<sup>2</sup>; Future AI and Robotics Hub for Space<sup>3</sup>; and Robotics and Artificial Intelligence for Nuclear<sup>4</sup>.

## 1 Formal Verification of Autonomous Robots

Autonomous robots are being developed for many purposes across society. These may be autonomous cars or pods, home robot assistants, warehouse robots, museum guides, delivery robots, companion robots, agricultural robots etc. Whilst these robots have the potential to be of great use to society, improving our lives, we need to make sure that they are reliable, safe, robust and trustworthy. We discuss work from several projects about experiences verifying autonomous robots.

---

<sup>1</sup> [www.robosafe.org](http://www.robosafe.org)

<sup>2</sup> [www.dcs.gla.ac.uk/research/S4/](http://www.dcs.gla.ac.uk/research/S4/)

<sup>3</sup> [www.fairspacehub.org](http://www.fairspacehub.org)

<sup>4</sup> [rainhub.org.uk](http://rainhub.org.uk)



© Clare Dixon;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 1; pp. 1:1–1:4

Leibniz International Proceedings in Informatics



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

Formal verification is a mathematical analysis of all behaviours using logics and tools such as theorem provers or model checkers. Formal verification is often applied to an abstraction of the real system to obtain a discrete and finite system that is not too large. There has been recent interest in applying formal verification to autonomous robot systems, see for example [14] for an overview.

Here we focus on temporal verification using automatic tools and techniques such as model checking and deduction (see for example [8]) that do not require user interaction. Model checking [4, 12, 3] is a fully automatic, algorithmic technique for verifying the temporal properties of systems. Input to the model checker is a model of the system and a property to be checked on that model specified in temporal logic. For temporal deduction both the system ( $S$ ) and the property ( $P$ ) are specified in logic and a calculus is applied to check that  $P$  is a logical consequence of  $S$ .

## **2** Domestic Robot Assistants

Robots can be used in healthcare or elderly care environments enabling people who need assistance to continue living in their own homes. Such robots can help carrying things, provide reminders to drink water or take medicine, inform the user when something in the house needs attention such as the doorbell is ringing or the bath is overflowing, or provide alerts to care givers if the person is not responding.

We considered a personal robot assistant located in the robot house at the University of Hertfordshire [16]. The house has sensors that provide information to the robot about the kettle or the television being on, the fridge door being left open, someone sitting on the sofa etc. The robot is controlled by a set of “if-then” rules (with priorities) where the “if” part checks whether a condition on the sensor data or internal Boolean flags is satisfied and the “then” part contains actions for the robot to execute and flags to be set. We modelled the decision making rules using a model checker, checking properties relating to the expected execution of the rules. We considered two different approaches: one modelling the system using Brahms [17] a human agent modelling language translated into the model checker Spin [20, 19]; and the other via direct modelling in the NuSMV model checker [5, 9]. Experiments with real robots were also carried out considering whether mistakes by the robot affected people’s trust in them [15].

## **3** Collaborative Manufacture

A second use case for robot assistants is in collaborative manufacture. We considered a robot co-worker with a scenario of a handover task for collaboratively constructing a table [21]. We considered this scenario using three types of verification: formal verification using model checking, simulation based testing [1] and user experiments with the robot.

We modelled the scenario using probabilistic timed automata and carried out verification using the PRISM model checker [11]. Simulation based testing involved developing a simulation of the system under consideration and generating and executing tests systematically that cover interesting parts of the state space. Also experiments with the robot and users were carried out. Issues found using one form of verification were used to improve the models and therefore the verification for the others. Some properties were more amenable to verification with some of these methods rather than others.

## 4 Swarm Robots and Wireless Sensor Networks

A robot swarm is a collection of simple, often identical, robots working together to carry out some task. Each robot has a small set of behaviours, is typically able to interact with other nearby robots and with its environment usually without a central controller. Robot swarms are often thought to be fault tolerant in that it may be possible to design a swarm so that the failure of some of the robots will not endanger the success of the overall mission. Wireless sensor networks are similar being a collection of simple, often identical, sensors with no centralised control.

We have modelled and verified properties of swarm robots relating to coherence [6] and energy optimisation for foraging robots [13] and relating to synchronisation properties for wireless sensor networks [10, 18] enabling us to detect cases and parameter settings where the required property does not (or is unlikely to) hold.

## 5 Robots in Dangerous Environments

In certain environments it may be preferable or we may need to use robots to carry out tasks because they are dangerous or hard to access for example underwater, space or nuclear clear up. Ensuring such robots are reliable and robust is particularly important as we may not be able to access them to reset or repair them if they go wrong. Here we advocate using different types of verification for different components [7, 2].

## 6 Conclusions

We have discussed experiences in verifying autonomous robots from robot assistants in the home and for collaborative manufacture, to swarm robots and robots in hazardous environments. We believe that the decision making aspects of the robot should be separated from other components to allow verification and explainability of the decisions made. We advocate using a range of verification techniques including formal verification, simulation based testing and end user experiments to improve the reliability of such systems. Different components or different types of property may require different types of verification. We believe better verification will lead to improvements not only in their safety and reliability but may also be used as evidence of this to regulators and improve trust in them by the public. Many challenges remain including how to verify systems that learn, designing systems in a modular way amenable to verification, modelling the environment, and how to provide evidence to certify autonomous robotics.

---

### References

- 1 D. Araiza-Illan, D. Western, A. G. Pipe, and K. Eder. Coverage-driven verification - an approach to verify code for robots that directly interact with humans. In N. Piterman, editor, *Hardware and Software: Verification and Testing - 11th International Haifa Verification Conference, HVC 2015, Haifa, Israel, November 17-19, 2015, Proceedings*, volume 9434 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2015.
- 2 R. C. Cardoso, M. Farrell, M. Luckcuck, A. Ferrando, and M. Fisher. Heterogeneous verification of an autonomous curiosity rover. In *Proceedings of the NASA Formal Methods Symposium*. Springer, 2020.
- 3 A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In

- Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.
- 4 E. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
  - 5 C. Dixon, M. Webster, J. Saunders, M. Fisher, and K. Dautenhahn. “The Fridge Door is Open”-Temporal Verification of a Robotic Assistant’s Behaviours. In M. Mistry, A. Leonardis, M. Witkowski, and C. Melhuish, editors, *Advances in Autonomous Robotics Systems*, volume 8717 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2014.
  - 6 C. Dixon, A.F.T. Winfield, M. Fisher, and C. Zeng. Towards Temporal Verification of Swarm Robotic Systems. *Robotics and Autonomous Systems*, 2012.
  - 7 M. Farrell, R. C. Cardoso, L. A. Dennis, C. Dixon, M. Fisher, G. Kourtis, A. Lisitsa, M. Luckcuck, and M. Webster. Modular verification of autonomous space robotics. In *Assurance of Autonomy for Robotic Space Missions Workshop*, 2019.
  - 8 M. Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic*. Wiley, 2011.
  - 9 P. Gainer, C. Dixon, K. Dautenhahn, M. Fisher, U. Hustadt, J. Saunders, and M. Webster. Cruton: Automatic verification of a robotic assistant’s behaviours. In L. Petrucci, C. Seceleanu, and A. Cavalcanti, editors, *Critical Systems: Formal Methods and Automated Verification, Proceedings of FMICS-AVoCS*, volume 10471 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 2017. doi:10.1007/978-3-319-67113-0\_8.
  - 10 P. Gainer, S. Linker, C. Dixon, U. Hustadt, and M. Fisher. Multi-Scale Verification of Distributed Synchronisation. *Formal Methods in System Design*, 2020.
  - 11 A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
  - 12 G. J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2003.
  - 13 S. Konur, C. Dixon, and M. Fisher. Analysing Robot Swarm Behaviour via Probabilistic Model Checking. *Robotics and Autonomous Systems*, 60(2):199–213, 2012.
  - 14 M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher. Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys*, 52(5), 2019.
  - 15 M. Salem, G. Lakatos, F. Amirabdollahian, and K. Dautenhahn. Would You Trust a (Faulty) Robot?: Effects of Error, Task Type and Personality on Human-Robot Cooperation and Trust. In *International Conference on Human-Robot Interaction (HRI)*, pages 141–148, Portland, Oregon, USA, 2015. ACM/IEEE.
  - 16 J. Saunders, N. Burke, K. L. Koay, and K. Dautenhahn. A User Friendly Robot Architecture for Re-ablement and Co-learning in A Sensorised Home. In *European AAATE (Associated for the Advancement of Assistive Technology in Europe) Conference*, Vilamoura, Portugal, 2013.
  - 17 M. Sierhuis, W. J. Clancey, and R. J. J. van Hoof. Brahms an agent-oriented language for work practice simulation and multi-agent systems development. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *Multi-Agent Programming, Languages, Tools and Applications*, pages 73–117. Springer, 2009.
  - 18 M. Webster, M. Breza, C. Dixon, M. Fisher, and J. McCann. Exploring the effects of environmental conditions and design choices on IOT systems using formal methods. *Journal of Computational Science*, 2020.
  - 19 M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. Koay, and K. Dautenhahn. Formal verification of an autonomous personal robotic assistant. In *Proceedings of Workshop on Formal Verification in Human Machine Systems (FVHMS)*. AAAI, 2014.
  - 20 M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. L. Koay, K. Dautenhahn, and J. Saez-Pons. Toward Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study. *IEEE Transactions on Human-Machine Systems*, 46(2):186–196, April 2016.
  - 21 M. Webster, D. Western, D. Araiza-Illan, C. Dixon, K. Eder, M. Fisher, and A. Pipe. A corroborative approach to verification and validation of human–robot teams. *International Journal of Robotics Research*, 39(1):73–99, 2020.

# Temporal Modalities in Answer Set Programming

Pedro Cabalar 

University of Corunna, Spain

cabalar@udc.es

---

## Abstract

Based on the answer set (or stable model) semantics for logic programs, Answer Set Programming (ASP) has become one of the most successful paradigms for practical Knowledge Representation and problem solving. Although ASP is naturally equipped for solving static combinatorial problems up to NP complexity (or  $\Sigma_2^P$  in the disjunctive case) its application to temporal scenarios has been frequent since its very beginning, partly due to its early use for reasoning about actions and change. Temporal problems normally suppose an extra challenge for ASP for several reasons. On the one hand, they normally raise the complexity (in the case of classical planning, for instance, it becomes PSPACE-complete), although this is usually accounted for by making repeated calls to an ASP solver. On the other hand, temporal scenarios also pose a representational challenge, since the basic ASP language does not support temporal expressions. To fill this representational gap, a temporal extension of ASP called Temporal Equilibrium Logic (TEL) was proposed in and extensively studied later. This formalism constitutes a modal, linear-time extension of Equilibrium Logic which, in its turn, is a complete logical characterisation of (standard) ASP based on the intermediate logic of Here-and-There (HT). As a result, TEL is an expressive non-monotonic modal logic that shares the syntax of Linear-Time Temporal Logic (LTL) but interprets temporal formulas under a non-monotonic semantics that properly extends stable models.

**2012 ACM Subject Classification** Theory of computation → Modal and temporal logics; Theory of computation → Constraint and logic programming; Computing methodologies → Nonmonotonic, default reasoning and belief revision; Computing methodologies → Logic programming and answer set programming; Computing methodologies → Temporal reasoning

**Keywords and phrases** Logic Programming, Temporal Logic, Answer Set Programming, Modal Logic

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.2

**Category** Invited Talk

**Funding** *Pedro Cabalar*: research partially supported by MINECO (grant TIN2017-84453-P) and Xunta de Galicia (grant GPC ED431B 2019/03), Spain.

**Acknowledgements** This document is a summary of a long term project jointly developed by the Knowledge Representation group (inside IRLab) at the University of Corunna, Spain, led by Pedro Cabalar and the Potassco group at the University of Potsdam, Germany, directed by Torsten Schaub. This includes joint work with, among others, Felicidad Aguado, Martín Diéguez, Roland Kaminski, François Laferrière, Philip Morkisch, Gilberto Pérez, Anna Schuhmann and Concepción Vidal. Authors from other universities that have undoubtedly contributed to the project are David Pearce, Philip Balbiani, Luis Fariñas and Jorge Fandinno.

## EXTENDED ABSTRACT

Based on the answer set (or stable model) semantics [12] for logic programs, *Answer Set Programming* [4] (ASP) has become one of the most successful paradigms for practical Knowledge Representation and problem solving. Although ASP is naturally equipped for solving static combinatorial problems up to NP complexity (or  $\Sigma_2^P$  in the disjunctive case) its application to temporal scenarios has been frequent since its very beginning, partly due to its early use for reasoning about actions and change [13]. Temporal problems normally suppose an extra challenge for ASP for several reasons. On the one hand, they normally raise the



© Pedro Cabalar;

licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 2; pp. 2:1–2:5

Leibniz International Proceedings in Informatics



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

complexity (in the case of classical planning, for instance, it becomes PSPACE-complete [5]), although this is usually accounted for by making repeated calls to an ASP solver. On the other hand, temporal scenarios also pose a representational challenge, since the basic ASP language does not support temporal expressions.

To fill this representational gap, a temporal extension of ASP called *Temporal Equilibrium Logic* (TEL) was proposed in [7] and extensively studied later on [1]. This formalism constitutes a modal, linear-time extension of *Equilibrium Logic* [15] which, in its turn, is a complete logical characterisation of (standard) ASP based on the intermediate logic of *Here-and-There* (HT) [14]. As a result, TEL is an expressive non-monotonic modal logic that shares the syntax of Linear-Time Temporal Logic (LTL) [16] but interprets temporal formulas under a non-monotonic semantics that properly extends stable models. This semantics is based on the idea of selecting some LTL temporal models of a theory  $\Gamma$  that satisfy some minimality condition, when examined under the weaker logic of temporal HT (THT). Thus, a temporal stable model of  $\Gamma$  is a kind of selected LTL model of  $\Gamma$ , and so, it has the form of an infinite sequence of states, usually called a *trace*. To put an example, the Yale Shooting scenario  $\square$  where we must shoot a loaded gun to kill a turkey, can be encoded in TEL as:

$$\square(\text{loaded} \wedge \circ\text{shoot} \rightarrow \circ\text{dead}) \quad (1)$$

$$\square(\text{loaded} \wedge \circ\text{shoot} \rightarrow \circ\text{unloaded}) \quad (2)$$

$$\square(\text{load} \rightarrow \text{loaded}) \quad (3)$$

$$\square(\text{dead} \rightarrow \circ\text{dead}) \quad (4)$$

$$\square(\text{loaded} \wedge \neg\circ\text{unloaded} \rightarrow \circ\text{loaded}) \quad (5)$$

$$\square(\text{unloaded} \wedge \neg\circ\text{loaded} \rightarrow \circ\text{unloaded}) \quad (6)$$

In this way, under TEL semantics, implication  $\alpha \rightarrow \beta$  has a similar behaviour to a directional inference rule, normally reversed as  $\beta \leftarrow \alpha$  or  $\beta :- \alpha$  in logic programming notation. The last two rules, (5)-(6), encode the *inertia law* for fluents *loaded* and *unloaded*, respectively. Note the use of  $\neg$  in these two rules: it actually corresponds to *default negation*, that is,  $\neg\alpha$  is read as “there is no evidence about  $\alpha$ .” For instance, (5) is read as “if the gun was loaded and we cannot prove that it will become unloaded then it stays loaded.”

Computation of temporal stable models is a complex task. THT-satisfiability has been classified [8] as PSPACE-complete, that is, the same complexity as LTL-satisfiability, whereas TEL-satisfiability rises to EXPSPACE-completeness, as proved in [3]. In this way, we face a similar situation as in the non-temporal case where HT-satisfiability is NP-complete like SAT, whereas existence of equilibrium model (for arbitrary theories) is  $\Sigma_2^P$ -complete (like disjunctive ASP). There exist a pair of tools, **STeLP** [6] and **ABSTEM** [9], that allow computing (infinite) temporal stable models (represented as Büchi automata). These tools can be used to check verification properties that are usual in LTL, like the typical safety, liveness and fairness conditions, but in the context of temporal ASP. Moreover, they can also be applied for planning problems that involve an indeterminate or even infinite number of steps, such as the non-existence of a plan. The tool **ABSTEM** also accepts pairs of theories to decide different types of equivalence: LTL-equivalence, TEL-equivalence (i.e. coincidence in the set of TS-models) and strong equivalence (i.e., THT-equivalence). Moreover, when strong equivalence fails, **ABSTEM** obtains a context, that is, an additional formula that added to the compared theories makes them behave differently.

The original definition of TEL was thought as a direct non-monotonic extension of standard LTL, so that models had the form of infinite traces. However, this rules out computation by ASP technology and is unnatural for applications like planning, where plans amount to finite prefixes of one or more traces [11]. In a recent line of research [10], TEL



was extended to cope with finite traces (which are closer to ASP computation). On the one hand, this amounts to a restriction of THT and TEL to finite traces. On the other hand, this is similar to the restriction of LTL to  $LTL_f$  advocated by [11]. Our new approach, dubbed  $TEL_f$ , has the following advantages. First, it is readily implementable via ASP technology. Second, it can be reduced to a normal form which is close to logic programs and much simpler than the one obtained for TEL. Finally, its temporal models are finite and offer a one-to-one correspondence to plans. Interestingly,  $TEL_f$  also sheds light on concepts and methodology used in incremental ASP solving when understanding incremental parameters as time points.

Another distinctive feature of  $TEL_f$  is the inclusion of future as well as past temporal operators. When using the causal reading of program rules, it is generally more natural to draw upon the past in rule bodies and to refer to the future in rule heads. As well, past operators are much easier handled computationally than their future counterparts when it comes to incremental reasoning, since they refer to already computed knowledge.

$TEL_f$  is implemented in the `telingo` system, extending the ASP system `clingo` to compute the temporal stable models of (non-ground) temporal logic programs. To this end, it extends the full-fledged input language of `clingo` with temporal operators and computes temporal models incrementally by multi-shot solving using a modular translation into ASP. `telingo` is freely available at [github](https://github.com/potassco/telingo)<sup>1</sup>. The interested reader might have a good time playing with the examples given in the `examples` folder at the same site. For instance, under `telingo` syntax, our theory (1)-(6) would be represented<sup>2</sup> as

```
#program dynamic.
dead :- shoot, 'loaded.
unloaded :- shoot, 'unloaded.
loaded :- load.
dead :- 'dead.
loaded :- 'loaded, not unloaded.
unloaded :- 'unloaded, not loaded.
```

The `telingo` input language actually allows the introduction of arbitrary LTL formulas in constraints or past formulas in the rule bodies (conditions).

Similar to the extension of  $LTL_f$  to its (linear) dynamic logic counterpart  $LDL_f$  [11], we just introduced in [2] a dynamic extension of HT that draws upon this linear version of dynamic logic. We refer to the resulting logic as *(Linear) Dynamic logic of Here-and-There* (DHT for short). As usual, the equilibrium models of DHT are used to define temporal stable models and induce the non-monotonic counterpart of DHT, referred to as *(Linear) Dynamic Equilibrium Logic* (DEL). In doing so, we actually parallel earlier work extending HT with LTL, ultimately leading to THT and TEL. To put an example in DEL, the formula  $[\neg help^*](\neg help \rightarrow sos)$  behaves as a logic program rule that repeats sending an *sos* while no evidence of *help* has been received along a sequence of states. DEL is general enough to cover LDL, as it shares the same syntax but introduces non-monotonicity with the definition of temporal stable models. It also covers LTL and TEL as particular cases, since LTL temporal operators can be defined as particular cases of DEL expressions: for instance  $\Box\alpha$  (i.e.  $\alpha$  always holds) can be represented in DEL as  $[\top^*]\alpha$ . The satisfiability problem in DEL is EXPSpace-complete; it thus coincides with that of TEL but goes beyond that of LDL and LTL, both being PSPACE-complete.

<sup>1</sup> <https://github.com/potassco/telingo>

<sup>2</sup> The left upper commas are read as *previously* and correspond to the past operator dual of *next* “ $\circ$ ”. The  $\Box$  operator is implicit in all dynamic rules.

These recent results open several interesting topics for future study. First, the version of DEL for finite traces,  $DEL_f$ , seems a natural step to follow, similar to the relation of LDL and  $LDL_f$ . We plan to propose and analyse this variation in an immediate future. As a second open topic, it would be interesting to adapt existing model checking techniques (based on automata construction) for temporal logics to solve the problem of existence of temporal stable models. This was done for infinite traces in [8, 6], but no similar method has been implemented for finite traces on  $TEL_f$  or  $DEL_f$  yet. The importance of having an efficient implementation of such a method is that it would allow deciding non-existence of a plan in a given planning problem, something not possible by current incremental solving techniques. Another interesting topic is the optimization of grounding in temporal ASP specifications as those handled by `telingo`. The current grounding of `telingo` is inherited from incremental solving in `clingo` and does not exploit the semantics of temporal expressions that are available now in the input language. Finally, we envisage to extend the `telingo` system with features of DEL in order to obtain a powerful system for representing and reasoning about dynamic domains, not only providing an effective implementation of TEL and DEL but, furthermore, a platform for action and control languages.

---

## References

- 1 F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, and C. Vidal. Temporal equilibrium logic: a survey. *Journal of Applied Non-Classical Logics*, 23(1-2):2–24, 2013.
- 2 A. Bosser, P. Cabalar, M. Diéguez, and T. Schaub. Introducing temporal stable models for linear dynamic logic. In M. Thielscher, F. Toni, and F. Wolter, editors, *Proceedings of the Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'18)*, pages 12–21. AAAI Press, 2018.
- 3 Laura Bozzelli and David Pearce. On the complexity of temporal equilibrium logic. In *Proceedings of the 30th Annual ACM/IEEE Symposium of Logic in Computer Science (LICS'15)*, Kyoto, Japan, 2015. (to appear).
- 4 G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
- 5 Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994. doi:10.1016/0004-3702(94)90081-7.
- 6 P. Cabalar and M. Diéguez. STELP - a tool for temporal answer set programming. In *LPNMR'11*, volume 6645 of *Lecture Notes in Computer Science*, pages 370–375, 2011.
- 7 P. Cabalar and G. Perez. Temporal Equilibrium Logic: A First Approach. In *Proceedings of the 11<sup>th</sup> International Conference on Computer Aided Systems Theory (EUROCAST'07)*, page 241–248, 2007.
- 8 Pedro Cabalar and Stéphane Demri. Automata-based computation of temporal equilibrium models. In *21st International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'11)*, 2011.
- 9 Pedro Cabalar and Martín Diéguez. Strong equivalence of non-monotonic temporal theories. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*, Vienna, Austria, 2014.
- 10 Pedro Cabalar, Roland Kaminski, Torsten Schaub, and Anna Schuhmann. Temporal answer set programming on finite traces. *Theory and Practice of Logic Programming*, 18(3-4):406–420, 2018.
- 11 G. De Giacomo and M. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In F. Rossi, editor, *Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 854–860. IJCAI/AAAI Press, 2013.

- 12 M. Gelfond and V. Lifschitz. The Stable Model Semantics For Logic Programming. In *Proc. of the 5<sup>th</sup> International Conference on Logic Programming (ICLP'88)*, page 1070–1080, Seattle, Washington, 1988.
- 13 Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17(2/3&4):301–321, 1993.
- 14 A. Heyting. Die formalen Regeln der intuitionistischen Logik. *Sitzungsberichte der Preussischen Akademie der Wissenschaften. Physikalisch-mathematische Klasse*, 1930.
- 15 D. Pearce. A New Logical Characterisation of Stable Models and Answer Sets. In *Proc. of Non-Monotonic Extensions of Logic Programming (NMELP'96)*, pages 57–70, Bad Honnef, Germany, 1996.
- 16 A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.



# Time and Business Process Management: Problems, Achievements, Challenges

Johann Eder 

Department of Informatics Systems, Universität Klagenfurt, Austria  
<https://www.aau.at/en/isys/ics/>  
johann.eder@aau.at

Marco Franceschetti 

Department of Informatics Systems, Universität Klagenfurt, Austria  
<https://www.aau.at/en/isys/ics/>  
marco.franceschetti@aau.at

---

## Abstract

Processes have been successfully introduced for modeling dynamic phenomena in many areas like business, production, health care, etc. Many of these applications require to adequately deal with temporal aspects. Process models need to express temporal durations, temporal constraints like allowed time between events, and deadlines. For checking the correctness of process definitions with temporal constraints, different notions and algorithms have been developed. Schedules for the execution of processes can be computed and proactive time management supports process managers to avoid time failures during the execution of a process. We present an overview of the problems and the requirements for treating time in business processes and the solutions achieved by applying results and techniques of research in temporal representation and reasoning. We reflect where expectations have not yet been met and sketch challenges in temporal representation and reasoning for addressing advanced requirements of the management of business processes.

**2012 ACM Subject Classification** Information systems → Process control systems; Applied computing → Business process management; Information systems → Temporal data

**Keywords and phrases** Business Process management, Temporal constraints, Scheduling, Process Evolution, Probabilistic Controllability

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.3

**Category** Invited Talk

## 1 Introduction

Processes have been successfully introduced for modeling dynamic features in many areas like trade, production, health care, etc. in various forms like workflows [27], extended transactions [30], business processes [15], web-service orchestrations [12], distributed workflows [24], etc. In many of these application areas, temporal aspects are crucial for the correct and admissible execution of processes. This observation led to a substantial body of research to master the plenitude of temporal aspects of process engineering: expressing temporal aspects in process models, formulating different notions of correctness of process models with temporal constraints, checking the temporal correctness of process definitions, computing execution schedules for processes, recognizing and handling temporal exceptions, and supporting process controllers to adhere to temporal constraints at run-time with proactive time management (see [5, 20, 28] for overviews).

Managing business processes means to make decisions, and it is the duty of time management to support the different actors to make these decisions well informed. The primary actors within a BPMS (Business Process Management System) are process participants, process managers and process designer.



© Johann Eder and Marco Franceschetti;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 3; pp. 3:1–3:8

Leibniz International Proceedings in Informatics



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

Support of process design achieved probably most attention. Process designers need formalisms to express temporal aspects of business processes. This includes representing temporal constraints such as deadlines, durations of activities, descriptive and prescriptive constraints. Descriptive constraints allow modeling temporal facts about the environment which is not controllable by the process manager, while prescriptive constraints denote goals, temporal properties which the process controller has to achieve. For an example, the constraint “money transfer lasts between 1 and 4 days” is a prescriptive constraint for banking processes, while it is a descriptive constraint for trading processes, giving the trading partners temporal restrictions but also temporal guarantees. Workflow time patterns [34] collected different patterns and notions for expressing various forms of temporal constraints. Process designers also need tools and techniques to check whether a process definition is feasible, in particular, whether it leads to a violation of temporal constraints.

Process participants need support for selecting items from work-lists. For this decision they need to know when each work item should be finished (internal deadline), and how important work items or process instances are. Information about the consequences of finishing work items late can lead to better decisions. Information about current and upcoming load can help to keep oversight and in particular to make decisions whether to accept additional obligations.

Process managers are responsible for the execution of business processes. They need support for decisions and policies scheduling and dispatching process instances and work-items to participants, Role resolution policies and capacity planning and management. A particular duty of process managers is to deal with exceptions including temporal exceptions and time - failures.

Many of these activities are already supported by current techniques of temporal representation and reasoning. The current state-of-the-art in support for temporal aspects of business process management is outlined in the next section. Then we will analyze some problems reducing broad acceptance of these technologies and sketch some areas where further research is needed.

## **2 State-of-the-Art**

In the last two decades, a number of works on time management have appeared in the BPM community. These works mainly focus either on modeling aspects, or on reasoning aspects. However, despite the numerous efforts, no modeling or reasoning standards, or unified approaches, have been achieved yet.

Only in recent years a notable contribution has emerged under the modeling perspective, with the work on time patterns [34]. It systematically identifies and classifies a number of recurring temporal constraint patterns in BPM. This work has particularly high relevance, since it is based on the rigorous analysis of a vast number of real world processes, taken from different application domains (healthcare, administration, industry, finance). Time patterns are supported by a formal foundation. However, awareness from process designers and managers around time patterns for a uniform approach to time management is still missing.

Is there any promising formal framework, which may be adopted to both encode time patterns, and support temporal reasoning, thus highlighting the benefits of a standardized representation and fostering a uniform approach to process time management?

Various specializations of Temporal Constraint Networks (TCNs) are among the most refined formalisms available for modeling and solving temporal problems. A first formalization can be found in [13], where the Simple Temporal Network (STN) is introduced for modeling

and solving the Simple Temporal Problem, i.e. verifying whether a given set of time-constrained time points admits an assignment of timestamps which fulfills all temporal constraints. To model uncertainties, the STN was extended into STN with Uncertainty (STNU) [35, 45], featuring contingent durations, and into Conditional STN (CSTN) [29, 43], including observed conditions. These networks were then combined into the CSTNU [29], which models unpredictability of both durations and conditions. Further extensions with increased expressiveness have been proposed in recent years, such as those which introduce decision time points to the CSTNU (CSTNUD) [46], or a degree of flexibility for reducing contingent durations to the STNU (STNPSU) [10, 33], or resources (CSTNUR) [11].

Contextually, the traditional notion of satisfiability of a set of temporal constraints is challenged by more advanced definitions of temporal correctness. Strong, weak, history-dependent, and dynamic controllability [9, 16, 44] are regarded as the most noteworthy such properties. They respond to the need to know whether correct schedules are possible, despite uncertainties. Techniques to verify such properties for TCNs have been proposed and improved over the years (e.g., [4]). Constraint propagation is among the most advanced and efficient of these techniques, with the added benefit that it makes implicit temporal knowledge explicit.

In the light of these advancements, in the last decade the BPM community started developing TCN-based approaches to represent the temporal aspect of processes, and to perform temporal reasoning on it, e.g. to verify temporal correctness [8, 17]. These approaches, however, were not met by widespread adoption yet.

### 3 Challenges

As outlined above, significant progress in research was made in the last two decades. Nevertheless, many of these techniques are not yet widespread in applications of BPM, and in some areas current approaches do not take into account recent developments in temporal representation and reasoning. For an example, predictive monitoring of processes [14], which among other issues tries to forecast whether a process will meet its deadline, is still based on satisfiability or consistency rather than on controllability ([3]). Most techniques for predictive monitoring rely nowadays on correlations derived by process mining [42] rather than on analytical temporal reasoning, using neural networks [41] rather than temporal constraint networks [32]. While it might be alright for highly repetitive processes, there is a considerable need for reasoning techniques for processes, with fewer instances, for frequently changing processes and for processes with a high number of variants, and for new processes. All these types of processes have in common that the relevant process logs are frequently too small for advanced process mining techniques.

Major roadblocks for the wider adoption of research results from temporal representation and reasoning seem to include the following:

- the inherent difficulty of some of the developed formalisms which are not popular with process designers
- popular constructs for the definition of processes (advanced control structures) are not supported, in particular loops, iterations, repetitions, and exception handling are not yet supported adequately
- focus on asymptotic complexity of algorithms rather than consideration of actual performance of algorithm for the typical size of real world application scenarios and without distinguishing between design time computations and the much higher performance and scalability requirements at run-time.
- connection of temporal aspects with other aspects of process execution, in particular capacity management, resource constraints, and cost.

Continuing, we will highlight a few of the areas, where considerable research needs are identified from the requirements of process management, which have the potential to further the adoption of temporal reasoning techniques.

### 3.1 Advanced Control Structures

BPMN, the Business Process Modeling Notation [36], for the better or the worse developed as the major notion for describing business processes in practice. Any technique for temporal representation and reasoning for business processes will have to support the control structures available in BPMN. This includes, in particular, loops and other repetitive structures. Unbounded loops (“while loops”) currently are not really supported by temporal reasoning techniques. A process with a *while loop* is neither controllable nor dynamically controllable. We have to develop notions for the correctness of temporally constrained processes definitions which also include loops in an adequate way. Temporal control structures [37] and probabilistic controllability are candidate approaches.

BPMN features a plenitude of control structures and modelling notations. There are several research endeavours under way (e.g. [38]) to extend formalisms for temporal reasoning to include structures found in BPMN.

### 3.2 Temporal Data

Both data of type time (e.g. age) and addressing the need to know about the history of data (e.g. price of item when the order was sent, rather the actual process)

The data dimension is one of the major aspects of business processes requiring definitions of data flows and the relationships between data and decisions about the control flow, the production and usage of data as well as the formulation of execution constraints based on data [40] and data aspects in general are quite well developed. Comparatively little attention was paid to data of type time or timestamp, which can combine data constraints with temporal constraints and offer new possibilities for dealing with temporal aspects [25].

In addition, as processes are usually long-running, data might change over time, leading to the necessity of dealing with different versions of data, and managing these data in the sense of temporal data representations [7].

### 3.3 Conflicting Requirements

In practical applications designers and process managers face the problem of conflicting requirements or constraints. There is a need to support detecting, which constraints are (potentially) in conflict and to provide means to resolve conflicts, e.g., by assigning different priorities to constraints such as in [31], or by reasoning over the effects of constraint violations.

Not all requirements are created equal, in particular prescriptive requirements are frequently derived through negotiations, or are the result of designing service offerings for particular markets. Designers need tools and techniques to reason about which constraints are acceptable without losing controllability and for computing trade-offs between different constraints, such as in [26] for the design phase. Similar procedures are also needed for the run-time support for process managers to make decisions when reacting to exceptions and escalations.



### 3.4 Probabilistic Controllability

Currently, the most elaborated notion of correctness is the controllability, resp. dynamic controllability of process definitions [6]. These notions, however, have the disadvantage that they are binary properties, i.e. a process is either controllable or not. In addition, checking controllability relies only on the minimum and maximum duration of activities. We do not represent whether the maximum, resp. minimum duration is a rare exception.

In many practical applications, the strict notion of controllability is too strong, as frequently some risk is taken, when the process is started, that, e.g., it will violate temporal constraints. However, to rationally reflect on this risk, it is important to know how likely such a time failure is.

For many practical applications it is therefore desirable to extend this notion to capture more information. We envision to extend process model by including a distribution function for each (contingent) activity, representing the probability that an activity requires a certain amount of duration. And we extend the notion of controllability to express the probability that a temporal constraint is violated. Earlier approaches focused primarily on the risk of a deadline violation ([21, 22], but it is necessary to extend this to all temporal constraints.

A definition of (dynamic) probabilistic controllability should express whether there is a (dynamic) execution strategy which allows the execution of a process, such that the probability of violating any temporal constraint is below a given threshold.

Probabilistic controllability, by the way, is also a promising approach for overcoming the problem of (dynamic) controllability of processes with loops.

### 3.5 Process Evolution

Temporal representation of process definitions and their evolution creating temporal variants and versions are necessary. Process logs might be more difficult to treat with process mining procedures, if the underlying process definitions, or the process environment, or the context, resp. the environment of the process execution is changing [18]. Some of these problems have been recognized, e.g., in [1]. The adequate representation both of evolving processes models, the relationship between log entries and these temporal process models, as well as the time-related representation of process mining results, still need research.

Such representations are also needed for correct checking of the compliance of process instances to the evolving process models and constraints [28]. One of the difficulties is here that the evolution of processes frequently leads to a manifold of hybrid process instances which are partly conform to the old process definitions and partly accord to the new one. Temporal reasoning is also necessary to check the correctness of the evolution steps and the transformation regulations of process evolution steps [39]. Long running processes might even be affected by several succeeding schema evolutions. And it is important to recognize that processes are constantly in a need of adaptation, optimization, and further development [2], and therefore require adequate support of the continuous progress.

### 3.6 Temporal Aspects in Combination of Other Dimensions

Temporal aspects and temporal constraints can frequently not be considered in isolation, but they have to be treated in combination with other types of constraints. Recently, a combined representation and reasoning of temporal constraints with resource constraints was reported (e.g. [11]). With the consideration of resources, the consideration of their capacity is a next step. This also leads to the management of the capacity and the agenda of workflow participants, such as in [23].

Moreover, we have to see that mutual influences of durations and cost, which have to be considered together, lead to a notion of *affordable controllability*, meaning that there is an execution strategy within the budgetary limits avoiding any temporal constraint.

Another important dimension for planning process executions is the representation of risks that process activities cannot be executed as planned, or that assumptions about the environment do not hold [19] and the consequences of such risks for constraint satisfiability.

## 4 Conclusions

Representing temporal constraints and reasoning about the temporal properties of business processes, steering their execution and supporting their management, have come a long way since the first approaches to deal with temporal aspects of workflow system more than 2 decades ago. And the TIME community contributed many valuable results - mainly notions for the representation of temporal aspects, and formalisms and algorithms for temporal reasoning. Yet, still there are many requirements of Business Process Management which are not supported in a satisfactory way. It was the aim of this presentation to highlight some of the research needs and some ongoing research efforts, to address the challenges of a better support of temporal aspects in Business Process Management.

---

## References

- 1 RP Jagadeesh Chandra Bose, Wil MP van der Aalst, Indrè Žliobaitė, and Mykola Pechenizkiy. Handling concept drift in process mining. In *International Conference on Advanced Information Systems Engineering*, pages 391–405. Springer, 2011.
- 2 Ruth Breu et al. Towards living inter-organizational processes. In *2013 IEEE 15th Conference on Business Informatics*, pages 363–366. IEEE, 2013.
- 3 Cristina Cabanillas, Claudio Di Ciccio, Jan Mendling, and Anne Baumgrass. Predictive task monitoring for business processes. In *International Conference on Business Process Management*, pages 424–432. Springer, 2014.
- 4 Massimo Cairo and Romeo Rizzi. Dynamic controllability of simple temporal networks with uncertainty: Simple rules and fast real-time execution. *Theor. Comput. Sci.*, 797:2–16, 2019. doi:10.1016/j.tcs.2018.11.005.
- 5 Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. The temporal perspective in business process modeling: a survey and research challenges. *Service Oriented Computing and Applications*, 9(1):75–85, 2015.
- 6 Carlo Combi, Luke Hunsberger, and Roberto Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In Joaquim Filipe and Ana Fred, editors, *Agents and Artificial Intelligence*, volume 449 of *Communications in Computer and Information Science*, pages 314–331. Springer Berlin Heidelberg, 2014.
- 7 Carlo Combi and Angelo Montanari. Data models with multiple temporal dimensions: Completing the picture. In *International Conference on Advanced Information Systems Engineering*, pages 187–202. Springer, 2001.
- 8 Carlo Combi and Roberto Posenato. Controllability in temporal conceptual workflow schemata. In *Business Process Management*, pages 64–79. Springer, 2009.
- 9 Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In *Temporal Representation and Reasoning (TIME)*, 2010 17th International Symposium on, pages 129–136. IEEE, 2010.
- 10 Carlo Combi and Roberto Posenato. Extending conditional simple temporal networks with partially shrinkable uncertainty. In Natasha Alechina, Kjetil Nørkvåg, and Wojciech Penczek, editors, *25th International Symposium on Temporal Representation and Reasoning, TIME*

- 2018, Warsaw, Poland, October 15-17, 2018, volume 120 of *LIPICs*, pages 9:1–9:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.TIME.2018.9.
- 11 Carlo Combi, Roberto Posenato, Luca Viganò, and Matteo Zavatteri. Conditional simple temporal networks with uncertainty and resources. *Journal of Artificial Intelligence Research*, 64:931–985, 2019.
  - 12 Florian Daniel and Barbara Pernici. Insights into web service orchestration and choreography. *International Journal of E-Business Research (IJEER)*, 2(1):58–77, 2006.
  - 13 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.
  - 14 Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, and Fredrik Milani. Predictive process monitoring methods: Which one suits me best? In *International Conference on Business Process Management*, pages 462–479. Springer, 2018.
  - 15 Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A Reijers, et al. *Fundamentals of business process management*, volume 1. Springer, 2013.
  - 16 Johann Eder. Computing history-dependent schedules for processes with temporal constraints. In *International Conference on Future Data and Security Engineering*, pages 145–164. Springer, 2019.
  - 17 Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 40–47. ACM, 2019.
  - 18 Johann Eder and Christian Koncilia. Modelling changes in ontologies. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 662–673. Springer, 2004.
  - 19 Johann Eder and Walter Liebhart. Workflow transactions. In *Workflow handbook 1997*, pages 195–202. John Wiley & Sons, Inc., 1997.
  - 20 Johann Eder, Euthimios Panagos, and Michael Rabinovich. Workflow time management revisited. In *Seminal Contributions to Information Systems Engineering*. Springer, 2013.
  - 21 Johann Eder and Horst Pichler. Duration histograms for workflow systems. In *Engineering Information Systems in the Internet Context*, pages 239–253. Springer, 2002.
  - 22 Johann Eder and Horst Pichler. Probabilistic calculation of execution intervals for workflows. In *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 183–185. IEEE, 2005.
  - 23 Johann Eder, Horst Pichler, Wolfgang Gruber, and Michael Ninaus. Personal schedules for workflow systems. In *International Conference on Business Process Management*, pages 216–231. Springer, 2003.
  - 24 Sérgio Esteves and Luís Veiga. Waas: Workflow-as-a-service for the cloud with scheduling of continuous and data-intensive workflows. *The Computer Journal*, 59(3):371–383, 2016.
  - 25 Marco Franceschetti and Johann Eder. Checking temporal service level agreements for web service compositions with temporal parameters. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 443–445. IEEE, 2019.
  - 26 Marco Franceschetti and Johann Eder. Negotiating temporal commitments in cross-organizational business processes. In *27th International Symposium on Temporal Representation and Reasoning (TIME 2020)*, 2020.
  - 27 Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
  - 28 Mustafa Hashmi, Guido Governatori, Ho-Pun Lam, and Moe Thandar Wynn. Are we done with business process compliance: state of the art and challenges ahead. *Knowl. Inf. Syst.*, 57(1):79–133, 2018. doi:10.1007/s10115-017-1142-1.
  - 29 Luke Hunsberger, Roberto Posenato, and Carlo Combi. The dynamic controllability of conditional stns with uncertainty. *CoRR*, abs/1212.2005, 2012. arXiv:1212.2005.

- 30 Sushil Jajodia and Larry Kerschberg. *Advanced transaction models and architectures*. Springer Science & Business Media, 2012.
- 31 Akhil Kumar, Sharat R. Sabbella, and Russell R. Barton. Managing controlled violation of temporal process constraints. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, volume 9253 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2015. doi:10.1007/978-3-319-23063-4\_20.
- 32 Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, pages 39–56. Springer, 2013.
- 33 Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Simple temporal networks with partially shrinkable uncertainty. In Stéphane Loiseau, Joaquim Filipe, Béatrice Duval, and H. Jaap van den Herik, editors, *ICAART 2015 - Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 2, Lisbon, Portugal, 10-12 January, 2015*, pages 370–381. SciTePress, 2015.
- 34 Andreas Lanz, Manfred Reichert, and Barbara Weber. Process time patterns: A formal foundation. *Information Systems*, 57:38–68, 2016.
- 35 Paul H. Morris and Nicola Muscettola. Managing temporal uncertainty through waypoint controllability. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1253–1258. Morgan Kaufmann, 1999. URL: <http://ijcai.org/Proceedings/99-2/Papers/083.pdf>.
- 36 ObjectManagementGroup. Business Process Model and Notation (BPMN), Version 2.0. <http://www.omg.org/spec/BPMN/2.0>, 2011.
- 37 Horst Pichler, Johann Eder, and Margareta Ciglic. Modelling processes with time-dependent control structures. In *Int. Conference on Conceptual Modeling*, pages 50–58. Springer, 2017.
- 38 Roberto Posenato, Francesca Zerbato, and Carlo Combi. Managing decision tasks and events in time-aware business process models. In *International Conference on Business Process Management*, pages 102–118. Springer, 2018.
- 39 Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness criteria for dynamic changes in workflow systems—a survey. *Data & Knowledge Engineering*, 50(1):9–34, 2004.
- 40 Nick Russell, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. Workflow data patterns: Identification, representation and tool support. In *International Conference on Conceptual Modeling*, pages 353–368. Springer, 2005.
- 41 Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with lstm neural networks. In *International Conference on Advanced Information Systems Engineering*, pages 477–492. Springer, 2017.
- 42 Irene Teinemaa, Marlon Dumas, Fabrizio Maria Maggi, and Chiara Di Francescomarino. Predictive business process monitoring with structured and unstructured data. In *International Conference on Business Process Management*, pages 401–417. Springer, 2016.
- 43 Ioannis Tsamardinos, Thierry Vidal, and Martha E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints An Int. J.*, 8(4):365–388, 2003. doi:10.1023/A:1025894003623.
- 44 Thierry Vidal. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999.
- 45 Thierry Vidal and Hélène Fargier. Contingent durations in temporal cps: From consistency to controllabilities. In *4th International Workshop on Temporal Representation and Reasoning, TIME '97, Daytona Beach, Florida, USA, May 10-11, 1997*, pages 78–85. IEEE Computer Society, 1997. doi:10.1109/TIME.1997.600786.
- 46 Matteo Zavatteri and Luca Viganò. Conditional simple temporal networks with uncertainty and decisions. *Theoretical Computer Science*, 797:77–101, 2019.


# Negotiating Temporal Commitments in Cross-Organizational Business Processes

Marco Franceschetti 

Department of Informatics Systems, Universität Klagenfurt, Austria

<https://www.aau.at/en/isys/ics/>

marco.franceschetti@aau.at

Johann Eder 

Department of Informatics Systems, Universität Klagenfurt, Austria

<https://www.aau.at/en/isys/ics/>

johann.eder@aau.at

---

## Abstract

Cross-organizational business processes emerge from the cooperation of intra-organizational business processes through exchange of messages. The involved parties agree on communication protocols, which contain in particular temporal constraints: as obligations on one hand, and as guarantees on the other hand. These constraints form also requirements for the design of the hidden implementation of the processes and are the basis for control decisions for each party. We present a comprehensive methodology for modeling the temporal aspects of cross-organizational business processes, checking dynamic controllability of such processes, and supporting the negotiation of temporal commitments. We do so by computing the consequences of temporal constraints in choreographies, and by computing the weakest preconditions for the dynamic controllability of a participating process.

**2012 ACM Subject Classification** Applied computing → Cross-organizational business processes

**Keywords and phrases** Cross-organizational processes, Temporal parameters, Range negotiation

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.4

## 1 Introduction

Cross-organizational business processes [13, 17, 20, 30] emerge from an ensemble of local processes communicating through process views [6]. Process views abstract from the internals of the process implementation, and describe the communication interfaces of the processes. The design of choreographies can thus be solely based on these process views. Such an architecture achieves the desired low coupling between the processes, and facilitates keeping business secrecy of process internals as far as possible. A widely adopted approach models cross-organizational processes by a protocol for exchanging messages governed by Service Level Agreements [28]. Formulating, checking, and enforcing temporal properties - characteristics, obligations, commitments, and objectives - for cross-organizational p2p-processes requires distributed algorithms for checking and/or computing temporal properties [14].

Temporal parameters [11] encode events, and can be exchanged via messages between the partners of a collaboration to communicate event timestamps, without having to expose process internals in the process views. The exchange of temporal parameters has been shown to enable an effective way for expressing cross-organizational temporal constraints, while preserving business secrets encoded in the local processes. Thus, temporal parameters foster lean interfaces caring for loose coupling of the involved processes [16].

Temporal constraints for the execution of processes can be formulated as relations between time-points of message exchanges and/or ranges for the temporal parameters. Such constraints (e.g. the time interval for the reaction to the receipt of a message) can be descriptive for one party (“when can I expect an answer from the other party”) and prescriptive for the other party (“when do I have to send an answer?”). These temporal constraints are part



© Marco Franceschetti and Johann Eder;

licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 4; pp. 4:1–4:15

Leibniz International Proceedings in Informatics



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

of the negotiation of the communication protocols (choreography), respectively of Service Level Agreements, which are part of the specification for cross-organizational processes. For process designers it is, therefore, of paramount importance to verify at design time, whether at run time it is possible to avoid any violation of these temporal obligations in the contracts, independent of uncertainties, when other parties are sending messages within their agreed time frames. Technically, this means that process designers need to check the dynamic controllability [26] of their processes. Basically, a process is dynamically controllable, if the execution environment is able to dynamically schedule its tasks in response to all foreseeable circumstances, in such a way that all temporal constraints can be met.

These computations are, however, also a necessary part of the negotiation step for the design of cross-organizational processes. In particular, it is necessary, for the negotiators, to calculate the consequences of any temporal obligation, and to compute which uncertainties (e.g. duration intervals) in the constraints of other parties are acceptable. In a nutshell, there is a need to calculate the trade-offs between temporal constraints to support the negotiation of temporal commitments.

In [11] we have shown how to check, at design time, whether a single process with temporal parameters is dynamically controllable. In [15] we have shown how to check, whether a given set of constraints on the temporal parameters ensures dynamic controllability of a service composition. For cross-organizational processes this problem is more complex, due to the distributed design and execution, and the potentially different requirements of each partner. So now we ask: *how to achieve contracts between the parties of a cross-organizational business process, with adequate restrictions on temporal parameters, admitting schedules free from time failures?*

The main contributions of this paper are:

- A cross-organizational process model based on process views and message exchanges, featuring temporal parameters for temporal constraints.
- A procedure for checking the dynamic controllability of cross-organizational processes with temporal parameters at design time.
- An algorithm for negotiating restrictions for temporal parameters of a cross-organizational process in a fully distributed way, such that these restrictions yield dynamic controllability.

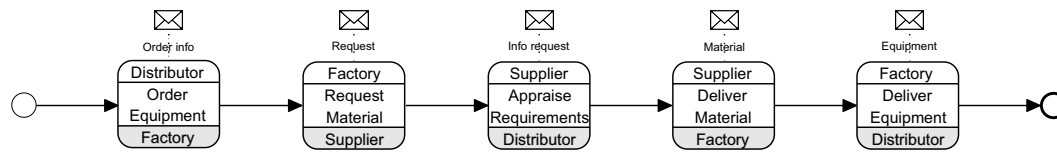
The remainder of this paper is structured as follows: in Section 2 we provide a motivating example. Sect. 3 introduces the process model for cross-organizational processes. In Sect. 4 we show a procedure for negotiating temporal parameter restrictions with dynamic controllability guarantees, and we evaluate the procedure under correctness and complexity dimensions. Sect. 5 (Related work) and Sect. 6 (Conclusions) conclude the paper.

## **2** Motivating Example

Consider the BPMN (Business Process Modeling Notation) choreography diagram depicted in Fig. 1, which models a simplified procurement process. Three organizations are involved: a distributor  $D$  of medical equipment, a factory  $F$  manufacturing surgical masks, and a supplier  $S$  of raw materials. The process starts at  $D$  with the request to  $F$  for a batch of masks. For simplicity, let us focus on the interactions between  $D$  and  $F$  only.

The process model includes three cross-organizational temporal constraints, which restrict the times of events in the local processes of  $D$  and  $F$ . These constraints are realized through the exchange of messages between  $D$  and  $F$ , which include temporal data specifying requirements.





■ **Figure 1** BPMN choreography diagram for a cross-organizational process.

A first constraint  $c1$  is internal to the process of  $D$ , and it requires that the masks are produced between 6 and 10 days after the order is placed. A second constraint  $c2$  in the process of  $F$ , however, requires the production to take place 9 to 11 days after the order is placed. A third constraint  $c3$  requires  $F$  to complete the delivery of the masks at most 2 days after they have been produced.

Here  $c1$ ,  $c2$  and  $c3$  are cross-organizational constraints, since they restrict events in processes of different organizations, to occur within specified bounds with respect to each other. These constraints can be enforced by contracts, which specify an earliest or latest date for completion of a task. However, to guarantee the fulfilment of the constraints, a negotiation needs to take place to ensure an agreement on the requirements. For example,  $c1$  and  $c2$  here are conflicting, since they require different time intervals for the same event, e.g.,  $D$  would not accept that  $F$  produces the masks 11 days after the order is placed.

On the other hand, temporal constraints may as well influence each other. For instance, if  $F$  is convinced by  $D$  to produce the masks earlier than intended, the need to ship them within 2 days (from  $c3$ ) may be unattended because shipment is already scheduled for a certain date.

So to make sure that the local processes can coordinate and execute meeting all constraints,  $D$  and  $F$  need to find an agreement on what they can expect from each other, as well as on what they can offer. Our aim is to provide the partners of a cross-organizational process a protocol for negotiating temporal parameter restrictions, which lead to agreed constraints with no risk of time failure.

### 3 Cross-Organizational Processes

Cross-organizational processes emerge from local processes interacting by exchanging messages according to some choreography. Here we do not focus on the definition of process views and the alignment of process views with choreographies or global processes, but we assume that a choreography or message exchange protocol has been defined already. We focus on the definition of temporal constraints for the process model. Temporal constraints may depend on each other and there may be trade-offs between different temporal constraints.

We consider a cross-organizational process from the viewpoint of one party: a process model in which some of the steps are used for receiving and sending messages. Temporal constraints from such a viewpoint are local constraints and choreography constraints. Choreography constraints can only be formulated using the elements (events, temporal variables) which are known by both parties, i.e. elements of the process view. Local constraints can also include local events and local temporal variables [16].

In such a cross-organizational process, the local processes need to agree on the choreography constraints such that the cross-organizational process can be executed without time failures for all foreseeable situations.

### 3.1 Local Process Model

We consider here a standard (local) process model, such as the model we introduced in previous work [16], to which we refer for details on the architecture and assumptions.

Basically, the model is a standard process model derived from workflow nets, consisting of steps connected by precedence relationships. It features the most commonly used control flow elements as identified in [33]: start and end events, tasks, sequence, exclusive split, parallel split, and their corresponding joins. In addition, it has 2 specialties: (i) *communication steps*: some of the steps send or receive messages, and (ii) *temporal parameters*: some of the parameters in the message payload may encode temporal information. In [16] we argued in detail about the use of temporal parameters for exchanging temporal information, and on the use of temporal parameters for formulating cross-organizational temporal constraints. According to the state-of-the-art and to avoid design flaws [8], we consider here acyclic block-structured processes, assume that variables are available when they ought to be sent, and that no deadlocks occur. Discussions about these assumptions for well-formedness are out of scope for this paper.

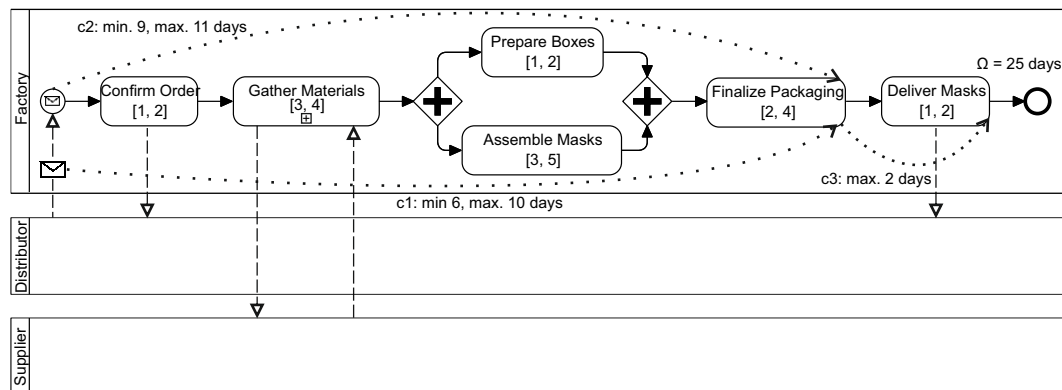
We formally define the local process model as follows:

► **Definition 1** (Local Process Model). *A local process model  $P$  is a tuple  $(proc\_id, N, E, V, C, \Omega)$ , where:*

- *$proc\_id$  is a unique process id.*
- *$N$  is a set of nodes. A node  $n$  has type  $n.type \in \{start, activity, xor - split, par - split, xor - join, par - join, send, receive, end\}$ , and start and end events  $n.s$ , resp.  $n.e$ . For each node  $n$ :*
  - *$n.I$  is the set of input variables;*
  - *$n.O$  is the set of output variables;*
  - *$n.t$  (to) is the id of the receiver for send nodes;*
  - *$n.f$  (from) is the id of the sender for receive nodes.*
- *$E \subseteq N \times N$  is a set of edges defining precedence constraints.*
- *$V$  is a set of temporal variables, partitioned in disjoint sets  $V^I, V^O, N^e$ :*
  - *$V^I$  is the set of input parameters of  $P$ :  
 $V^I = \bigcup_{n \in N} \{v \in n.I \mid n.type = receive\}$ ;*
  - *$V^O$  is the set of output parameters of  $P$ :  
 $V^O = \bigcup_{n \in N} \{v \in n.O \mid n.type = send\}$ ;*
  - *$N^e$  is the set of variables for start and end events of nodes:  $N^e = \bigcup_{n \in N} \{n.s, n.e\}$ .*
- *$C$  is a set of temporal constraints consisting of*
  - *duration constraints for each  $n \in N$ :  $d(n, d_{min}, d_{max})$ , with  $n.type = activity$ , where  $d_{min}, d_{max} \in \mathbb{N}$  with  $d_{min} \leq d_{max}$ ; for all other nodes,  $d_{min} = d_{max} = 0$ ;*
  - *range constraints for temporal variables  $v \in V^I \cup V^O$ :  $r(v, v_{min}, v_{max})$ , where  $v_{min}, v_{max} \in \mathbb{N}$  with  $v_{min} \leq v_{max}$ ;*
  - *upper-bound constraints:  $ubc(a, b, \delta)$ , where  $a, b \in V, \delta \in \mathbb{N}$ , requiring that  $b \leq a + \delta$ ;*
  - *lower-bound constraints:  $lbc(a, b, \delta)$ , where  $a, b \in V, \delta \in \mathbb{N}$ , requiring that  $b \geq a + \delta$ .*
- *$\Omega$  is the maximum process duration.*

Given the definition of the local process model, we can now illustrate in detail its temporal aspect, and define its temporal semantics.





■ **Figure 2** BPMN collaboration diagram for the process of Fig. 1 from the viewpoint of the Factory.

### 3.2 Temporal Aspect of the Local Process Model

The temporal aspect of the process model includes time points for events, durations of nodes, temporal parameters, and temporal constraints as relations between timepoints and/or durations. Each process node  $n$  is associated with two events,  $n.s$  and  $n.e$ , representing the start and end of the respective node.

Duration constraints specify minimum and maximum bounds for the actual durations of activities. Without loss of generality [12] we consider all activities as contingent, i.e. their actual duration cannot be controlled, but only be observed at run time, but it can be assumed to be within the interval specified by the minimum and maximum duration.

Temporal parameters are data elements containing a timestamp value. They can be sent to some other process via a communication channel. Input parameters are received from another process by a receiving node; output parameters are sent to another process by a sending node.

Temporal constraints restrict the points in time when events may occur. By imposing temporal constraints between parameters and events, it is possible to constrain the occurrence of local events, or events at other local processes.

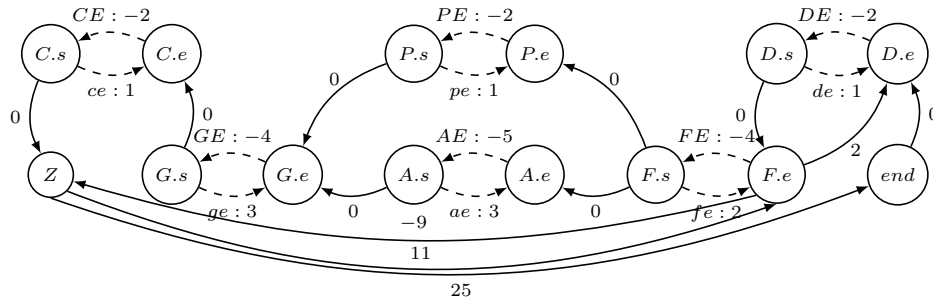
A special temporal constraint is defined by  $\Omega$ , which requires that the overall process duration is less than the value specified by  $\Omega$ .

Constraints can only be formulated between the elements in the process model. Hence constraints between events in different processes can only be formulated, if information about event is sent to the other process in form of temporal parameters.

In Fig. 2 we show the BPMN collaboration diagram for the choreography diagram from Fig. 1, enriched with an explicit representation of the temporal constraints as dotted arrows. We stress the problem of not being able to access the internals of other local processes, by showing the diagram from the viewpoint of the factory  $F$ , with the other processes modeled as black boxes. In the example,  $D$  must include the date she requires for mask production, as part of the message sent with the first choreography task (see Fig. 1). In its local process definition,  $F$  would use such a date, which is an input parameter, in a constraint binding the time for completing mask production.

### 3.3 Negotiation of Temporal Commitments

Parties in a cross-organizational process have to agree on temporal commitments, i.e. on constraints defined on admissible timepoints of communication events (sending and receiving messages) and on the ranges for the temporal parameters. The goal is to agree on a set



■ **Figure 3** STNU derived from the local process of *F* in Fig. 1.

of constraints, such that time failures can be avoided for all possible process instances, i.e. if all local processes with temporal parameters are dynamically controllable, then the cross-organizational process is dynamically controllable.

For the receiving process, a range of an input parameter is a guarantee that the values of the parameter will be within the range. On the other hand, it is an obligation for the sending process to send a value within the agreed range. If the range of an input variable is wider, then the uncertainty for the receiving process is higher, and it is more difficult to avoid time failures. However, if the range of an output parameter is smaller, it is more difficult to send a valid parameter. Every parameter is both output for some process and input for some other process.

Moreover, the ranges of the input parameters influence the possible ranges of the output parameters, and dynamic controllability also depends on these ranges. Therefore, these ranges have to be negotiated between the different parties. Additionally, there might be inter-dependencies between parameters, leading to trade-offs between the ranges, i.e. a broader range for one parameter may reduce the ranges of other parameters.

In negotiating parameter restrictions, each party has to understand the dependencies and trade-offs between different ranges and other constraints. Since these dependencies are distributed over all the participating processes, these dependencies and their consequences can only be calculated with a distributed procedure.

For the negotiations, the following computations are necessary:

1. which ranges of the input parameters can be accepted,
2. which ranges of the output parameters can be guaranteed,
3. which constraints have to hold between parameters.

For the input parameters, we aim to compute the widest possible ranges; for the output parameters, the most narrow ranges (cf. contra-variant subtyping [5]). In addition, the computations need to determine the constraints between parameters, representing the above mentioned trade-offs.

For checking dynamic controllability, and for computing admissible restrictions for temporal parameters, we map process models into temporal constraint networks, and resort to the temporal reasoning capabilities of temporal constraint networks. In particular, we currently consider Simple Temporal Networks with Uncertainty (STNUs) as the formal apparatus, instead of more expressive networks (e.g., CSTNUs): this may lead to stricter results, but the reasoning algorithms have lower complexity. In Fig. 3 we show the STNU equivalent to the process of *F* from Fig. 2, with abbreviated activity names for better readability. For details on STNUs, and how to map process models into STNUs, we refer to Appendix A.

## 4 Constraint Negotiation

Process models and the problem of negotiating constraints are mapped into terms of temporal constraint networks. Thus, here we present a framework for the negotiation of temporal constraints, which is based on temporal networks as the formal apparatus for inferring temporal constraints. For solving the negotiation problem outlined above, we developed new inference mechanisms for STNUs. Finally, we map the solution back into process model terms.

In the following we consider only STNUs which are verified to be dynamically controllable.

### 4.1 Computing Constraints for Temporal Parameters

Recently, an efficient method for checking dynamic controllability of STNUs has been proposed in [3]. It applies three rules for inferring implicit constraints, and checks for contradictions in form of negative cycles in the STNU. For dynamic controllability, however, the possible values of a node might depend on all the nodes which have smaller values. Thus, the value of a parameter node might depend on the observed duration of contingent activities. This is unacceptable for parameter nodes, whose timestamps have to be fixed when they are communicated. We even require that possible ranges for these parameter nodes are defined at design time as part of the negotiations outlined above. We address this requirement by proposing three rules, which infer constraints on nodes such that they are independent of observed durations of contingent activities.

Another challenge is that there is no central STNU, but there is a set of communicating STNUs and an infrastructure for communicating constraints between these STNUs in the design phase. All possible algorithms have to take into account that local STNUs do not expose their internal structure and internal constraints, and only communicate the constraints on parameters to their peers.

We solve these problems by first introducing basic rules for inferring constraints, which make parameter nodes independent of observed contingent durations. Then we propose a procedure to compute restrictions on input and output parameters for a local STNU. Finally, we discuss a framework for the communication of constraints to come up with an agreement on parameter restrictions, which are compatible with the dynamic controllability of each participating STNU.

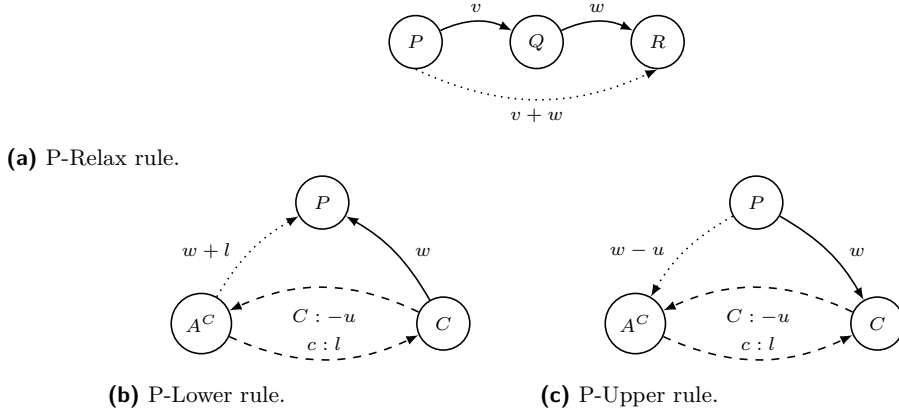
### 4.2 Basic Inference Rules

We propose three basic rules (*P-Relax*, *P-Upper*, and *P-Lower*, shown in Fig. 4) for the deduction of constraints in form of non-contingent STNU edges implicitly contained in an STNU, such that the derived constraints for parameters are independent of any contingent duration.

**P-Relax:** This rule is applied when three time points are connected by a sequence of two consecutive non-contingent edges with weights  $v$  and  $w$ , respectively. It introduces a new non-contingent edge with weight  $v + w$  between the time point origin of the first edge and the time point destination of the second edge.

► **Lemma 2.** *P-Relax derives the broadest non-contingent constraint between time point  $P$  and time point  $R$ .*

**Proof.** Since *P-Relax* is defined as *Relax* in the RUL system, for the proof we refer to [3]. ◀



■ **Figure 4** Constraint inference rules. The derived constraints are dotted.

P-Lower: This rule is applied when a time point  $P$  is the target of an upper-bound constraint with origin the contingent time point of a contingent link. Given the minimum contingent duration  $l$ , and the upper-bound constraint bound  $w$ , the rule derives an upper-bound constraint from the activation time point of the contingent link to  $P$ , with bound  $w + l$ .

► **Lemma 3.** *P-Lower derives the broadest non-contingent constraint  $c = (P \leq A^C + w + l)$  between time point  $A^C$  and time point  $P$  in a dynamically controllable STNU  $S$ , which makes the constraint  $(P \leq C + w)$  between  $C$  and  $P$  redundant.*

**Proof.** Redundancy: the introduction of  $c$  in  $S$  requires to satisfy  $P \leq A^C + w + l$ .  $\forall d : l \leq d \leq u, A^C + d \leq C$ . So  $P \leq A^C + w + l \implies P \leq C + w$ .

Broadness: suppose that the derived constraint between  $A^C$  and  $P$  is less strict, e.g.  $P \leq A^C + w + l + 1$ . Then in the scenario in which  $P = A^C + w + l + 1$  and the contingent link takes its minimum duration ( $C = A^C + l$ ), the requirement that  $P \leq C + w$  from the original constraint is violated by 1 time unit. Therefore  $P \leq A^C + w + l$  is the most lenient constraint which can be set between  $A^C$  and  $P$  without contradicting the one between  $C$  and  $P$ . ◀

P-Upper: This rule is applied when a time point  $P$  is the origin of an upper-bound constraint with target the contingent time point of a contingent link. Given the maximum contingent duration  $u$ , and the upper-bound constraint bound  $v$ , the rule derives an upper-bound constraint from  $P$  to the activation time point of the contingent link, with bound  $w - u$ .

► **Lemma 4.** *P-Upper derives the broadest non-contingent constraint  $c = (A^C \leq P + w - u)$  between time point  $P$  and time point  $A^C$  in a dynamically controllable STNU  $S$ , which makes the constraint  $(C \leq P + w)$  between  $P$  and  $C$  redundant.*

**Proof.** Redundancy: the introduction of  $c$  in  $S$  requires to satisfy  $A^C \leq P + w - u$ , i.e.  $A^C + u \leq P + w$ .  $\forall d : l \leq d \leq u, A^C + d \leq C$ . So  $A^C \leq P + w - u \implies C \leq P + w$ .

Broadness: suppose that the derived constraint is less strict, e.g.,  $A^C \leq P + w - u + 1$ . Then the scenario in which  $P = A^C - w + u - 1$  and the contingent link takes its maximum duration ( $C = A^C + u$ ), the original constraint  $C \leq P + w$  is violated by 1 time unit. Therefore  $A^C \leq P + w - u$  is the most lenient constraint between  $P$  and  $A^C$  without contradicting the constraint between  $P$  and  $C$ . ◀

With these basic inference rules we can now define a procedure for inferring parameter ranges and restrictions through iterated application of these rules.

### 4.3 Local Inference of Parameter Restrictions

The inference procedure iteratively applies the basic inference rules to a dynamically controllable STNU  $S$  until quiescence (i.e. no new edge can be derived). We call the resulting STNU the *closure* of  $S$ .

► **Lemma 5.** *The inference procedure is sound and complete.*

**Proof.** (1) The procedure always terminates, as each rule only adds constraints, no rule has the absence of a constraint as condition, increments in constraints are always multiples of 1 chronon, and there is an overall deadline. (2) The correctness of a derived constraint is an immediate consequence of the correctness of each of the three rules. (3) As each of the rules derives a constraint which is at least as strict as a constraint derived by the rules in [3] but is the weakest constraint satisfying the requirements (Lemma 1-3) completeness follows from the completeness of the rules in [3]. ◀

The closure can now be analyzed to extract parameter restrictions. We are mainly interested in the derived links between a parameter and *zero* and between 2 parameters. For a node  $n$ , a non-contingent edge  $(zero, n, w)$  means that  $n$  is allowed to take at most value  $w$ , for the STNU to be dynamically controllable. A non-contingent edge  $(n, zero, -v)$  means that  $n$  has to take a value greater or equal to  $v$ , for the STNU to be dynamically controllable. We call  $[v, w]$  the range restriction for  $n$ .

► **Lemma 6.** *Let  $S$  be the closure of the STNU for a process  $P$ . Let  $p \in param(S)$  ( $p$  is a temporal parameter of  $P$ ) such that  $\forall q \in param(S) \exists (p, q, \delta), (q, p, \delta') \in S$ . Let  $(zero, p, w), (p, zero, -v)$  be edges in  $S$ . Then  $[v, w]$  is the broadest range of values for  $p$  which is compatible with the dynamic controllability of  $S$ .*

**Proof.** Let us assume that  $S$  as above is dynamically controllable. It is easy to see, that fixing  $p$  (by introducing edges  $(zero, p, \bar{p})$  and  $(p, zero, -\bar{p})$ ) to any value  $\bar{p}$  either smaller than  $v$ , or larger than  $w$ , would introduce a negative loop in the STNU, thus making it not dynamically controllable. Now let us assume that  $S$  is dc but there is a value  $v \leq \bar{p} \leq w$  for  $p$ , such that  $S' = S \cup \{(zero, p, \bar{p}), (p, zero, -\bar{p})\}$  is not dc, i.e there is a negative cycle in  $S'$ . Then with the Decomposability theorem [10] we can conclude that the negative cycle was already in  $S$  - which is a contradiction to the assumption that  $S$  is dynamically controllable. ◀

STNU edges derived between any two nodes  $m$  and  $n$ , represent constraints restricting the values for the timestamps of these nodes with respect to each other, which as well need to be fulfilled, in order for the STNU to be dynamically controllable. For instance, a derived edge  $(m, n, 20)$  would mean *the timestamp of  $n$  must be no more than 20 time units after the timestamp of  $m$ .*

► **Lemma 7.** *Let  $S$  be the closure of the STNU for a process  $P$ . Let  $p$  and  $q$  be nodes in  $S$  for temporal parameters. Let  $(p, q, w)$  be an edge in  $S$ . Then  $q \leq p + w$  is the broadest constraint between the timestamps of  $p$  and  $q$  which is compatible with the dynamic controllability of  $S$ .*

**Proof.** It is easy to see, that fixing  $q$  (by introducing edges  $(zero, q, \bar{q})$  and  $(q, zero, -\bar{q})$ ) to any value  $\bar{q}$  violating  $q \leq p + w$ , would introduce a negative loop in  $S$ , thus making it not dynamically controllable.

■ **Listing 1** Compute, distribute, and receive parameter restrictions

```

repeat
  S' := S;
  S' := infer_restrictions(S);
  for (n in N | n.type = receive)
    transmit_restrictions(n);
  end for
  for (n in N | n.type = send)
    S' := S' ∪ receive_restrictions(n);
  end for
until S' = S;
for (n in N | n.type = receive)
  make_contingent(S, n);
end for

```

Now suppose that  $q$  is fixed (by introducing edges as above) to any value  $\bar{q}$  fulfilling  $q \leq p + w$  and all other constraints on  $q$ , and that a negative cycle is in  $S$ . Then the negative cycle must be due to some other configuration of constraints, which is a contradiction since  $S$  is dynamically controllable. ◀

#### 4.4 Communicating and Negotiating Constraints

A local STNU communicates the inferred restrictions to the senders resp. receivers of the parameters, with the aim of deriving a shared set of restrictions leading to a constellation of communicating STNUS each of which is dynamically controllable.

A difficulty for such a procedure are inter-dependencies between parameters. Consider, as an example, a local STNU  $LP_0$ , which receives two input parameters: parameter  $p_1$  is received from a STNU  $LP_1$ , and parameter  $p_2$  from a different STNU  $LP_2$ . Through inference, it is discovered that, for the dynamic controllability of  $LP_0$ , both  $p_1$  and  $p_2$  have to take values in the range  $[15, 20]$ ; and  $p_2$  must be at least equal to  $p_1 - 1$ , and at most equal to  $p_1 + 1$ . So it is not sufficient that  $LP_0$  asks  $LP_1$  and  $LP_2$  to provide parameters with values in the ranges, since, e.g.,  $p_1 = 16$  and  $p_2 = 20$  would be invalid instantiations.

In Listing 1 we provide a general formulation of an algorithm for deriving a shared set of restrictions on parameters. For each local STNU  $S$ , restrictions for its parameters can be derived by applying the procedure for range inference. Then, all inferred restrictions for input parameters are sent to the respective senders. Restrictions to output parameters computed at the receivers are then received and corresponding non-contingent edges are added to the STNU. The procedure repeats until no change occurs at any local STNU. The procedure assumes that all parameter restrictions are shared between all STNUs, and, therefore, each local STNU can observe when a fix-point is reached or a contradiction is derived. Nonetheless, the algorithm does not require exposing internal constraints of local STNUs.

There are several variants for implementing this basic procedure, with different effects and assumptions, e.g.:

1. Global choreography for the cross-organizational process, with shared parameter space: *"everybody knows all parameter constraints"*;
2. A central coordinator, with full access to all parameter restrictions acts as mediator between local processes, and recognizes, when a fix-point is reached;

3. Global hierarchical process without a global view or central coordinator: a partial order of parameter and process dependencies can be constructed;
4. Fully distributed procedure based on distributed cycle detection (e.g. [2, 27]) for addressing dependencies between parameters.

A discussion of each of these variants and a comparison of their advantages and disadvantages is out of the scope of this paper, and we reserve it for future work. Future work also includes the development of algorithms, which are adaptive to the topology of the communication structure.

## 4.5 Interpreting the Results

If in the procedures described any STNU gets to a negative cycle, then the cross-organizational process is not dynamically controllable.

Otherwise, the algorithm in Listing 1 results in a set of dynamically controllable STNUs which are coherent in their constraints on the shared temporal parameters. The restrictions on these parameters can now be interpreted as follows: the constraints on input parameters define for which constellation of parameter values the network is dynamically controllable and are thus requirements sending processes have to fulfill. The restrictions on output parameters are guarantees which the receiving processes are allowed to assume. Constraints between parameters show the trade-offs to be resolved by further negotiations.

## 4.6 Correctness and Complexity of the Procedure

We observe that the correctness of the algorithm in Listing 1 is based (1) on the correctness of the procedure for inferring parameter restrictions called in line 4, and (2) on the fact that introducing contingent edges specifying the ranges for input parameters in line 13 does not violate dynamic controllability. For (1), an informal sketch of proof is based on the inference procedure deriving ranges which are not less restrictive than the ones derived by the more general rules of [3]. Thus, if a closure of the STNU can be computed, the STNU is dynamically controllable. For (2), we give proof to the following Lemma:

► **Lemma 8.** *Let  $P$  be a process; let  $S$  be the STNU for  $P$ . Let  $p_1, \dots, p_n$  be input parameters, and  $r_i = [p_{i_{min}}, p_{i_{max}}]$  be range restrictions for each  $p_i$  derived from the closure of  $S$ . Then  $S \cup_i (zero, p_{i_{min}}, p_{i_{max}}, p_i)$  is dynamically controllable.*

**Proof.** For parameters for which only range restrictions are derived in the closure, the proof directly follows from Lemma 6. For parameters having additional restrictions, the proof follows from Lemma 7, and the requirement that all derived restrictions are enforced by the senders, in whose STNU these restrictions are non-contingent edges. ◀

At the basis of the algorithm in Listing 1 is the repeated application of the procedure for inferring restrictions. From previous results [25], the complexity of STNU constraint propagation algorithms is polynomial in the number of STNU nodes  $O(N^4)$ . Existing figures [11] for the local application of such procedures to STNUs derived from process definitions indicate the practical applicability of the approach at design time. The local execution of the algorithm may invoke several times the inference procedure, if re-computation is necessary. However, each new invocation only restricts previously computed bounds, and the procedure stops in case a negative cycle is found. Thus, the overall complexity of the distributed execution of the algorithm in Listing 1 at each local participant, is given by  $O(N^4)$ , with  $N$  the number of nodes in the global STNU.



## **5** Related Work

A substantial body of research is devoted to time management for business processes: general overviews of works in the area can be found in [7, 9, 14]. Checking whether deadlines and time constraints can be fulfilled in time-constrained process definitions is addressed by early works such as [1, 24], which are based on network analysis, scheduling, or constraint networks. The specific case of cross-organizational processes and service compositions is addressed by works such as [4, 18]; modularized processes are addressed in [21]. However, none of these approaches considers using temporal parameters for expressing temporal properties or temporal requirements; here we have shown how temporal parameters enable the expression of temporal constraints crossing the scope of a single intra-organizational process.

Pro-active monitoring of the compliance of process instances to their process model, which is considered in, e.g., [19, 22], plays an essential role in the management of timed processes in general. Collaborative processes in particular are addressed in [23], which is based on timed automata and model checking techniques. However, all these approaches consider satisfiability rather than dynamic controllability as the notion for temporal correctness.

Here we showed an algorithm for computing constraints on parameters, which is based on inferring knowledge from the temporal aspect of a local process definition, and the communications with other local process definitions, with no visibility of their internals. Alternatively, process mining techniques [29] may be used to derive missing temporal qualities for a model; however, this is only possible if there is a sufficient number of traces available in the process logs. In contrast to such an approach, here we focus on new process definitions, and on a design time check of their temporal properties.

As an implementation for the proposed algorithm, we showed an approach based on mapping process definitions to Simple Temporal Networks with Uncertainty (STNUs) [26]. Considerable research efforts have been devoted in the last decades both to developing different notions of controllability and more expressive network models [21, 31, 32]. Considering the increasing complexity for verifying dynamic controllability of these more refined networks, we regard STNUs as a suitable formalism for representing the temporal dimension of a process model and deriving missing temporal information at design time.

## **6** Conclusions

Temporal parameters proved as a highly adequate means for expressing temporal obligations and guarantees between the participants of a cross-organizational business process. Negotiating constraints on temporal parameters has to respect the need for keeping internals of the participating processes secret, while arriving at a solution, which allows the distributed control of processes in a way that no temporal constraint is violated.

The procedures proposed in this paper are a first attempt to successfully support the negotiation of temporal commitments through the computation of maximum ranges for input parameters and minimum ranges for output parameters in an effective way. These parameter ranges serve as obligations and guarantees of temporal properties for the participants in the cross-organizational business process. Additionally, they build the basis for the distributed and autonomous scheduling of the participating processes, without risking time failures and temporal exceptions.



---

**References**

---

- 1 Claudio Bettini, X.Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002.
- 2 Lubos Brim, Ivana Černá, and Lukás Hejtmánek. Distributed negative cycle detection algorithms. In *Advances in Parallel Computing*, volume 13, pages 297–304. Elsevier, 2004.
- 3 Massimo Cairo and Romeo Rizzi. Dynamic controllability of simple temporal networks with uncertainty: Simple rules and fast real-time execution. *Theor. Comput. Sci.*, 797:2–16, 2019. doi:10.1016/j.tcs.2018.11.005.
- 4 Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *J of Web Semantics*, 1(3):281–308, 2004.
- 5 Giuseppe Castagna. Covariance and contravariance: conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, 1995.
- 6 Issam Chebbi, Schahram Dustdar, and Samir Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data & Knowledge Engineering*, 56(2):139–173, 2006.
- 7 Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. The temporal perspective in business process modeling: a survey and research challenges. *Service Oriented Computing and Applications*, 9(1):75–85, 2015.
- 8 Carlo Combi and Mauro Gambini. Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data. In *OTM Confederated International Conferences*, pages 42–59. Springer, 2009.
- 9 Carlo Combi and Giuseppe Pozzi. Temporal conceptual modelling of workflows. In *Conceptual Modeling-ER 2003*, pages 59–76. Springer Berlin Heidelberg, 2003.
- 10 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.
- 11 Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 40–47. ACM, 2019.
- 12 Johann Eder, Marco Franceschetti, Julius Köpke, and Anja Oberrauner. Expressiveness of temporal constraints for process models. In *International Conference on Conceptual Modeling*, pages 119–133. Springer, 2018.
- 13 Johann Eder, Nico Kerschbaumer, Julius Köpke, Horst Pichler, and Amirreza Tahamtan. View-based interorganizational workflows. In *Proceedings of the 12th International Conference on Computer Systems and Technologies*, pages 1–10. ACM, 2011.
- 14 Johann Eder, Euthimios Panagos, and Michael Rabinovich. Workflow time management revisited. In *Seminal Contributions to Information Systems Engineering*, pages 207–213. Springer Berlin Heidelberg, 2013.
- 15 Marco Franceschetti and Johann Eder. Checking temporal service level agreements for web service compositions with temporal parameters. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 443–445. IEEE, 2019.
- 16 Marco Franceschetti and Johann Eder. Designing decentralized business processes with temporal constraints. In *CAiSE Forum (forthcoming)*, 2020.
- 17 Paul Grefen and Yigal Hoffner. Crossflow-cross-organizational workflow support for virtual organizations. In *Proceedings 9th Int. Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises.*, pages 90–91. IEEE, 1999.
- 18 Nawal Guermouche and Claude Godart. Timed model checking based approach for web services analysis. In *ICWS 2009.*, pages 213–221. IEEE, 2009.
- 19 Mustafa Hashmi, Guido Governatori, Ho-Pun Lam, and Moe Thandar Wynn. Are we done with business process compliance: state of the art and challenges ahead. *Knowledge and Information Systems*, pages 1–55, 2018.
- 20 Julius Köpke, Marco Franceschetti, and Johann Eder. Optimizing data-flow implementations for inter-organizational processes. *Distributed and Parallel Databases*, pages 1–45, 2018.

- 21 Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Controlling time-awareness in modularized processes. In *Enterprise, Business-Process and Information Systems Modeling*, pages 157–172. Springer, 2016.
- 22 Linh Thao Ly, Fabrizio Maria Maggi, Marco Montali, Stefanie Rinderle-Ma, and Wil M.P. van der Aalst. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information systems*, 54:209–234, 2015.
- 23 Sihem Mallek, Nicolas Daclin, Vincent Chapurlat, and Bruno Vallespir. Enabling model checking for collaborative process analysis: from bpmn to ‘network of timed automata’. *Enterprise Information Systems*, 9(3):279–299, 2015.
- 24 Olivera Marjanovic and Maria E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2):157–192, 1999.
- 25 Paul Morris. A structural characterization of temporal dynamic controllability. In *International Conference on Principles and Practice of Constraint Programming*, pages 375–389. Springer, 2006.
- 26 Paul H Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *AAAI*, pages 1193–1198, 2005.
- 27 Gabriele Oliva, Roberto Setola, Luigi Glielmo, and Christoforos N Hadjicostis. Distributed cycle detection and removal. *IEEE Trans. on Control of Network Systems*, 5(1):194–204, 2016.
- 28 Irfan Ul Haq, Altaf Huqqani, and Erich Schikuta. Aggregating hierarchical service level agreements in business value networks. In *International Conference on Business Process Management*, pages 176–192. Springer, 2009.
- 29 Wil M.P. van der Aalst, M.Helen Schonenberg, and Minseok Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.
- 30 Wil MP van der Aalst and Mathias Weske. The p2p approach to interorganizational workflows. In *International Conference on Advanced Information Systems Engineering*, pages 140–156. Springer, 2001.
- 31 Thierry Vidal. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999.
- 32 Matteo Zavatteri and Luca Viganò. Conditional simple temporal networks with uncertainty and decisions. *Theoretical Computer Science*, 797:77–101, 2019.
- 33 Michael Zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Seminal Contributions to Information Systems Engineering*, pages 429–443. Springer, 2013.

## **A** Mapping Process Models into STNUs

Previous works have already shown how to map a process model into an equivalent temporal network, such as the Simple Temporal Networks with Uncertainty (STNU), for expressing the temporal semantics and verifying temporal correctness of the process model [11, 16]. The advantage of mapping to temporal networks is that it is an established formalism with sound and complete procedures for temporal reasoning. To be self-contained, we report here a brief definition of the STNU, and the rules which allow mapping the process model of Def. 1 into a STNU.

### **A.1** STNU

A STNU  $S = (\mathcal{T}, \mathcal{C}, \mathcal{L})$  is a directed weighted graph, in which nodes (set  $\mathcal{T}$ ) represent time points, and edges (sets  $\mathcal{C}, \mathcal{L}$ ) represent temporal constraints between pairs of time points. A special node *zero* ( $Z$ ) marks the reference in time after which all other time points occur. Two types of edges exist: non-contingent (set  $\mathcal{C}$ ), and contingent (set  $\mathcal{L}$ ). Non-contingent

STNU edges between time points  $A$  and  $B$  have the form  $(A, B, \delta)$ : they require to assign  $A$  and  $B$  timestamps, such that  $B \leq A + \delta$  holds. A contingent STNU edge (also called link) between from a time point  $A^C$  (called the activation time point) to a contingent time point  $C$  have the form  $(A^C, l, u, C)$ : they state that the timestamp of  $C$  will be observed to fall between  $A^C + l$  and  $A^C + u$ .

Dynamic controllability of a STNU requires the existence of a dynamic execution strategy, which assigns values to the non-contingent nodes, such that all temporal constraints are met, for all possible assignment of values to the contingent nodes. A frequently adopted approach to check the dynamic controllability of a STNU is based on propagating its constraints. Constraint propagation derives new edges according to a number of propagation rules. Different systems of rules have been proposed over the years, e.g., [3, 26]. A STNU is dynamically controllable if it is not possible to derive any negative loop through constraint propagation.

## A.2 Mapping Rules

Given the process model of Def. 1, we show here how to map it into a STNU. We formulate the mapping in terms of mapping rules as follows:

► **Definition 9** (Mapping to STNU). *Let  $P = (proc\_id, N, E, V, C, \Omega)$  be a process defined as in Def. 1. The STNU  $S = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ , equivalent to  $P$ , is obtained by applying the following rules:*

1. Each  $n \in N$  with  $n.s, n.e \in N^e$  is mapped into corresponding time points  $n.s, n.e \in \mathcal{T}$ ;
2. Each  $v \in V^I \cup V^O$  is mapped into a corresponding time point  $v \in \mathcal{T}$ ;
3. Each  $e = (m, n) \in E$  is mapped into a corresponding non-contingent edge  $(n.s, m.e, 0) \in \mathcal{C}$ ;
4.  $(start.s, zero, 0) \in \mathcal{C}$ , and  $(zero, end.e, \Omega) \in \mathcal{C}$ ;
5. Each duration constraint  $d(n, d_{min}, d_{max}) \in C$  is mapped into a corresponding contingent link  $(n.s, d_{min}, d_{max}, n.e) \in \mathcal{L}$ ;
6. Each range constraint  $r(v, v_{min}, v_{max}) \in C$ , with  $v \in V^I$ , is mapped into a corresponding contingent link  $(zero, v_{min}, v_{max}, v) \in \mathcal{L}$ ;
7. Each range constraint  $r(v, v_{min}, v_{max}) \in C$ , with  $v \in V^O$ , is mapped into corresponding non-contingent edges  $(zero, v, v_{max})$ ,  $(v, zero, -v_{min}) \in \mathcal{C}$ ;
8. Each  $ubc(a, b, \delta) \in C$ , is mapped into a corresponding non-contingent edge  $(a, b, \delta) \in \mathcal{C}$ ;
9. Each  $lbc(a, b, \delta) \in C$ , is mapped into a corresponding non-contingent edge  $(b, a, -\delta) \in \mathcal{C}$ .

To keep the STNU compact and without loss of generality, process nodes with duration 0 can be collapsed into a single STNU node; process control nodes may not be included in the STNU, by linking their predecessors to their successors; similarly, we may make *start* coincide with *zero* (see Fig. 2 and Fig. 3).

We use the mapping rules of Def. 9 to bring the definition of the temporal aspect of a process model into STNU terms for further temporal reasoning, such as checking its dynamic controllability, and deriving missing temporal information.



# The Horn Fragment of Branching Algebra

**Alessandro Bertagnon** 

Department of Engineering, University of Ferrara, Italy  
alessandro.bertagnon@unife.it

**Marco Gavanelli** 

Department of Engineering, University of Ferrara, Italy  
marco.gavanelli@unife.it

**Alessandro Passantino** 

Department of Mathematics and Computer Science, University of Ferrara, Italy  
alessandr.passantino@student.unife.it

**Guido Sciavicco** 

Department of Mathematics and Computer Science, University of Ferrara, Italy  
guido.sciavicco@unife.it

**Stefano Trevisani** 

Department of Mathematics and Computer Science, University of Ferrara, Italy  
stefan.trevisani@student.unife.it

---

## Abstract

Branching Algebra is the natural branching-time generalization of Allen's Interval Algebra. As in the linear case, the consistency problem for Branching Algebra is NP-hard. Being relatively new, however, not much is known about the computational behaviour of the consistency problem of its sub-algebras, except in the case of the recently found subset of convex branching relations, for which the consistency of a network can be tested via path consistency and it is therefore deterministic polynomial. In this paper, following Nebel and Bürckert, we define the Horn fragment of Branching Algebra, and prove that it is a sub-algebra of the latter, being closed under inverse, intersection, and composition, that it strictly contains both the convex fragment of Branching Algebra and the Horn fragment of Interval Algebra, and that its consistency problem can be decided via path consistency. Finally, we experimentally prove that the Horn fragment of Branching Algebra can be used as an heuristic for checking the consistency of a generic network with a considerable improvement over the convex subset.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** Constraint programming, Consistency, Branching time, Horn Fragment

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.5

**Funding** *Guido Sciavicco*: G. Sciavicco acknowledges partial support by the Italian INDAM GNCS project *Strategic Reasoning and Automated Synthesis of Multi-Agent Systems*.

## 1 Introduction

In the context of temporal reasoning, Allen's Interval Algebra [1] (*IA*) is certainly one of the most important formalisms. Applications of the *IA* are widespread, and range from scheduling, to planning, database theory, and natural language processing, among others. In Allen's *IA* we consider the domain of all intervals on a linear order, and define thirteen basic relations ( $IA_{basic}$ ) between pairs of intervals (such as, for example, *meets* or *before*); a constraint between two intervals is any disjunction of basic relations, and a network of constraints is defined as a set of variables plus a set of constraints between them, interpreted as a logical conjunction. Among the problems that emerge naturally in this field, checking the consistency of a network  $N$  of constraints is probably the most relevant one, and consists



© Alessandro Bertagnon, Marco Gavanelli, Alessandro Passantino, Guido Sciavicco, and Stefano Trevisani;

licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 5; pp. 5:1–5:16



Leibniz International Proceedings in Informatics

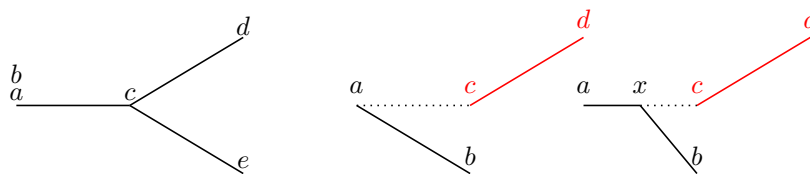
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of deciding whether  $N$  can be realized, that is, deciding if every variable can be instantiated to an interval without violating any constraint. The consistency problem is archetypical of the class of constraints satisfaction problems (CSP), because a network is a conjunction of constraints. The consistency problem for the  $IA$  is NP-complete, and classical approaches to efficient implementations are based either on clever brute-force enumerating algorithms (see, e.g. [8, 18]), or on tractable fragments of the algebra, which are interesting both on their own [9] and as heuristics to reduce the branching factor in branch-and-bound approaches for the full algebra [10, 13]. Two important fragments of the  $IA$  are the convex fragment ( $IA_{convex}$ ), introduced by van Beek and Cohen [23], and encompassing 82 relations, and the more general ORD-HORN fragment (or, simply, the Horn fragment -  $IA_{Horn}$ ), introduced by Nebel and Bürckert in [14], with 868 relations. In particular, to prove the tractability of the latter, Nebel and Bürckert identify a suitable point-based language that allows one to translate every relation of the Horn fragment of  $IA_{Horn}$  to a conjunction of Horn clauses; then, they prove that  $IA_{Horn}$  is closed under inverse, intersection, and composition, and that path consistency is complete for it.

In [15], the authors define a branching version of Allen's  $IA$ , which we refer to as Branching Algebra ( $BA$ ), and introduce two possible sets of basic relations that may hold between two intervals on a tree-like partial order. One of these sets, composed of 24 mutually exclusive and jointly exhaustive basic relations, and also studied from the (first-order) expressive power point of view in [5], is characterized by basic relations whose semantics cannot be always written in the language of endpoints, therefore requiring quantification. By joining some of these relations via disjunction, one obtains a second set of 19, still mutually exclusive and jointly exhaustive, relations ( $BA_{basic}$ ), each of which is translatable to the language of endpoints without using quantification. The consistency problem for a network of constraints in the algebra that emerges from these relations is, quite obviously, still NP-hard, and, in general, computationally more difficult than the one for  $IA$ . In [6], the authors presented the subset  $BA_{convex}$  of convex  $BA$ -relations, inspired by the convex fragment of the  $IA$  ( $IA_{convex}$ ). The fragment  $BA_{convex}$ , that encompasses 91 relations, unlike its linear analogous, is not a subalgebra of  $BA$ , as it is not closed under composition; yet, it is closed on the (less restricting) operation of path consistency, which is also complete (w.r.t. deciding consistency) for it, making  $BA_{convex}$  the first non-trivial tractable fragment of  $BA$ . In this paper, we follow Nebel and Bürckert's approach, and define, first, a first order Horn theory (TORD-HORN), whose models can be interpreted as trees and in which  $BA$ -relations can be translated; then, we enumerate the subset of all and only  $BA$ -relations that can be translated in the language of TORD-HORN; finally, we prove, by enumeration, that such a subset (which we call  $BA_{Horn}$ ) is closed under inverse, intersection, and composition, and it is therefore a subalgebra of  $BA$ . Finally, in the spirit of [6, 17], we implement a simple branch-and-bound algorithm for  $BA$ -networks to empirically study the expected improvement in computation time when the splitting is driven by  $BA_{Horn}$ -relations instead of basic relations.

## 2 Preliminaries

**Notation.** Let  $(\mathcal{T}, <)$  be a partial order, whose elements are generally denoted by  $a, b, \dots$ , and where  $a \parallel b$  (resp.,  $a \text{ lin } b$ ) denotes that  $a$  and  $b$  are incomparable (resp., comparable) with respect to the ordering relation  $<$ . We use  $x, y, \dots$  to denote variables in the domain of points, and  $x \leq y$  to denote  $x < y \vee x = y$ . A partial order  $(\mathcal{T}, <)$ , often denoted by  $\mathcal{T}$ , is a *future branching model of time* (or, simply, a *branching model*) if for all  $a, b \in \mathcal{T}$  there is a greatest lower bound of  $a$  and  $b$  in  $\mathcal{T}$ , and, if  $a \parallel b$  then there exists no  $c \in \mathcal{T}$  such that  $c > a$



■ **Figure 1** A pictorial representation of the four basic branching point relations, where  $a = b$ ,  $a < c$ ,  $d > c$ , and  $d \parallel e$  (left-hand side), and an example of two situations that require quantification to be distinguished in the language of endpoints (right-hand side).

■ **Table 1** Composition of basic branching relations between points.

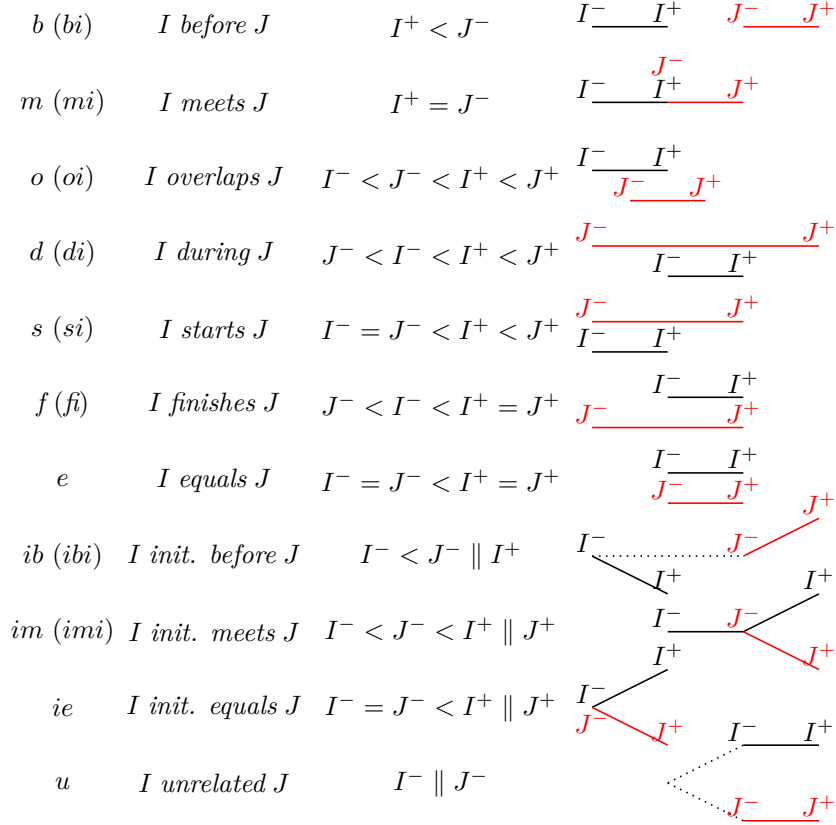
$\circ$	$<$	$>$	$=$	$\parallel$
$<$	$\{<\}$	$lin$	$\{<\}$	$\{\parallel, <\}$
$>$	$?$	$\{>\}$	$\{>\}$	$\{\parallel\}$
$=$	$\{<\}$	$\{>\}$	$\{=\}$	$\{\parallel\}$
$\parallel$	$\{\parallel\}$	$\{>, \parallel\}$	$\{\parallel\}$	$?$

and  $c > b$  (that is, it is a tree). There are four basic relations that may hold between two points on a branching model: *equals* ( $=$ ), *incomparable* ( $\parallel$ ), *less than* ( $<$ ), and *greater than* ( $>$ ); the first two are symmetric, while the last two are inverse of each other. These relations are depicted in Figure 1 (left-hand side), and are called *basic branching point relations*. The set of basic branching point relations is denoted by  $BPA_{basic}$ . In the linear setting, the set of basic relations has only three elements,  $<$ ,  $=$ , and  $>$ , and it is called  $PA_{basic}$  (*basic point relations*). An *interval* in  $\mathcal{T}$  is a pair  $[a, b]$  where  $a < b$ , and  $[a, b] = \{x \in \mathcal{T} : a \leq x \leq b\}$ . Intervals are generically denoted by  $I, J, \dots$ . For an interval  $I$  (resp.,  $X$ ), we use  $I^-, I^+$  to denote its endpoints. Following [5], one can describe 24 basic branching relations based on the possible relative position of two pairs of ordered points on a branching model, that is, by directly generalizing the universally known set of 13 *basic interval relations* [1] ( $IA_{basic}$ ). While towards a precise study of the expressive power of branching relations in a first-order context this is an optimal choice, this is no longer true when studying the computational properties of the consistency problem. In particular, some of these relations require first-order quantification to be defined: for example, in Figure 1 (right-hand side) we see that, in order to distinguish the two situations, we need to quantify of the existence, or non-existence, of a point between  $a$  and  $c$ . To overcome this problem, that becomes relevant when we study the behaviour of branching relations in association with the behaviour of branching point relations (that is, by studying the properties of their point-based translations), Ragni and Wöfl [15] introduce a set of coarser relations, characterized by being translatable to point-based relations using only the language of endpoints, without quantification. These 19 relations are depicted in Figure 2, and form the set of *basic branching interval relations* ( $BA_{basic}$ ); for each relation, the symbol in parentheses corresponds to its inverse, if the relation is not symmetric. A relation in the set  $BA_{basic}$  is either a linear relation, or the relation  $u$  (*unrelated*), or it corresponds to the disjunction between a pair of finer relations from the set of 24 [5]. For example, the relation  $ib$  is the disjunction of the two relations in Figure 1.

**Operations and algebras.** In general, given the basic relations  $r_1, \dots, r_l$ , we denote by  $R = \{r_1, \dots, r_l\}$  the disjunctive relation  $r_1 \vee \dots \vee r_l$ ; thus, a relation is seen as a set, and a basic relation as a singleton. As the set  $IA_{basic}$  contains 13 elements, the set  $IA$  of all



## 5:4 The Horn Fragment of Branching Algebra



■ **Figure 2** A pictorial representation of the nineteen basic branching interval relations. In this picture, in which  $I = [I^-, I^+]$  and  $J = [J^-, J^+]$ , we assume  $I^- < I^+$  and  $J^- < J^+$ . Solid lines are actual intervals, dashed lines complete the underlying tree structure. We use  $aR_1bR_2c$  as a shorthand for  $aR_1b$  and  $bR_2c$ .

interval relations in the linear setting encompasses  $2^{13} - 1$  elements; similarly, the set  $BA_{basic}$  of 19 basic relations entails  $2^{19} - 1$  interval relations in the branching setting. A *constraint* is an object of the type  $xRy$ , where  $x, y$  are point variables and  $R$  is a relation. There are three basic operations with relations: (Boolean) intersection, inverse, and composition. The *inverse* of a relation  $R = \{r_1, \dots, r_l\}$  is the relation  $R^{-1} = \{r_1^{-1}, \dots, r_l^{-1}\}$ , where, for each basic relation  $r$ ,  $r^{-1}$  is its inverse. In our notation, for example, *bi* (*later*) denotes the inverse of the basic relation *b* (*before*). The *composition* of two basic relations  $r_1, r_2$  is defined as follows: for variables  $s, t, z$ , we say that  $s$  is in the *composed relation*  $r_1 \circ r_2$  with  $t$ , denoted  $s(r_1 \circ r_2)t$ , if there exists  $z$  such that  $sr_1z$  and  $zr_2t$ . The composition of two relations  $R_1, R_2$  is defined component-wise:  $R_1 \circ R_2 = \{r \mid \exists r_1 \in R_1 \exists r_2 \in R_2 (r = r_1 \circ r_2)\}$ . When a set of relations  $A$  is closed under inverse, intersection, and composition, we call it an *algebra*. Clearly, to compute the composition of two non-basic relations we base ourselves on the composition between basic relations, and to compute the latter in the interval ontology, both in the linear and the branching setting, we use the composition between basic relations in the point ontology. The latter can be easily computed “by hand” (see Table 1 for the branching case). The entire composition table between two intervals in the branching case is fully reported in [16] (and in [2] in the linear case). Given a set  $A$  of relations, an *A-network* is a directed graph  $N = (V, E)$ , where  $V$  is a set of variables and  $E \subseteq V \times V$  is a set of



$A$ -constraints between pairs of variables. To denote a constraint between the variables  $s$  and  $t$  in a network, we use indistinctly the notation  $(s, t)$  or the infix notation  $sRt$  (when we want to specify the relation). Given a network  $N = (V, E)$ , we say that  $N'$  is a sub-network of  $N$  if  $N' = (V', E')$ ,  $V' \subseteq V$ , and  $E'$  is the projection of  $E$  on the variables in  $V'$ . Given a network, we say that it is *consistent* if there exists a model such that each variable can be mapped (*realized*) to a concrete element so that every constraint is respected; establishing if an  $A$ -network is consistent is the  $A$ -consistency problem.

**Tractability and local consistency.** Consistency problems such as those for  $IA$ ,  $BA$ , and their fragments are often approached via popular heuristics such as constraint propagation and local consistency. A network  $N$  is said to be  $k$ -consistent if, given any consistent realization of  $k - 1$  variables, there exists an instantiation of any  $k$ -th variable such that the constraints between the subset of  $k$  variables can be satisfied together. Because of the particular nature of networks of constraints in temporal algebras, they are always 1-consistent (also called *node consistent*) and 2-consistent (also called *arc consistent*), by definition. Enforcing *path consistency*, that is, 3-consistency, in a network  $N$ , corresponds to apply the following simple algorithm: for every triple  $(s, t, z)$  of variables in  $N = (V, E)$  such that  $sRt, sR_1z, tR_2z \in E$ , replace  $sRt$  by  $s(R \cap (R_1 \circ R_2))t$ . Clearly, if enforcing path consistency results in at least one empty constraint, the entire network  $N$  is not consistent. But, in general, enforcing path consistency (in fact,  $k$ -consistency for any constant  $k < |V|$ ) does not imply consistency, and, indeed checking the consistency of a  $IA$ -network is a NP-hard problem [9]. Much effort has been devoted to identify, and classify, the relevant fragments of the  $IA$  for which the consistency problem becomes tractable. Besides the fragment of basic relations only,  $IA_{basic}$ , which is trivially tractable, two important tractable fragments are the *convex* fragment ( $IA_{convex}$ ), introduced by van Beek and Cohen [23], and encompassing 82 relations, and the more general ORD-HORN fragment (or, simply, the Horn fragment -  $IA_{Horn}$ ), introduced by Nebel and Bürckert in [14], encompassing 868 relations. Both  $IA_{convex}$  and  $IA_{Horn}$  are subalgebras of the  $IA$ , and in both cases checking path consistency is a complete method for checking the consistency.

In analogy with the linear case, Ragni and Wöfl [15] proved that also checking the consistency of a  $BA$ -network is at least NP-hard, and observed that the set of basic  $BA$ -relations only constitutes a tractable fragment (although not an algebra); also, in [6], the authors presented the branching version of the fragment  $IA_{convex}$ , called  $BA_{convex}$ , which is tractable, but, unlike its linear homologue, not closed under composition, and therefore not an algebra. Tractable fragments are not only important per se. As a matter of fact, the consistency problem for the full  $IA$  and  $BA$  alike is NP-complete, thanks to the fact that it can be decided by a simple branch-and-bound algorithm based on basic relations, and the completeness of path consistency for a fragment has another interesting consequence: improving the performances of such an algorithm. A branch-and-bound consistency checking algorithm is a backtracking algorithm that enforces path-consistency in each node of the search tree (more detail is in Section 5). At each step, the algorithm tries one basic relation for each relation. If at any step one relation results in the empty relation, it backtracks to the last choice; otherwise it proceeds to the next relation in the network. Fragments of the full algebra, both in the linear and the branching case, whose consistency can be decided via path consistency can be used to drive the splitting in such an algorithm, as a heuristics to speed up the branch-and-bound process: if, at any step, one ends up with a network whose labels are all contained in any such a fragment, that particular branch can be decided by simply enforcing path consistency. This has been done with both  $IA_{convex}$  and  $IA_{Horn}$  in the linear case, and with  $BA_{convex}$  in the branching case.

### 3 Applying Branching Algebra

Interval algebra has been known for 30 years, and its role in planning and database theory is universally accepted. Temporal reasoning in a branching setting is also a very well-established research area, at least at the logical level. Therefore, studying interval algebra in the branching setting is very natural. Possible application fields include the following ones.

**Planning with errors.** The use of *IA*, and in particular of *IA*-networks of constraints, to model planning problems is ubiquitous in the literature (see, e.g. [11, 12, 24]). A typical modelling exercise involves a set of *tasks* to be executed in order for a *goal* to be reached. Plans that are modelled with linear time, however, allow no margin for error: once the plan is being followed, every task must be executed. Using branching time we can develop plans that have alternative routes that can be taken in case some action fails. While we are following an (initial) part of the plan with actions that have no possibility of failing (in our abstract model), the underlying temporal model is linear; as soon as we encounter a task that may fail, the underlying model becomes branching, and, from that moment onward, different plans may be followed. In this sense, different branches will never join again; so, the underlying model is in fact tree-like, and a network of constraints that takes mistakes or obstacles into account is naturally modelled in *BA*.

**Automatic generation of narrative.** Generation of narrative is a modern application of artificial intelligence, and, more specifically, of natural language processing [22]. While the classical applications of automatically generated narratives include weather reports, instructions, descriptions of museum artifacts, narratives can be also used as the basis of automatic storytelling and plot generation [19]. Many modern and classic science-fiction stories, movies, and even video games make substantial use of parallel, incomparable timelines. To keep an adequate cause-effect consistency, however, in presence of non-trivial literary *escamotage* (such as time travel, for example), modelling the basic elements with *BA* may be a solution. The generated narrative can be checked for consistency to ensure that, while being possibly non-linear, it is internally coherent.

**Verification of parallel programs.** Some techniques for program verification make use of *IA* (see, e.g. [20]). Verifying parallel programs is a challenging task [4] which may take advantage from a branching interval algebra such as *BA*, in which the typical *fork* constructs can be modelled in a natural way. Consistency, in this case, can be interpreted as the absence of temporal contradictions in the executions of (sub)routines.

### 4 The Horn Fragment of *BA*

**Horn branching relations.** Every basic relation of *IA*, interpreted on a linear model  $(\mathcal{T}, <)$  can be translated into a conjunction of formulas of the language of endpoints. Every non-basic relation, obviously, gives rise to a disjunction of such conjunctions, which, in turn, can be re-written into a conjunction of disjunctions, that is, of *clauses*. Thus, a network of constraints can be translated into a conjunction of clauses. Let us denote by  $\Pi(r)$  (resp.,  $\Pi(R)$ ,  $\Pi(N)$ ) the translation of a basic relation (resp., non-basic relation, network), and by  $C, D, \dots$  (resp.,  $\mathcal{C}, \mathcal{D}$ ) a generic clause (resp., set of clauses). As observed in [14], some translations of relations have the additional property that their corresponding set of clauses are *Horn*, that is, each clause has at most one positive literal; these are called *IA<sub>Horn</sub>*-relations. By associating

such a translation to a first-order Horn theory, called ORD-HORN, whose models can be interpreted as linear orders, one obtains that:

- (i) for a network  $N$  of  $IA_{Horn}$ -constraint,  $N$  is consistent if and only if  $\Pi(N) \wedge \text{ORD-HORN}$  is satisfiable, and
- (ii) checking the satisfiability of  $\Pi(N) \wedge \text{ORD-HORN}$  is a tractable problem, for example via *positive unit resolution* [7].

In order to define the branching equivalent of  $IA_{Horn}$ , we need to construct the branching equivalent of ORD-HORN, which we denote by TORD-HORN. First, we need to define the language of TORD-HORN; then, we shall specify its axioms, and prove that every model of TORD-HORN can be interpreted as future branching models of time; finally, we can check which subset of relations of  $BA$  can be translated to the language of TORD-HORN, and that such subset forms an algebra.

► **Definition 1.** *The language of TORD-HORN encompasses an enumerable set of variables  $X, Y, \dots$  and the binary relations  $\doteq$  (equality),  $\preceq$  (less or equal),  $\sim$  (linear),  $\parallel$  (incomparable), and  $\prec_{\parallel}$  (less or incomparable).*

In this context, the theory of future branching models of time cannot be (fully) axiomatized in the standard way, because some of the necessary properties are not in form of Horn formulas (e.g.,  $X \sim Y$  defined as  $X \preceq Y \vee Y \preceq X$ ). However, to our purposes it suffices to have models that can be *extended to tree-like orderings*.

► **Definition 2.** *The theory TORD-HORN is characterized by the following axioms:*

1.  $X \doteq X$  (*reflexivity of  $\doteq$* );
2.  $X \doteq Y \rightarrow Y \doteq X$  (*symmetry of  $\doteq$* );
3.  $X \doteq Y \wedge Y \doteq Z \rightarrow X \doteq Z$  (*transitivity of  $\doteq$* );
4.  $X \preceq X$  (*reflexivity of  $\preceq$* );
5.  $X \preceq Y \wedge Y \preceq X \rightarrow X \doteq Y$  (*antisymmetry of  $\preceq$* );
6.  $X \preceq Y \wedge Y \preceq Z \rightarrow X \preceq Z$  (*transitivity of  $\preceq$* );
7.  $X \not\parallel X$  (*irreflexivity of  $\parallel$* );
8.  $X \parallel Y \rightarrow Y \parallel X$  (*symmetry of  $\parallel$* );
9.  $X \sim X$  (*reflexivity of  $\sim$* );
10.  $X \sim Y \rightarrow Y \sim X$  (*symmetry of  $\sim$* );
11.  $X \not\prec_{\parallel} X$  (*irreflexivity of  $\prec_{\parallel}$* );
12.  $X \prec_{\parallel} Y \wedge Y \prec_{\parallel} X \rightarrow X \parallel Y$  (*antisymmetry of  $\prec_{\parallel}$* );
13.  $X \doteq Y \rightarrow X \preceq Y \wedge Y \preceq X \wedge X \sim Y$  (*weakening of  $\doteq$* );
14.  $X \preceq Y \rightarrow X \sim Y$  (*weakening of  $\preceq$* );
15.  $X \parallel Y \rightarrow X \prec_{\parallel} Y \wedge Y \prec_{\parallel} X$  (*weakening of  $\parallel$* );
16.  $X \parallel Y \rightarrow X \not\preceq Y \wedge Y \not\preceq X$  (*compatibility of  $\parallel$  and  $\preceq$* );
17.  $X \sim Y \rightarrow X \not\parallel Y$  (*compatibility of  $\sim$  and  $\parallel$* );
18.  $X \prec_{\parallel} Y \rightarrow Y \not\preceq X$  (*compatibility of  $\prec_{\parallel}$  and  $\preceq$* );
19.  $X \parallel Y \wedge Y \preceq Z \rightarrow X \parallel Z$  (*tree-likeness*).

In the following, we denote by TORD-HORN the set of axioms 1-19<sup>1</sup>; observe that TORD-HORN is a Horn theory. We use the language of TORD-HORN to translate certain relations of  $BA$ ; as we have recalled, such a translation is correct if and only if the resulting model can

<sup>1</sup> We do not claim these axiom are minimal; having a minimal set of axioms, however possible, would probably hid some of the underlying structure.

be interpreted as a future branching model of time. It turns out that, in order to guarantee that this is possible, we need to further limit the use of the language of TORD-HORN in translations, and, in particular, we say that  $C$  is an *admissible clause* if it uses only literals with the positive relations  $\dot{=}, \dot{\preceq}, \sim, \parallel, \prec_{\parallel}$ , and the negative relation  $\neq$ . Observe that limiting the use of certain relations does not decrease the (semantic) expressive power of the language, as  $X \not\dot{\preceq} Y$  (resp.,  $X \not\sim Y$ ,  $X \not\parallel Y$ ,  $X \not\prec_{\parallel} Y$ ) can be written as  $Y \prec_{\parallel} X$  (resp.,  $X \parallel Y$ ,  $X \sim Y$ ,  $Y \dot{\preceq} X$ ).

► **Theorem 3.** *Every model  $(\mathcal{M}, \dot{=}, \dot{\preceq}, \sim, \parallel, \prec_{\parallel})$  of  $\text{TORD-HORN} \cup \mathcal{C}$ , where  $\mathcal{C}$  is a set of admissible clauses, can be represented as a branching model of time.*

It is important to remark that the use of an extended signature to specify the properties of a tree-like model is justified by the need of such a specification to be Horn. Admissible Horn clauses, as it can be proved by computer-assisted enumeration, are expressive enough to translate a subset of  $BA$ -relations that form an algebra, and allowing any of the forbidden symbols would require some non-Horn axiom.

► **Definition 4.** *The set  $BA_{Horn}$  is the subset of  $BA$  of all and only the relations that can be translated to the language of TORD-HORN using only admissible Horn clauses.*

► **Theorem 5.**  *$BA_{Horn}$  is an algebra, that is, it is closed under inverse, intersection, and composition.*

The set  $BA_{Horn}$  can be computed automatically, and it consists of 4510 relations. Although it covers less than 1% of the entire algebra, it is about 50 times more extended than  $BA_{convex}$ .

**Completeness of path consistency.** Let us consider a network  $N$  of  $BA_{Horn}$ -constraints. By the above results, we know that  $N$  is consistent if and only if  $\Pi(N) \wedge \text{TORD-HORN}$  is satisfiable. Now, we ask ourselves if the consistency of  $N$  can also be checked by path consistency, in the same way in which the consistency of a network of  $IA_{Horn}$ -constraints can. Again following [14], proving that path consistency is complete for  $BA_{Horn}$  boils down to proving that, given a path consistent network  $N$ , the empty clause cannot be derived from  $\Pi(N) \wedge \text{TORD-HORN}$ ; to show the latter, one can restrict the attention to derivations that use positive unit resolution, which is complete for Horn clauses [7].

Let  $N$  be a path consistent  $BA_{Horn}$ -network. Let  $\Pi(N) = \{\varphi_1, \varphi_2, \dots\}$  be the Horn formulas of the signature TORD-HORN that are the result of translating the  $BA_{Horn}$ -constraints of  $N = \{IR_1J, KR_2Z, \dots\}$ ; each  $\varphi_i$  is a conjunction of Horn clauses. The following observation will be relevant for us: by exhaustive exploration of all clauses that can be obtained from translating  $BA_{Horn}$ -relations, we realize that they either are unary or of the type:

$$(X \diamond Y \vee X \neq Y) \text{ or } (Y \diamond X \vee X \neq Y),$$

where  $\diamond \in \{\dot{\preceq}, \parallel, \prec_{\parallel}\}$ . In the following we assume that each formula  $\varphi_i$  is *explicit*, that is, it explicitly contains all consequences of every axiom of TORD-HORN, and that each clause  $C_j \in \varphi_i$  is *minimal*, that is, it contains no redundant literal; a set  $\Pi(N)$  in which every formula is explicit, and every clause in every formula is minimal will be called *explicit* and *clause-minimal*. We want to prove that if  $N$  is path consistent and contains no empty relation, then positive unit resolution cannot deduce the empty clause from  $\Pi(N) \wedge \text{TORD-HORN}$ .

► **Theorem 6.** *Let  $N$  be a path consistent  $BA_{Horn}$ -network. Then, if  $\Pi(N)$  is explicit and clause-minimal, then the empty clause cannot be obtained from  $\Pi(N) \wedge \text{TORD-HORN}$  by positive unit resolution.*

■ **Algorithm 1** Backtracking algorithm.

---

```

1: function CONSISTENT( $P, Split$ )
2:   enforce generalized arc consistency on  $P$ 
3:   if there is a variable  $\nu_{XY}$  such that  $\mathfrak{D}_{XY} = \emptyset$  then
4:     return false
5:   else
6:     choose an unprocessed variable  $\nu_{XY}$  such that  $\mathfrak{D}_{XY} \notin Split$ 
7:     if there is no such variable then
8:       return true
9:      $\{\mathfrak{D}_1, \dots, \mathfrak{D}_p\} = \text{PARTITION}(\mathfrak{D}_{XY}, Split)$ 
10:    for all  $\mathfrak{D}_i \in \{\mathfrak{D}_1, \dots, \mathfrak{D}_p\}$  do
11:       $P' = P_{\mathfrak{D}_{XY}/\mathfrak{D}_i}$ 
12:      if CONSISTENT( $P', Split$ ) then
13:        return true
14:    return false

```

---

► **Corollary 7.** *Path consistency is complete for checking the consistency of a network of  $BA_{Horn}$ -relations.*

## 5 Experiments

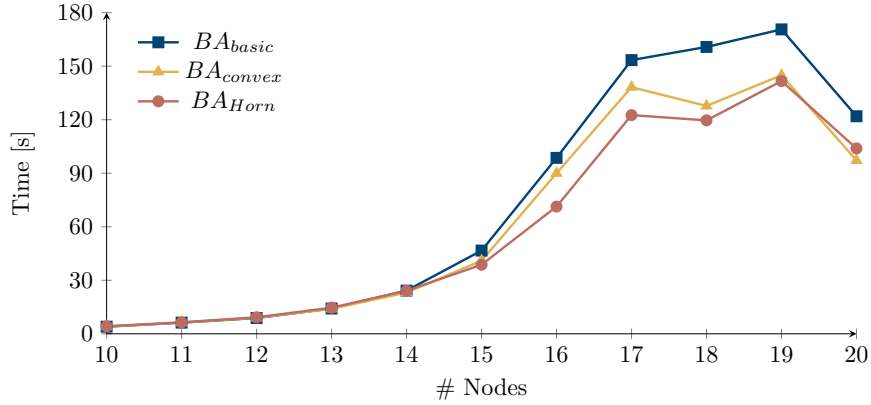
In order to assess the usefulness of the fragment  $BA_{Horn}$  to improve the experimental computation time for checking the consistency of a network of  $BA$ -constraints, we devised a series of tests.

**Constraint satisfaction problems.** We designed a simple algorithm based on encoding the temporal network into a *constraint satisfaction problem (CSP)* using the classical *dual CSP* approach by Condotta et al. [3], based on the fact that enforcing path consistency on the original qualitative temporal network corresponds to enforcing *generalized arc consistency* on the corresponding dual CSP.

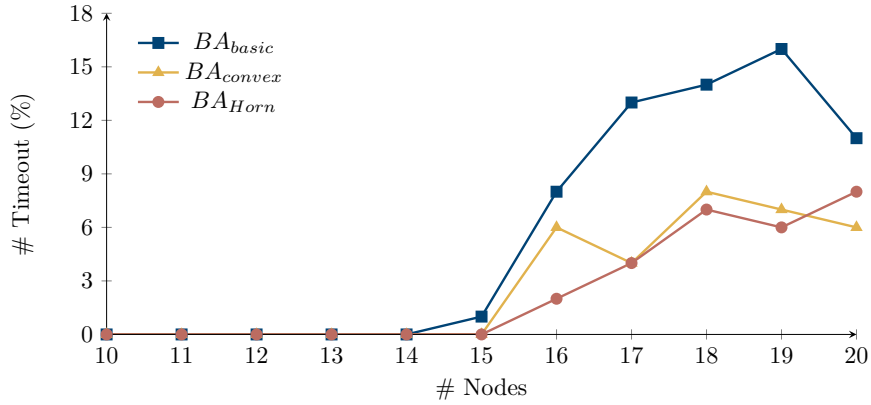
► **Definition 8.** *Given a  $BA$ -network  $N = (V, E)$ , its dual CSP is a triple  $P = (\mathfrak{V}, \mathfrak{D}, \mathfrak{C})$ , where  $\mathfrak{V}$  is a set of variables,  $\mathfrak{D}$  is a set of variable domains, and  $\mathfrak{C}$  is a set constraints, such that:*

- (i)  $\mathfrak{V}$  contains a variable  $\nu_{XY}$  for each pair of nodes  $X, Y \in V$ ;
- (ii)  $\mathfrak{D}$  contains a domain  $\mathfrak{D}_{XY}$  for each variable in  $\mathfrak{V}$ , which corresponds to the constraint  $XRY \in E$ , and
- (iii)  $\mathfrak{C}$  contains a binary constraint  $\text{inverse}(\nu_{XY}, \nu_{YX})$  for each pair of nodes  $X, Y \in V$ , satisfied by all pairs  $(r, r^{-1})$ , where  $r \in BA_{basic}$ , and a ternary constraint  $\text{composition}(\nu_{XY}, \nu_{YZ}, \nu_{XZ})$  for each triple of nodes  $X, Y, Z \in V$ , which encodes the composition table and is satisfied by all triples  $(r_1, r_2, r_3)$  such that  $r_3 = r_2 \circ r_1$ .

Since path consistency is not complete for consistency checking of general networks, it is typically associated to a search algorithm, such as the one depicted in Algorithm 1 [6, 13]. Algorithm 1 checks the consistency of a general network; moreover, when there is a known fragment which is tractable through path consistency, Algorithm 1 can exploit it to speedup the search. The family of sets *Split* represents exactly such a tractable fragment. If no tractable fragment is known, the set *Split* contains just basic relations (as singleton sets),

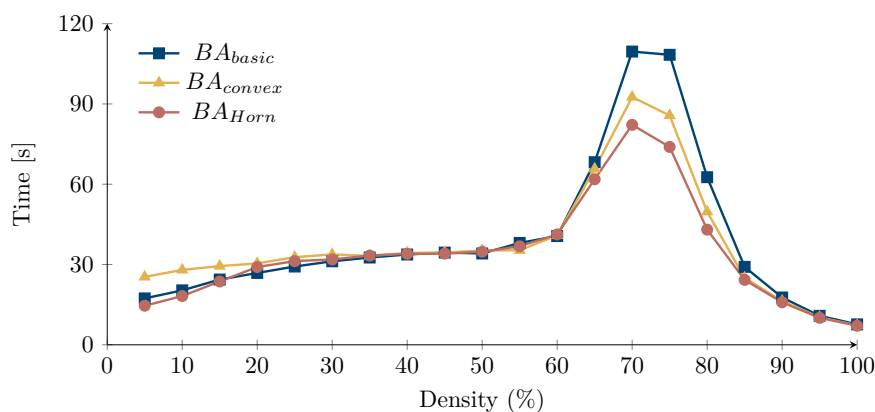


■ **Figure 3** Running time of the backtracking algorithm varying the number of nodes  $n$  of the network. Each point represents the geometric mean of 100 instances, with density  $d = 70\%$ . Different lines represent different fragments as *Split* set.

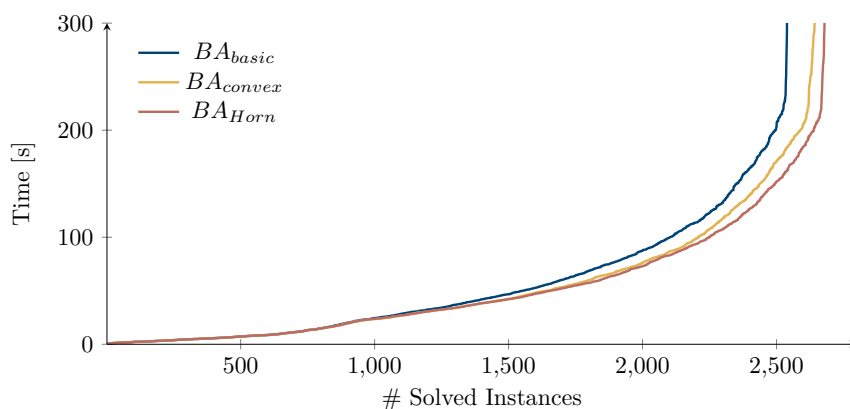


■ **Figure 4** Fraction of instances that incurred in a timeout varying the number of nodes  $n$  of the network and fixing the density to  $d = 70\%$ . Different lines represent different fragments as *Split* set.

and the algorithm amounts to selecting a variable of the CSP, then splitting its domain into the basic relations (function PARTITION), nondeterministically assigning it one of the basic relations in its domain, enforcing path consistency on the obtained network, and recursively solving the remaining part of the CSP. On backtracking, another basic relation is selected, and so on; the search stops when all the variables of the CSP are assigned a basic relation in their domain. This is sound in the case of  $BA$ , since path consistency is complete for consistency for the set of basic  $BA$ -relations [15]. In case a larger fragment (e.g.,  $BA_{convex}$  [6], or  $BA_{Horn}$ ) is known to be solvable by path consistency, such fragment can be effectively used. Again, a variable of the CSP is selected, and its domain is partitioned into subsets, each belonging to the family *Split*. Since the subsets are no longer required to be singleton, the branching factor can be reduced; in general, the larger the fragment, the better the algorithm is expected to behave. Function PARTITION requires the solution of a set-partitioning problem, which is itself NP-hard, in the general case. In our case the trivial solution that splits a domain into its singletons is always feasible, and it can be computed in polynomial time; however such solution is useless, as Algorithm 1 would not exploit the tractable fragment. As in [6], we used a *trie* to store the tractable fragment, and a greedy



■ **Figure 5** Running time of the backtracking algorithm varying the density  $d$  of the network. Each point represents the geometric mean of 50 instances, with number of nodes  $n = 16$ . Different lines represent different fragments as *Split* set.



■ **Figure 6** Cactus plot showing the number of solved instances varying the solving time. Instances have been generated with a number  $n$  of nodes varying from 15 to 20 and a constraint density  $d$  varying from 55% to 100%. Different lines represent different fragments as *Split* set in backtracking algorithm.

algorithm to quickly find a partition of the domain. Algorithm 1 was implemented in the Constraint Logic Programming environment  $ECL^iPS^e$  [21], that is a declarative language with built-in libraries for constraint satisfaction problems.

**Experimental setting and results.** In this experiment, random instances are generated as in [6], with a technique derived from [17]. Each instance is characterized by three parameters: the number of nodes  $n$ , the network density  $d$ , and the probability of a constraint  $p$ . Given the three parameters, for each given cardinality  $n$ , we generate a graph with  $n$  nodes, then we select  $d \frac{n(n-1)}{2}$  edges at random. For each selected edge, we generate its domain by choosing with probability  $p$  each of the basic relations in  $BA_{basic}$ . Edges not selected are associated with the universal relation. Our experiments aim to assess the improvement of the backtracking algorithm when the  $BA_{Horn}$  fragment is used as *Split* heuristics as opposed to use the  $BA_{convex}$  fragment or using basic relations only.



In Figure 3 each point represents the geometric mean of 100 problem instances with density of the network fixed to 70%. The results suggest that using the  $BA_{Horn}$  fragment positively influences the computation time, not only with respect to not using it but also with respect to using the  $BA_{convex}$  fragment. Figure 4 where the fraction of instances that incurred in a timeout is plotted, confirm this observation. Figure 5 shows the running time of the backtracking algorithm varying the density of the network, while fixing the number of nodes to 16. Each point represents the geometric mean of 50 instances. The shape of the curves shows the *phase transition* as shown by  $BA_{convex}$  fragment in [6]: low density networks are easily satisfiable, while in high density networks the unsatisfiability is easily provable. Note that the new fragment improves in particular in the hardest region, at a density between 70% and 80%, in which both satisfiability and unsatisfiability are hard to prove. Finally, we generated 3000 random instances varying the number of nodes  $n$  from 15 to 20 and varying the constraint density  $d$  from 55% to 100%; also the cactus plot in Figure 6 shows that exploiting the  $BA_{Horn}$  fragment leads to an improvement in computation time. All experiments were run on ECL<sup>i</sup>PS<sup>e</sup> v. 7.0, build #54, with a time limit of 600s on Intel<sup>®</sup> Xeon<sup>®</sup> E5-2630 v3 CPUs running at 2.4GHz on CentOS Linux 7, using only one core and with 1GB of reserved memory.

## 6 Conclusions

Branching Algebra is the natural branching-time generalization of Allen’s Interval Algebra. Being relatively new, not much is known about the computational behaviour of the consistency problem of its sub-algebras. Branching Algebra has been introduced in [15], where it has been proven that the consistency problem for the subset that includes only basic relations is tractable. Later, in [6], the subset of convex branching relations was introduced, showing that path consistency is complete for consistency in that case as well. In this paper, following Nebel and Bürckert [13], we further extended the convex fragment to obtain the Horn fragment of the Branching Algebra. We proved that it is a subalgebra, being closed under inverse, intersection, and composition, and that its consistency problem is treatable; we also proved that path consistency is complete for consistency in this case as well. Finally, we designed and conducted a series of experiments on randomly generated networks of constraints in the full algebra, to evaluate the improvement in computation time that comes from using the Horn fragment as heuristics.

This paper constitutes yet another step towards the complete classification between tractable/intractable fragments of Branching Algebra. At the moment, the Horn fragment is the biggest tractable known fragment, and our initial investigation points towards its maximality w.r.t. the tractability of the consistency problem. Yet, other, incomparable fragments may exist. The algebra of intervals is traditionally applied to task scheduling. In the branching case, applications are more difficult to visualize; yet, the Branching Algebra can be applied to a variety of situations in which multiple, incomparable timelines co-exist. In this paper, we have suggested a series of possible application scopes, but our list can be certainly extended and further explored.

---

## References

- 1 J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- 2 J.F. Allen and P. J. Hayes. Short time periods. In *Proc. of IJCAI 1987: 10th International Joint Conference on Artificial Intelligence*, pages 981–983, 1987.



- 3 J.F. Condotta, D. D’Almeida, C. Lecoutre, and L. Saïs. From qualitative to discrete constraint networks. In *Proc. of KI 2006: Workshop on Qualitative Constraint Calculi*, pages 54–64, 2006.
- 4 S. Darabi, S.C.C. Blom, and M. Huisman. A verification technique for deterministic parallel programs. In *Proc. of NFM 2017: 9th International Symposium on NASA Formal Methods*, volume 10227 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2017.
- 5 S. Durhan and G. Sciavicco. Allen-like theory of time for tree-like structures. *Information and Computation*, 259(3):375–389, 2018.
- 6 M. Gavanelli, A. Passantino, and G. Sciavicco. Deciding the consistency of branching time interval networks. In *Proc. of TIME 2018: 25th International Symposium on Temporal Representation and Reasoning*, volume 120 of *LIPICs*, pages 12:1–12:15, 2018.
- 7 L. Henshen and L. Wos. Unit refutation and Horn sets. *Journal of the ACM*, 21:590–605, 1974.
- 8 P. Jonsson and V. Lagerkvist. An initial study of time complexity in infinite-domain constraint satisfaction. *Artificial Intelligence*, 245:115–133, 2017.
- 9 A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen’s interval algebra. *Journal of the ACM*, 50(5):591–640, 2003.
- 10 P.B. Ladkin and A. Reinefeld. Fast algebraic methods for interval constraint problems. *Annals of Mathematics and Artificial Intelligence*, 19(3-4):383–411, 1997.
- 11 M. Mantle, S. Batsakis, and G. Antoniou. Large scale reasoning using Allen’s Interval Algebra. In *Proc. of the 15th Mexican International Conference on Artificial Intelligence*, volume 11062 of *Lecture Notes in Computer Science*, pages 29–41. Springer, 2017.
- 12 L. Mudrová and N. Hawes. Task scheduling for mobile robots using interval algebra. In *Proc. of ICRA 2015: International Conference on Robotics and Automation*, pages 383–388. IEEE, 2015.
- 13 B. Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. *Constraints*, 1(3):175–190, 1997.
- 14 B. Nebel and H.J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen’s interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.
- 15 M. Ragni and S. Wöfl. Branching Allen. In *Proc. of ISCS 2004: 4th International Conference on Spatial Cognition*, volume 3343 of *Lecture Notes in Computer Science*, pages 323–343. Springer, 2004.
- 16 A.J. Reich. Intervals, points, and branching time. In *Proc. of TIME 1994: 9th International Symposium on Temporal Representation and Reasoning*, pages 121–133. IEEE, 1994.
- 17 J. Renz and B. Nebel. Efficient methods for qualitative spatial reasoning. *Journal of Artificial Intelligence Resoning*, 15:289–318, 2001.
- 18 J. Renz and B. Nebel. Qualitative spatial reasoning using constraint calculi. In *Handbook of Spatial Logic*, pages 161–215. Springer, 2007.
- 19 E. Rishes, S.M. Lukin, D.K. Elson, and M.A. Walker. Generating different story tellings from semantic representations of narrative. In *Proc. of ICIDS 2013: 6th International Conference on Interactive Storytelling*, volume 8230 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2013.
- 20 G. Rosu and S. Bensalem. Allen linear (interval) temporal logic - translation to LTL and monitor synthesis. In *Proc. of CAV 2006: 18th International Conference on Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2006.
- 21 J. Schimpf and K. Shen.  $Ecl^1ps^e$  - from LP to CLP. *Theory and Practice of Logic Programming*, 12(1-2):127–156, 2012.
- 22 M. Theune, K. Meijs, D. Heylen, and R. Ordelman. Generating expressive speech for storytelling applications. *IEEE Transactions on Audio, Speech & Language Processing*, 14(4):1137–1144, 2006.

- 23 P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.
- 24 A.K. Zaidi and L.W. Wagenhals. Planning temporal events using point-interval logic. *Mathematical and Computer Modelling*, 43(9):1229–1253, 2006.

## A Appendix

**Proof.** (of Theorem 3) Since  $\doteq$  is an equivalence relation, we can take the quotient  $\mathcal{M}/\doteq$ , denoted  $\mathcal{T}$ , and equipped with the canonical equivalence  $=$ . In the following, we denote by  $x, y, \dots$ , rather than  $[X]_{/\doteq}, [Y]_{/\doteq}$ , the elements of  $\mathcal{T}$ . We define the binary relation  $\leq$  between classes:

$$x \leq y =^{def} \exists X, Y (X \in x \wedge Y \in y \wedge X \preceq Y),$$

and, consequently,  $x < y$  as  $x \leq y \wedge x \neq y$ . We want to prove that  $(\mathcal{T}, <)$  can be extended to a branching model of time.

- $\leq$  is an ordering relation. Clearly,  $\leq$  is reflexive and antisymmetric because so is  $\preceq$ . Moreover, assume that  $x \leq y$  and  $y \leq z$  for some  $x, y, z$ . This means that  $X \preceq Y$  and  $Y' \preceq Z$  for some  $X, Y, Y', Z$  such that  $X \in x, Y, Y' \in y$ , and  $Z \in z$ . But since  $Y, Y' \in y$ , we have that  $Y \doteq Y'$ , and by axiom 13 we know that  $Y \preceq Y'$ . Since  $\preceq$  is transitive, we obtain that  $X \preceq Z$ , implying that  $x \leq z$ . So,  $\leq$  is also transitive. This also implies that  $<$  is a strict pre-order, as it is irreflexive (because  $\doteq$  is reflexive).
- $\leq$  can be extended to a tree-like order. To see this, observe that tree-likeness could be violated by having  $x \not\leq y, y \not\leq x, y \leq z$ , and either  $x \leq z$  or  $z \leq x$  for some  $x, y, z$ , but  $\not\leq$  simply cannot be generated by the set  $\mathcal{C}$ , since it contains only admissible clauses. Because we need to interpret every symbol of the language of TOR-D-HORN, let us define the *incomparable* relation between classes, as:

$$x \parallel y =^{def} \exists X, Y (X \in x \wedge Y \in y \wedge X \parallel Y),$$

which is well-defined thanks to axiom 7 and axiom 8. To ensure that  $(\mathcal{T}, <)$  can be extended to a tree-like ordering, we also have to guarantee that the introduction of  $\parallel$  does not generate any contradictions. So, suppose that  $x \parallel y, x \leq z$ , and  $y \leq t$  for some  $x, y, z, t$ . By definition, for some  $X \in x$  and  $Y \in y$  we have that  $X \parallel Y$ . Moreover, since  $x \leq z$ , for some  $X' \in x$  and  $Z \in z$  we have that  $X' \preceq Z$ . But this implies, by axiom 13, that  $X \preceq Z$ . So, axiom 19 applies, implying that  $Y \parallel Z$ . The same argument can be re-applied, leading us the conclusion that  $Z \parallel T$ . By definition, this implies that  $z \parallel t$ . By contradiction, assume now that  $x \parallel y$  and  $x \leq y$  for some  $x, y$ . This means that  $X \parallel Y$  and  $X' \preceq Y'$  for some  $X, X' \in x$  and  $Y, Y' \in y$ . By axiom 13 and axiom 6, this implies that  $X \preceq Y$ , which is in contradiction with axiom 16. As a consequence of these two facts we have that  $x \parallel y \leftrightarrow (x \not\leq y \wedge y \not\leq x)$  is realizable in  $(\mathcal{T}, <)$ . Now define:

$$x \text{ lin } y =^{def} \exists X, Y (X \in x \wedge Y \in y \wedge X \sim Y),$$

Clearly *lin* is reflexive and symmetric because so is  $\sim$ , which implies that it is well-defined. Once again, we need to make sure that introducing *lin* does not generate contradictions. So, suppose, by contradiction, that  $x \text{ lin } y$  and  $x \parallel y$  hold for some  $x, y$ . This means that  $X \sim Y$  and  $X' \parallel Y'$  for some  $X, X' \in x$  and  $Y, Y' \in y$ . By axiom 13, this implies that  $X \parallel Y$ , which is in contradiction with axiom 17. Similarly, assume that  $x \leq y$  for some  $x, y$  (the case in which  $y \leq x$  or  $x = y$  are similar). This means that  $X \preceq Y$  for some

$X \in x$  and  $Y \in y$ . By axiom 14, this implies that  $X \sim Y$ , leading us to conclude that  $x \text{ lin } y$ . Finally, since  $\mathcal{C}$  is admissible,  $x \text{ lin } y \wedge x \not\parallel y$  cannot occur. As a consequence, we have that  $x \text{ lin } y \leftrightarrow (x \leq y \vee y \leq x \vee x = y)$  is realizable in  $(\mathcal{T}, <)$ . Finally, let us define:

$$x <\parallel y \stackrel{\text{def}}{=} \exists X, Y (X \in x \wedge Y \in y \wedge X <\parallel Y),$$

which is well-defined thanks to axiom 11, 12, and 15. Suppose that, for some  $x, y$  it is the case that  $x <\parallel y$  and  $y \leq x$ . This means that  $X <\parallel Y$  and  $Y' \preceq X'$  for some  $X, X' \in x$  and  $Y, Y' \in y$ . But since  $X, X' \in x$  and  $Y, Y' \in y$ , we have that  $X \doteq X'$  and  $Y \doteq Y'$ , and by axiom 13 and axiom 6, we know that  $X \preceq Y$ , which is in contradiction with axiom 18. Moreover, since  $\mathcal{C}$  is admissible,  $x <\parallel y \wedge y \not\leq x$  cannot occur. As a consequence, we have that  $x <\parallel y \leftrightarrow x < y \vee x \parallel y$  is realizable in  $(\mathcal{T}, <)$ .

In conclusion, the structure  $(\mathcal{T}, <)$  can be extended to a branching model of time, as we wanted.  $\blacktriangleleft$

**Proof.** (of Theorem 6) We prove a stronger claim, that is, we prove that if  $N$  is a path consistent  $BA_{Horn}$ -network, and  $\Pi(N)$  is explicit and clause-minimal, then no new positive unit clauses at all can be deduced by positive unit resolution from  $\Pi(N) \wedge \text{TORD-HORN}$ . As a matter of fact, to deduce a new unit clause, it must be the case that  $\Pi(N) \wedge \text{TORD-HORN}$  contains one clause  $C = \neg L_1 \vee \neg L_2 \vee \dots \vee \neg L_q \vee L$  (where  $L_1, L_2, \dots$  are propositional atoms), and a sequence of positive unit clauses  $C_1 = L_1, C_2 = L_2, \dots, C_q = L_q$ , but does not contain the clause  $C = L$ . Moreover, it must also be the case that  $q \leq 2$ , as we have observed that clauses of  $\Pi(N)$  are at most binary, and instances of axioms are at most ternary. We proceed by case analysis.

- Suppose, first, that  $C$  and  $C_1, \dots, C_q$  belong to  $\Pi(N)$ . If their variables are endpoints of different interval variables, then no resolution step can be applied. Suppose, then, that they contain the same endpoint variables; therefore, they also belong to the same formula  $\varphi_i$ . So, it must be the case that  $C = X \diamond Y \vee X \neq Y$ ,  $C_1 = X \doteq Y$ , and  $q = 1$  (because, as we have observed,  $\diamond$  must be positive). But, as it turns out,  $\diamond \notin \{\doteq, \preceq, \sim\}$ , otherwise  $\Pi(N)$  could not be explicit, and  $\diamond \notin \{\parallel, <\parallel\}$ , otherwise  $\Pi(N)$  could not be clause-minimal. Therefore,  $C, C_1, \dots, C_q$  cannot all belong to  $\Pi(N)$ .
- Suppose, then, that  $C$  is an instance of some transitivity axiom (3 or 6). Then, no  $C_j$  can be an instance of some irreflexivity axiom (7 or 11), because it would not be positive, neither can be an instance of any other axiom except 1, 4, and 9, because it would not be unitary; no resolution step can be carried on with 9, because  $\sim$  is not transitive, and the only possible resolution steps that could be completed with the reflexivity of  $\doteq$  and  $\preceq$  would lead to tautologies. Therefore, every  $C_j$  must belong to  $\Pi(N)$ . If they are all clauses of the same formula  $\varphi$ , then either  $C_1 = X \preceq Y$ ,  $C_2 = Y \preceq Z$ , and  $q = 2$ , in which case  $C = X \preceq Z \in \varphi$  as well because  $\varphi$  is explicit, or  $C_1 = X \doteq Y$ ,  $C_2 = Y \doteq Z$ , and  $q = 2$ , in which case  $C = X \doteq Z \in \varphi$  for the same reason. Therefore,  $C_1$  belongs to  $\varphi_1$ , which translates some constraint  $IR_1J$ , and  $C_2$  belongs to  $\varphi_2$ , which translates some constraint  $JR_2K$  (if the constraints referred to completely different interval variables, then the endpoints variables would be different, and no resolution step could be performed). As before, either  $C_1 = X \preceq Y$  and  $C_2 = Y \preceq Z$ , or  $C_1 = X \doteq Y$  and  $C_2 = Y \doteq Z$ , and in both cases  $q = 2$ . Because  $N$  is path consistent, the constraint  $IR_3K$  exists, and  $R_3 \subseteq R_1 \circ R_2$ . Thus,  $\Pi(N)$  also contains its translation  $\varphi_3$ . Since

- (i)  $R_3$  is stronger than  $R_1 \circ R_2$ ,

- (ii) composition is the systematic application of the transitivity axiom(s) and the tree-likeness axiom (see Table 1), and
- (iii)  $L$  (which is either  $X \preceq Z$  or  $X \doteq Z$ ) can be deduced from  $C, C_1$ , and  $C_2$ ,  
it must be the case that  $L \in \varphi_3$ , so, also in this case, no new deduction can be performed.
- Assume, therefore, that  $C$  is an instance of the tree-likeness axiom. Then no  $C_j$  can be an instance of some reflexivity axiom (1 or 4), because  $L$  would not be new, nor can it be an instance of some irreflexivity axiom (7 or 11), because it would not be positive. Also,  $C_j$  can never be the instance of any other axiom because it would not be unitary. Therefore, every  $C_j$  must belong to  $\Pi(N)$ . If they are all clauses of the same formula  $\varphi$ , then  $C_1 = X \parallel Y$ ,  $C_2 = Y \preceq Z$ , and  $q = 2$ , in which case we have that  $X \parallel Z \in \varphi$  as well, because  $\varphi$  is explicit. Therefore,  $C_1$  belongs to  $\varphi_1$ , which translates some constraint  $IR_1J$ , and  $C_2$  belongs to  $\varphi_2$ , which translates some constraint  $JR_2K$  (if the constraints referred to completely different interval variables, then the endpoints variables would be different, and no resolution step could be performed). As above,  $C_1 = X \parallel Y$  and  $C_2 = Y \preceq Z$ , and  $q = 2$ . Because  $N$  is path consistent, there exists a constraint  $IR_3K$ , and  $R_3 \subseteq R_1 \circ R_2$ . Thus,  $\Pi(N)$  also contains its translation  $\varphi_3$ . Since
  - (i)  $R_3$  is stronger than  $R_1 \circ R_2$ ,
  - (ii) composition is the systematic application of the transitivity axiom(s) and the tree-likeness axiom, and
  - (iii)  $L$  (which is  $X \preceq Z$ ) can be deduced from  $C, C_1, C_2$ ,  
it must be the case that  $L \in \varphi_3$ , so, also in this case, no new deduction can be performed.
- Finally, suppose that  $C$  is any other axiom.  $C$  cannot be an instance of a reflexivity axiom (1 or 4), because it would be unitary and positive, and it cannot be an instance of any irreflexivity axiom (7 or 11) because  $C_1$  would not be admissible.  $C$  cannot be the instance of any symmetry axiom (2, 8, or 10), because this would entail  $q = 1$ , which is to say  $C_1$  would suffice for a deduction, but if  $C_1$  belongs to some formula  $\varphi$ , then the latter must also contain  $L$  because it is explicit. Finally, if  $C$  is an instance of some antisymmetry axiom (5 or 12), then both  $C_1$  and  $C_2$  must refer to the same two endpoints as  $C$ , that is, they must belong to the same formula  $\varphi$ ; therefore, since  $\varphi$  is explicit,  $L$  already belongs to  $\varphi$ , and if it is the instance of some weakening axiom (13, 14, or 15), or the instance of some compatibility axiom (16, 17, or 18), then  $C_1$  must refer to the same two endpoints as  $C$ , and the same argument applies.

Therefore, no deduction can be performed on the translation of a path consistent network. ◀

# Temporal Logic with Recursion

**Florian Bruse**

School of Electrical Engineering and Computer Science, University of Kassel, Germany  
florian.bruse@uni-kassel.de

**Martin Lange**

School of Electrical Engineering and Computer Science, University of Kassel, Germany  
martin.lange@uni-kassel.de

---

## Abstract

---

We introduce extensions of the standard temporal logics CTL and LTL with a recursion operator that takes propositional arguments. Unlike other proposals for modal fixpoint logics of high expressive power, we obtain logics that retain some of the appealing pragmatic advantages of CTL and LTL, yet have expressive power beyond that of the modal  $\mu$ -calculus or MSO. We advocate these logics by showing how the recursion operator can be used to express interesting non-regular properties. We also study decidability and complexity issues of the standard decision problems.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Modal and temporal logics; Theory of computation  $\rightarrow$  Program specifications

**Keywords and phrases** formal specification, temporal logic, expressive power

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.6

## 1 Introduction

Temporal logic is a well-established formalism for the specification of the behaviour of dynamic systems, typically separated into two classes: linear-time vs. branching-time, reflecting the philosophical question of whether the future is determined or not [31]. There is a direct correspondence in computer science: the linear-time view regards programs as being stand-alone with input given at the beginning and a computation running without further interaction with the program's environment; in the branching-time view programs may be reactive, i.e. able to react to input as it occurs during a computation. The most prominent member of the linear-time family is LTL [29], the most prominent members of the branching-time family are CTL [8] and CTL\* [10].

The classification into linear-time and branching-time typically has consequences with regards to expressiveness and computational complexity of the two major decision problems: satisfiability and model checking. For genuine linear-time logics, these two are closely related as model checking is a generalisation of validity checking, and validity checking can express model checking of finite-state systems. For LTL, these problems are PSPACE-complete [30]. The picture for branching-time logics is different: here, model checking is typically easier than satisfiability checking, for instance P- vs. EXPTIME-complete for CTL [9] and PSPACE- vs. 2EXPTIME-complete for CTL\* [10, 11, 33].

Various extensions of these logics have been investigated for purposes of higher expressiveness: there are “semantic” extensions like action-based [21], dynamic logic [12], real-time [1], metric [18] and probabilistic temporal logics [13] for instance, as well as combinations thereof. Then there are “syntactic” extensions like the modal  $\mu$ -calculus ( $\mathcal{L}_\mu$ ) [19] with its explicit least and greatest fixpoint operators. It extends the expressive power to full regularity, i.e. that of Monadic Second-Order Logic [16] (up to bisimilarity).

Extensions in expressive power beyond that are possible, and sometimes even necessary for particular purposes. For instance,  $\mathcal{L}_{\mu-}$  and therefore temporal logics embeddable into it – have the finite model property [20]. Hence, they cannot be used to reason about inherent



© Florian Bruse and Martin Lange;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 6; pp. 6:1–6:14

Leibniz International Proceedings in Informatics



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

properties of infinite-state systems. Such an observation has led to the investigation of Propositional Dynamic Logic of Non-Regular Programs (PDL[CFL]) [15] for instance. It uses context-free instead of regular languages as in ordinary PDL and, thus, can express some non-regular properties but not all regular ones. When restricted to visibly pushdown languages (PDL[VPL]), it even becomes decidable [26]. Other extensions include Fixpoint Logic with Chop (FLC) [28], integrating process-algebraic operations into the modal  $\mu$ -calculus, or Higher-Order Fixpoint Logic (HFL) [34] which incorporates a simply-typed  $\lambda$ -calculus into  $\mathcal{L}_\mu$ .

High expressive power also comes at a high price in two regards. First, it is tightly linked to high computational complexity including undecidability. In fact, the satisfiability checking problems for all the logics mentioned above here, capable of expressing non-regular properties, are (highly) undecidable. The second downside concerns pragmatics. Already  $\mathcal{L}_\mu$  is commonly seen as unsuitable for a non-expert as writing temporal properties using least and greatest fixpoints is cumbersome and error-prone. This holds even more so for extensions like FLC and HFL.

We introduce extensions of LTL and CTL in order to make the formal specification of non-regular properties more widely available through temporal logics with a more intuitive syntax. We introduce a recursion operator, resulting the logics Recursive LTL, resp. CTL (RecLTL, RecCTL). The standard temporal operators from LTL and CTL are preserved even though the recursion operator is expressive enough to mimic these. Note that temporal operators like Until can be seen as abbreviations of infinite Boolean combinations of basic propositional and modal formulas. The recursion operator can be used to construct more complex infinite Boolean connections and, thus, express interesting properties, including non-regular ones. Semantically, it is defined via least fixpoints of monotone functions of order 1 over the powerset lattice of the underlying labelled transition systems. The model-theoretic design of RecLTL and RecCTL is inspired by the machinery underlying a complex logic like HFL, yet their pragmatics aims at more understandable and usable temporal specification languages.

In Sect. 2 we introduce RecCTL and RecLTL formally but also try to build intuition about what they can be used for and how to use them. In Sect. 3 we study the expressive power of RecCTL and RecLTL formally by placing them into the hierarchy of the those logics mentioned above. In Sect. 4 we show that model checking RecCTL is EXPTIME-complete whereas model checking RecLTL and satisfiability checking for both is undecidable. We also present a decidable fragment of RecCTL. The paper concludes with remarks on further work in Sec. 5.

## 2 Designing Temporal Logics of Higher Expressiveness

We assume familiarity with the standard temporal logics CTL and LTL.

**Labelled Transition Systems.** Let  $\mathcal{P}$  be a finite set of atomic propositions. A labeled transition system (LTS) is a tuple  $\mathcal{T} = (\mathcal{S}, \rightarrow, \ell)$  where  $\mathcal{S}$  is a (potentially infinite) set of states,  $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$  is the transition relation that is assumed to be total in the sense that for every  $s \in \mathcal{S}$  there is a  $t \in \mathcal{S}$  s.t.  $s \rightarrow t$ . Finally,  $\ell : \mathcal{S} \rightarrow 2^{\mathcal{P}}$  labels each state with the set of propositions that hold at this state. A *path* is an infinite sequence  $\pi = s_0, s_1, \dots$  s.t.  $s_i \rightarrow s_{i+1}$  for all  $i \geq 0$ . We write  $\pi(i)$  to denote the  $i$ th state on this path.



**Infinitary modal logic.** Infinitary modal logic adds infinitary junctors  $\bigwedge_{i \in I}$ , resp.  $\bigvee_{i \in I}$  for arbitrary sets  $I$  to modal logic with the operators  $\diamond$  and  $\square$  or, **EX** and **AX** as they are usually written in the temporal setting. Hence, formulas can become infinitely wide, but every path in the syntax tree is still of finite length.

Over any set of LTS, any standard propositional temporal logic can be translated into infinitary modal logic. For instance, the CTL formula **EF** $q$  expressing reachability of a  $q$ -state is equivalent to  $\varphi_1 := \bigvee_{i \geq 0} \mathbf{EX}^i q$ , even uniformly over the class of all LTS.

Not every infinitary modal formula corresponds to a (finite) temporal formula, though. For instance,  $\varphi_2 := \bigvee_{i \geq 0} \mathbf{AX}^i q$  is not expressible in CTL, or even the modal  $\mu$ -calculus [7].

**Patterns of infinitary formulas expressible in temporal logics.** The reason for the fact that  $\varphi_1$  is expressible in CTL but  $\varphi_2$  is not, is given by the interplay of two principles:

First, the operator **EX** commutes with disjunctions –  $\mathbf{EX}\varphi \vee \mathbf{EX}\psi \equiv \mathbf{EX}(\varphi \vee \psi)$  – but **AX** does not. Hence, we have

$$\varphi_1 = \bigvee_{i \geq 0} \mathbf{EX}^i q = q \vee \bigvee_{i \geq 1} \mathbf{EX}^i q \equiv q \vee \mathbf{EX} \bigvee_{i \geq 0} \mathbf{EX}^i q = q \vee \mathbf{EX}\varphi_1 . \quad (1)$$

Second, the fixpoint operators in CTL, LTL and  $\mathcal{L}_\mu$  are propositional, i.e. (monadic) second-order. In other words, they can only be used to define a least or greatest fixpoint recursion over an operator that is a modal formula itself (disregarding nested fixpoints for the moment). Eq. 1 shows that  $\varphi_1 \equiv \mu Z. q \vee \mathbf{EX} Z$  in  $\mathcal{L}_\mu$  terms with  $q \vee \mathbf{EX} Z$  for a propositional variable  $Z$  describing the evaluation in each iteration of the fixpoint recursion.

Now consider  $\varphi_2$  again. It is simply not possible to rewrite it in a way like  $\varphi_1$ , obtaining an  $\mathcal{L}_\mu$  formula because  $\mathbf{AX}\varphi \vee \mathbf{AX}\psi \not\equiv \mathbf{AX}(\varphi \vee \psi)$ . Most importantly,  $\varphi_2 \not\equiv \mu Z. q \vee \mathbf{AX} Z$ .

This is not to say that the infinitary modal formula representing  $\varphi_2$  could not be built up in a recursive way using fixpoints, as it is equivalent to the FLC formula  $(\mu Z. \tau \vee (Z; \mathbf{AX})); q$ . The syntax is not easily understood – see the literature on FLC for a detailed introduction [28, 23] – especially with **AX** becoming a 0-ary operator. The main point to observe here, though, is the structure of the formula  $\tau \vee (Z; \mathbf{AX})$  defining the fixpoint iteration using a recursion anchor  $Z$ . It is, in a sense, *left-linear* as opposed to the *right-linear* recursion schemes definable in  $\mathcal{L}_\mu$ .

**Adding recursion to temporal logics.** The aim of this paper is to design expressive extensions of CTL and LTL that retain their nice pragmatic features, in particular an intuitive and readable syntax. Likewise, the semantics needs to be – at the same time – mathematically sound. We aim for a moderate increase in expressive power in the sense of the example above: on top of standard CTL and LTL, it should be possible to express certain additional patterns of infinitary modal formulas, for instance infinitary disjunctions over uniform families of **AX**-formulas.

As an example, recall that two paths  $\pi = s_0, s_1, \dots$  and  $\pi' = s'_0, s'_1, \dots$  of an LTS  $\mathcal{T} = (\mathcal{S}, \rightarrow, \ell)$  are called *trace-equivalent* iff  $\ell(s_i) = \ell(s'_i)$  for all  $i \geq 0$ . The existence of two non-trace-equivalent paths can be expressed in infinitary modal logic as

$$\varphi_{\text{nonlin}} := \bigvee_{p \in \mathcal{P}} \bigvee_{i \geq 0} \mathbf{EX}^i p \wedge \mathbf{EX}^i \neg p$$

In other words,  $\neg \varphi_{\text{nonlin}}$  states that all paths beginning in the state of evaluation, are trace-equivalent or, equivalently, that the LTS (from that state) is bisimilar to a word.

## 6:4 Temporal Logic with Recursion

Note that in  $\varphi_{\text{nonlin}}$ , the EX-operators are “guarded” by a conjunction. It is therefore not possible to rewrite this formula into a least fixpoint recursion of  $\mathcal{L}_\mu$  in the style of Eq. 1. In order to capture such patterns of infinitary modal logic, a temporal logic would have to provide operators which allow the build-up of modal formulas “behind” the recursion anchor, similar to the FLC formula equivalent to  $\varphi_2$  above.

An appropriate mechanism for creating such effects can be borrowed from HFL: it lifts the recursion anchor from the propositional level to a higher one. In HFL, this can be a function of arbitrary order; here we restrict ourselves to order 1 to keep the resulting logic reasonably simple. The recursion anchor then becomes a function whose arguments are temporal formulas. In order to manipulate the arguments, we use propositional variables as symbolic names for the argument values in each recursive call.

Thus, our recursive temporal logic should have a recursion operator which defines a recursion anchor in the form of a variable, say  $\mathcal{F}$ . At the same time, it needs to provide symbolic names for some arguments to  $\mathcal{F}$ , say  $x_1, \dots, x_n$ , and then the definition of the recursion can use these as standard temporal formulas, as well as  $\mathcal{F}$  applied to  $n$  formulas. Likewise, the entire recursion formula would have to be applied to  $n$  propositional formulas, which are just the initial arguments for the recursive iteration. Such a formula could look like

$$\varphi_p := \underbrace{(\text{rec } \mathcal{F}(y, z).(y \wedge z) \vee \mathcal{F}(\text{EX}y, \text{EX}z))}_{\Psi}(p, \neg p) .$$

From a pragmatic point of view, recursion can be read in a natural way using unfolding and replacement of symbolic argument variables, i.e. using  $\beta$ -reduction. Here, we would get

$$\begin{aligned} \varphi_p &\equiv (p \wedge \neg p) \vee \Psi(\text{EX}p, \text{EX}\neg p) \equiv (p \wedge \neg p) \vee (\text{EX}p \wedge \text{EX}\neg p) \vee \Psi(\text{EXEX}p, \text{EXEX}\neg p) \\ &\equiv \dots \equiv \bigvee_{i \geq 0} \text{EX}^i p \wedge \text{EX}^i \neg p . \end{aligned}$$

In other words, such an extension of CTL (with appropriately defined semantics) would be able to express  $\varphi_{\text{nonlin}}$  via  $\bigvee_{p \in \mathcal{P}} \varphi_p$ .

**The formal syntax.** Let  $\mathcal{P} = \{p, q, \dots\}$  be a finite set of atomic propositions,  $\mathcal{V}_1 = \{x, y, \dots\}$  and  $\mathcal{V}_2 = \{\mathcal{F}, \mathcal{G}, \dots\}$  be sets of *propositional*, resp. *recursion* variables. Formulas of *Recursive CTL* (RecCTL) are given by the grammar

$$\begin{aligned} \varphi, \psi &::= p \mid x \mid \varphi \wedge \psi \mid \neg \varphi \mid \text{EX}\varphi \mid \text{E}(\varphi \cup \psi) \mid \Phi(\varphi, \dots, \varphi) \\ \Phi &::= \mathcal{F} \mid \text{rec } \mathcal{F}(x_1, \dots, x_k).\varphi \end{aligned}$$

where  $p \in \mathcal{P}$ ,  $k \geq 0$ ,  $x, x_1, \dots, x_k \in \mathcal{V}_1$  and  $\mathcal{F} \in \mathcal{V}_2$ . The formulas derived from  $\varphi, \psi$  are called *propositional*, those derived from  $\Phi$  are called *first-order*.

Other Boolean and temporal operators are defined in the usual way, for instance  $\text{AX}\varphi := \neg \text{EX}\neg\varphi$ ,  $\text{EF}\varphi := \text{E}(\text{tt} \cup \varphi)$ ,  $\text{AG}\varphi := \neg \text{EF}\neg\varphi$ , and will be used freely henceforth. The notions of a subformula, a free or bound occurrence of a variable are the usual ones.

The semantics of the recursion operator will be explained later on using least fixpoints in complete function lattices. This makes the Bekiç Lemma [5] 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. [3] for its use in  $\mathcal{L}_\mu$ , is a

$$\text{rec } i \left( \begin{array}{cc} \mathcal{F}_1(x_1, \dots, x_{k_1}) & \cdot \varphi_1 \\ & \vdots \\ \mathcal{F}_n(x_1, \dots, x_{k_n}) & \cdot \varphi_n \end{array} \right) (\psi_1, \dots, \psi_k)$$



s.t.  $1 \leq i \leq n$  and  $k = k_i$ . Informally, this defines not just one but several functions  $\mathcal{F}_1, \dots, \mathcal{F}_n$  which may all depend on each other in a mutually recursive way formalised in the  $\varphi_j$ 's. In the end, the function named by  $\mathcal{F}_i$  is applied to the initial arguments  $\psi_1, \dots, \psi_k$ .

We will also write  $\text{fun}(x_1, \dots, x_k).\varphi$  instead of  $\text{rec } \mathcal{F}(x_1, \dots, x_k).\varphi$  when  $\mathcal{F}$  does not occur in  $\varphi$ .

**Well-formed formulas.** Clearly, not every formula generated by the formal grammar above is meaningful. For instance, in  $(\text{rec } \mathcal{F}(x).x \wedge \mathcal{F}(\text{EX}x))(p, \neg p)$  the number of formal parameters of  $\mathcal{F}$  does not match the number of given parameters. Such mistakes are easy to spot; henceforth, we assume that all formulas are well-formed in this respect.

However, further restrictions need to be imposed before a formal semantics can be given, since the addition of the recursion operator requires careful handling of negations.

Consider  $\varphi_p$  from above. Its subformula  $\Psi$  can be seen as a function mapping a pair of propositions to a proposition. When interpreted over an LTS with state set  $\mathcal{S}$ , such a function is an object of type  $2^{\mathcal{S}} \times 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}} =: M_{1,2}$  (functions of order 1 with 2 arguments). Note that  $M_{1,2}$  is a complete lattice when equipped with the point-wise order where  $f \leq g$  iff  $f(x, y) \leq g(x, y)$  for all  $x, y \in 2^{\mathcal{S}}$ . This also is true when restricted to just monotonic functions from  $M_{1,2}$ . Since recursion should be explained using least fixpoints, we are interested in the function  $f: M_{1,2} \rightarrow M_{1,2}$  that maps a (monotonic) function  $\mathcal{F}: M_{1,2}$  to the function  $(y, z) \mapsto (y \cap z) \cup \mathcal{F}(\text{EX}y, \text{EX}z): M_{1,2}$ . All operators used here are monotonic in the usual powerset lattice  $2^{\mathcal{S}}$  and, hence, if  $\mathcal{F}$  is monotonic, so is  $f$ . Thus, the Knaster-Tarski Theorem [32, 17] yields that  $f$  has a least fixpoint  $F$ , which is a natural candidate for the semantics of  $\Psi$ .

Having negation in a specification logic is desirable, yet negation is clearly not a monotonic operator. This is not problematic in CTL and LTL, where negation can only occur in places where the implicit recursion in e.g. the operator  $\text{U}$  is not affected by negation in one of its arguments. Already  $\mathcal{L}_\mu$  does not allow unrestricted use of negation, since  $\mu X. \neg X$  for instance cannot be given a proper semantics. The solution is to restrict the syntax to only allow negation in front of non-variable atoms, or to require that recursion variables only occur under an even number of negations in the defining fixpoint formula.

The first way is also viable mathematically here, but it would restrict the pragmatics of this logic strongly since one may often want to specify undesired properties, i.e. use negation on top level for instance. Hence, we aim for a syntactic criterion that allows negation to be used as freely as possible. Unfortunately, the comparatively simple rule used in  $\mathcal{L}_\mu$  does not generalise so easily.

Consider the toy example  $(\text{rec } \mathcal{F}(x).(\text{fun } y.\neg y)(\mathcal{F}(x)))p$ . It appears to be harmless, since negation occurs only in front of the propositional variable  $y$  but not any recursion variable. However, unfolding and  $\beta$ -reducing this to  $(\text{rec } \mathcal{F}(x).\neg \mathcal{F}(x))p$  reveals that the negation in front of  $\mathcal{F}$  was simply hidden away in the anonymous function  $\text{fun } y.\neg y$  which is clearly not monotonic.

While the use of this antitonic function violates the monotonicity requirement of the function defined by  $\mathcal{F}$ , functions that are antitonic in one of their arguments are not necessarily problematic in general. In fact, much of the expressive power of RecCTL and ReLTL would be lost if antitonic functions were forbidden in general. Consider

$$\varphi_{\text{unbound}} := \neg(\text{rec } \mathcal{F}(x, y).(x \wedge \neg y) \vee \mathcal{F}(\text{EX}x, \text{EX}y))(p, \text{EX}p)$$

stating that there is no  $n$  such that  $p$  can be reached in  $n$  steps but not in  $n + 1$ . Clearly, the function  $\mathcal{F}$  is antitonic in its second argument. However, the function  $\mathcal{F} \mapsto ((x, y) \mapsto (x \cap \bar{y}) \cup \mathcal{F}(\text{EX}x, \text{EX}y))$  is indeed monotonic in  $\mathcal{F}$  and therefore has a least fixpoint which is

why the recursion in  $\varphi_{\text{unbound}}$  is well-defined in this case. In this instance, we make use of the fact that  $M_{1,2}$  stays a complete lattice when restricted to functions that are monotonic in their first argument, and antitonic in the second argument, whence any monotonic function that maps a function from this sublattice back into the sublattice has a least fixpoint. In fact, this holds for all  $M_{1,j}$  and all partitions of the arguments into monotonic and antitonic.

In the following we will devise a syntactic criterion that allows antitonic functions to be used in a harmless way, i.e. such that a formal semantics can still be given via least fixpoints. The criterion is necessarily more complex than the  $\mathcal{L}_\mu$  one about occurrence under an even number of negations as seen above; on the other hand we also do not require an entire type system as it is the case in HFL.

We will call a formula well-formed if, in addition to the constraint on matching numbers of arguments, it is possible to separate each list of formal and given parameters into two parts of monotonically and antitonically used arguments, such that based on this separation we can establish that the former ones are only used positively and the latter ones are only used negatively. In the following we will make the meaning of this precise. For the moment, suppose that recursive definitions and calls are written as

$$(\text{rec } \mathcal{F}(x_1, \dots, x_k \mid y_1, \dots, y_{k'}) \cdot \varphi) \quad \text{resp.} \quad \mathcal{F}(\varphi_1, \dots, \varphi_k \mid \psi_1, \dots, \psi_{k'}) .$$

Either part of such a list can also be empty. For example, we would write  $\varphi_{\text{unbound}}$  as

$$\neg(\text{rec } \mathcal{F}(x \mid y) \cdot (x \wedge \neg y) \vee \mathcal{F}(\text{EX}x \mid \text{EX}y))(p \mid \text{EX}p)$$

declaring, in particular,  $x$  to be used monotonically and  $y$  to be used antitonically. The other possibilities for separating the arguments would not pass the following check about positive, resp. negative use. We say that  $x \in \mathcal{V}_1$  is used positively in  $x$  and  $\mathcal{F} \in \mathcal{V}_2$  is used positively in  $\mathcal{F}(x_1, \dots, x_k \mid y_1, \dots, y_{k'})$ . Moreover,  $x$ , or  $\mathcal{F}$  is used

- positively, resp. negatively in  $\varphi_1 \wedge \varphi_2$ ,  $\text{EX}\varphi_1$ ,  $\text{E}(\varphi_1 \text{U}\varphi_2)$  or  $\text{rec } \mathcal{F}(x_1, \dots, x_k \mid y_1, \dots, y_{k'}) \cdot \varphi_1$ , if it is used positively, resp. negatively in  $\varphi_1$  or  $\varphi_2$ ;
- positively, resp. negatively in  $\neg\varphi$  if it is used negatively, resp. positively in  $\varphi$ ;
- positively in  $\mathcal{G}(\varphi_1, \dots, \varphi_k \mid \psi_1, \dots, \psi_{k'})$  or  $(\text{rec } \mathcal{G}(\dots) \cdot \varphi)(\varphi_1, \dots, \varphi_k \mid \psi_1, \dots, \psi_{k'})$  if it is used positively in one of the  $\varphi_i$ , or negatively in one of the  $\psi_i$ ;
- negatively in  $\mathcal{G}(\varphi_1, \dots, \varphi_k \mid \psi_1, \dots, \psi_{k'})$  or  $(\text{rec } \mathcal{G}(\dots) \cdot \varphi)(\varphi_1, \dots, \varphi_k \mid \psi_1, \dots, \psi_{k'})$  if it is used negatively in one of the  $\varphi_i$ , or positively in one of the  $\psi_i$ .

Intuitively, the polarity of use switches at an actual negation, and when the subformula in question is an argument right of the separator in a recursive call. Having defined a notion of positive and negative occurrences, we can restrict the syntax accordingly to ensure that the following semantics will be well-defined. Note that  $x$  or  $\mathcal{F}$  can be used both positively and negatively in a formula.

► **Definition 1.** A formula of RecCTL is called well-formed if it is possible to separate the formal and given arguments of recursive definitions and calls into two lists each such that the following conditions hold.

- If  $(\text{rec } \mathcal{F}(x_1, \dots, x_k \mid y_1, \dots, y_{k'}) \cdot \varphi)(\varphi_1, \dots, \varphi_l \mid \psi_1, \dots, \psi_{l'})$  is a recursive definition, then  $k = l$  and  $k' = l'$ , and if  $\mathcal{F}(\varphi_1, \dots, \varphi_m \mid \psi_1, \dots, \psi_{m'})$  is a recursive call of the same variable, then  $k = m$  and  $k' = m'$ ,
- If  $\text{rec } \mathcal{F}(x_1, \dots, x_k \mid y_1, \dots, y_{k'}) \cdot \varphi$  is a recursive definition then none of the  $x_i$  is used negatively in  $\varphi$ , and none of the  $y_i$  is used positively in  $\varphi$ , and
- If  $\text{rec } \mathcal{F}(x_1, \dots, x_k \mid y_1, \dots, y_{k'}) \cdot \varphi$  is a recursive definition then  $\mathcal{F}$  is not used negatively in  $\varphi$ .

**The formal semantics.** Let  $\mathcal{T} = (\mathcal{S}, \rightarrow, \ell)$  be an LTS, let  $\varphi_0$  be a well-formed formula of RecCTL. An environment is a function from  $\eta: \mathcal{V}_1 \cup \mathcal{V}_2 \rightarrow 2^{\mathcal{S}} \cup \bigcup_{j \geq 0} M_{1,j}$  where the value of a variable matches its type in  $\varphi_0$  and the declared monotonicity, resp. antitonicity of its arguments. Here  $M_{1,j}$  is the space of order-1 functions with  $j$  arguments.

The formal semantics assigns a proposition, i.e. set of states to each propositional subformula  $\varphi$  of  $\varphi_0$ , and a first-order function to each first-order subformula  $\Phi$  of  $\varphi$  of as follows.

$$\begin{aligned}
\llbracket p \rrbracket_{\eta}^{\mathcal{T}} &= \{s \in \mathcal{S} \mid p \in \ell(s)\} \\
\llbracket x \rrbracket_{\eta}^{\mathcal{T}} &= \eta(x) \\
\llbracket \varphi \wedge \psi \rrbracket_{\eta}^{\mathcal{T}} &= \llbracket \varphi \rrbracket_{\eta}^{\mathcal{T}} \cap \llbracket \psi \rrbracket_{\eta}^{\mathcal{T}} \\
\llbracket \neg \varphi \rrbracket_{\eta}^{\mathcal{T}} &= \mathcal{S} \setminus \llbracket \varphi \rrbracket_{\eta}^{\mathcal{T}} \\
\llbracket \text{EX} \varphi \rrbracket_{\eta}^{\mathcal{T}} &= \{s \in \mathcal{S} \mid \text{there exists } t \in \llbracket \varphi \rrbracket_{\eta}^{\mathcal{T}} \text{ such that } s \rightarrow t\} \\
\llbracket \text{E}(\varphi \text{ U } \psi) \rrbracket_{\eta}^{\mathcal{T}} &= \{s \in \mathcal{S} \mid \text{there exists path } \pi, \text{ integer } i \text{ such that} \\
&\quad s = \pi(0), \pi(i) \in \llbracket \psi \rrbracket_{\eta}^{\mathcal{T}} \text{ and } \pi(j) \in \llbracket \varphi \rrbracket_{\eta}^{\mathcal{T}} \text{ for all } 0 \leq j < i\} \\
\llbracket \mathcal{F} \rrbracket_{\eta}^{\mathcal{T}} &= \eta(\mathcal{F}) \\
\llbracket \text{rec } \mathcal{F}(x_1, \dots, x_k). \varphi \rrbracket_{\eta}^{\mathcal{T}} &= \text{LFP } f \mapsto ((S_1, \dots, S_k) \mapsto \llbracket \varphi \rrbracket_{\eta[\mathcal{F} \mapsto f, x_1 \mapsto S_1, \dots, x_k \mapsto S_k]}^{\mathcal{T}}) \\
\llbracket \Phi(\varphi_1, \dots, \varphi_k) \rrbracket_{\eta}^{\mathcal{T}} &= \llbracket \Phi \rrbracket_{\eta}^{\mathcal{T}}(\llbracket \varphi_1 \rrbracket_{\eta}^{\mathcal{T}}, \dots, \llbracket \varphi_k \rrbracket_{\eta}^{\mathcal{T}})
\end{aligned}$$

Notions like satisfaction ( $\mathcal{T}, s \models \varphi$ ), satisfiability and equivalence ( $\varphi \equiv \psi$ ) are defined as usual.

The following lemma states that this semantics is well-defined, in particular, that least fixpoints as used in the second to last clause do indeed exist. This is guaranteed by well-formedness of  $\varphi_0$  which in turn guarantees monotonicity of the function whose least fixpoint is used to give meaning to the recursion operator in the penultimate clause.

► **Lemma 2.** *Let  $\varphi$  be a well-formed RecCTL sentence, let  $\eta$  be an environment and let  $\mathcal{T}$  be an LTS. Then  $\llbracket \psi \rrbracket_{\eta}^{\mathcal{T}}$  is well-defined.*

Since  $\llbracket \varphi \rrbracket_{\eta}^{\mathcal{T}}$  does not depend on  $\eta$ , we simply write  $\llbracket \varphi \rrbracket^{\mathcal{T}}$  and drop the environment. The lemma's proof is purely technical but standard by induction on the structure of  $\varphi$ .

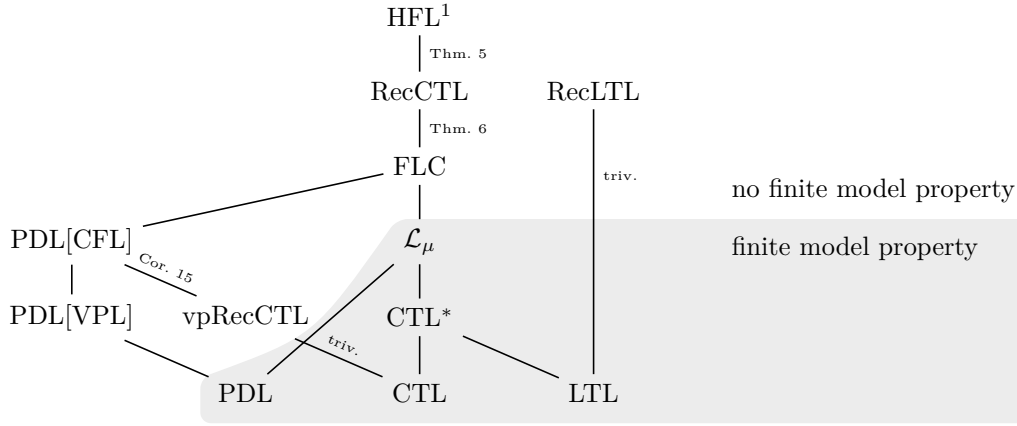
We also state a fundamental equivalence which is very helpful for understanding formulas. The proof is simply by combining the well-known equivalence-preserving principles of fixpoint unfolding and  $\beta$ -reduction, and is therefore omitted. As usual,  $\varphi[\psi_1/x_1, \dots, \psi_k/x_k]$  denotes the simultaneous replacement of every free occurrence of  $x_i$  by  $\psi_i$ .

► **Lemma 3.** *For any  $\varphi, \psi_1, \dots, \psi_k$  we have*

$$(\text{rec } \mathcal{F}(x_1, \dots, x_k). \varphi)(\psi_1, \dots, \psi_k) \equiv \varphi[\psi_1/x_1, \dots, \psi_k/x_k, \text{rec } \mathcal{F}(x_1, \dots, x_k). \varphi / \mathcal{F}]$$

**The linear-time case.** RecLTL is obtained from RecCTL syntactically by removing the path quantifiers **E** and **A** just like the syntax of LTL can be obtained from CTL in this way. A RecLTL formula is interpreted over a linear-time structure  $\pi$ , i.e. a transition system with a single path only. The semantics is defined in the same way as for RecCTL. Given a RecLTL formula  $\varphi$ , an LTS  $\mathcal{T}$  with state  $s$ , and a path  $\pi$ , we write  $\pi \models \varphi$  to denote that the path  $\pi$  satisfies  $\varphi$ . We write  $\mathcal{T}, s \models \varphi$  iff all paths starting in  $s$  satisfy  $\varphi$ . In other words, the usual for-all-paths semantics for linear-time formulas can also be applied to the richer language RecLTL.

It is not hard to see that Lemmas 2 and 3 as well as previously worked out concepts like well-formedness etc. hold for RecLTL as well.



■ **Figure 1** Placing RecCTL and RecLTL into the hierarchy of temporal logics w.r.t. expressive power.

### 3 The Power of Recursion in Temporal Logics

Recall the RecCTL example  $\varphi_{\text{nonlin}}$  above and remember that  $\neg\varphi_{\text{nonlin}}$  states that all paths emerging from the state under consideration are trace-equivalent, i.e. indistinguishable through the sequence of their propositional labels. This gives an easy satisfiability-preserving reduction from RecLTL into RecCTL.

► **Theorem 4.** *For every  $\varphi \in \text{RecLTL}$  there is an equi-satisfiable  $\varphi' \in \text{RecCTL}$  such that  $|\varphi'| = \mathcal{O}(|\varphi|)$ .*

**Proof.** Simply take  $\varphi' := \widehat{\varphi} \wedge \neg\varphi_{\text{nonlin}}$  where  $\widehat{\varphi}$  results from  $\varphi$  by replacing every subformula of the form  $X\psi$  with  $AX\psi$ . The second conjunct requires a model of  $\varphi'$  to only have trace-equivalent paths which are of infinite length by assumption. Thus, if  $\varphi$  has a model  $\pi$  then this is clearly a model of  $\varphi'$ . Moreover, any path of an LTS model for  $\varphi'$  is a model for  $\varphi$ . ◀

Next we place RecCTL and RecLTL into the expressiveness hierarchy of well-known (and some lesser known) temporal and modal fixpoint logics. Note that the models under consideration here are transition systems without edge labels, as they are usually used for temporal logics like CTL and LTL. The results of this section are presented for this class of structures, even though logics like  $\mathcal{L}_\mu$  are typically interpreted over the richer class of transition system *with* edge labels. The results can easily be extended to this richer class, provided that the syntax of RecCTL and RecLTL is extended to speak about  $a$ -successors rather than just successors, for example by replacing EX with  $EX_a$  for any edge label.

This hierarchy is shown in Fig. 1. It contains the standard temporal logics CTL and LTL as fragments of CTL\*, which in turn is known to be embeddable into  $\mathcal{L}_\mu$ . Above, there are the expressive logics mentioned in the introduction, namely

- *Fixpoint Logic with Chop* (FLC): it interprets every formula as a predicate transformer mapping a set of states to a set of states of an LTS. Predicates, the basic semantic objects of  $\mathcal{L}_\mu$ , can be seen as constant predicate transformers which explains why FLC extends  $\mathcal{L}_\mu$  [28].
- *Higher-Order Fixpoint Logic* (HFL): it allows functions of arbitrary higher-order to be built from modal and Boolean operators as well as fixpoints. Its fragment  $\text{HFL}^1$  is obtained by restricting all functions to first order. This includes predicate transformers as they can be seen as *unary* functions of order 1. Hence, FLC is embeddable into  $\text{HFL}^1$  [34].

The graph also includes the dynamic logic PDL as it is closely related to CTL and its non-regular extension PDL[CFL], in order to complete the picture of expressiveness of temporal logics, in particular to give a better feeling for the power of RecCTL and RecLTL. The embedding of PDL[CFL] into FLC was shown in [25]. At last, it includes the fragment vpRecCTL of RecCTL which will be discussed in Sect. 4 in the context of decidability questions.

The picture also draws the distinguishing line of regularity vs. non-regularity in terms of possessing the finite model property (FMP). It is lost for all of these logics that are not embeddable into  $\mathcal{L}_\mu$  [23].

RecCTL can be placed between FLC and HFL<sup>1</sup>. We refrain from presenting the full (and sometimes cumbersome) syntax and semantics of these two logics here. Instead we refer to the existing literature for full details [28, 34] and only give the main ideas here.

► **Theorem 5.** *Every RecCTL formula can equivalently be expressed in HFL<sup>1</sup>.*

**Proof.** (Sketch) Well-formed formulas can straight-forwardly be translated. The only interesting case is that of a recursive first-order formula ( $\text{rec } \mathcal{F}(x_1, \dots, x_k \mid y_1, \dots, y_{k'}) \cdot \varphi$ ). It is equivalent to the HFL<sup>1</sup> formula  $\mu \mathcal{F}^\tau \cdot \lambda x_1^\bullet \dots \lambda x_k^\bullet \cdot \varphi$  with type annotation  $\tau = \bullet^+ \rightarrow \dots \rightarrow \bullet^+ \rightarrow \bullet^- \rightarrow \dots \rightarrow \bullet^- \rightarrow \bullet$ . ◀

For the lower bound we need to quickly recall FLC. Its formulas are built from basic literals  $p$ ,  $\neg p$  and the modal  $\diamond$  and  $\square$ . Note that they receive no fixed argument, as they are being interpreted not as predicates but as predicate transformers, i.e. a function of type  $2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$  over a state space  $\mathcal{S}$  of some LTS.

The syntax has conjunctions and disjunctions and a *chop* operator “;” which is interpreted as the functional composition of two predicate transformers, and another atomic formula  $\tau$  which is interpreted as the neutral element to composition, i.e. the identity predicate transformer. On top of this, fixpoint quantifiers are added which are interpreted as fixpoints in the complete lattice of pointwise-ordered predicate transformers.

► **Theorem 6.** *Every FLC formula can equivalently be expressed in RecCTL.*

**Proof.** We devise a translation  $\widehat{\cdot} : \text{FLC} \rightarrow \text{RecCTL}$  that preserves equivalence using, for every FLC fixpoint variable  $X$ , a unique recursion variable  $\mathcal{F}_X$ .

$$\begin{aligned} \widehat{p} &:= \text{fun } x.p & \widehat{\varphi \vee \psi} &:= \text{fun } x.\widehat{\varphi}(x) \vee \widehat{\psi}(x) & \widehat{\diamond} &:= \text{fun } x.EXx \\ \widehat{\neg p} &:= \text{fun } x.\neg p & \widehat{\varphi \wedge \psi} &:= \text{fun } x.\widehat{\varphi}(x) \wedge \widehat{\psi}(x) & \widehat{\square} &:= \text{fun } x.AXx \\ \widehat{\tau} &:= \text{fun } x.x & \widehat{\varphi; \psi} &:= \text{fun } x.\widehat{\varphi}(\widehat{\psi}(x)) \\ \widehat{X} &:= \mathcal{F}_X & \widehat{\mu X.\varphi} &:= \text{rec } \mathcal{F}_X(y).\widehat{\varphi}(y) & \widehat{\nu X.\varphi} &:= \neg \text{rec } \mathcal{F}_X(y).\neg \widehat{\varphi}[\neg \mathcal{F}_X/\mathcal{F}_X](y) \end{aligned}$$

where  $[\neg \mathcal{F}_X/\mathcal{F}_X]$  denotes the substitution of any  $(\mathcal{F}_X(y).\psi)(\psi')$  with  $\neg((\mathcal{F}_X(y).\psi)(\psi'))$ .

A straight-forward induction on the structure of an FLC formula  $\varphi$  shows that  $\widehat{\varphi}$  denotes the same predicate transformer as  $\varphi$  under any variable assignment that maps  $X$  and  $\mathcal{F}_X$  to the same predicate transformer. We then get that the FLC formula  $\varphi$  is equivalent to the RecCTL formula  $\widehat{\varphi}(\text{tt})$  by virtue of the way that the semantics of FLC turns a predicate transformer into a predicate. ◀

► **Corollary 7.** *Neither RecCTL nor RecLTL have the finite model property.*

**Proof.** For RecCTL this is a consequence of Thm. 6 since FLC does not have the finite model property [28]. For RecLTL consider the formula  $\varphi_{\text{steps}}(p)$  to be defined next. It is easily seen to be satisfiable, yet unsatisfiable on any finitely represented linear structure. ◀

## 4 Satisfiability and Model Checking

In this section we investigate the issues of decidability and computational complexity of the two most important reasoning problems for temporal logics: satisfiability and model checking.

**Undecidability results.** Consider the RecLTL formula (scheme)

$$\begin{aligned} \varphi_{\text{steps}}(\psi) &:= \psi \wedge X\psi \wedge XX\neg\psi \wedge XXX\psi \\ &\wedge G\left(\psi \rightarrow X\left(\underbrace{\text{rec } \mathcal{H}(x).(\psi \wedge Xx) \vee (\neg\psi \wedge X\mathcal{H}(\neg\psi \wedge Xx))}_{\Phi_{\mathcal{H}}}\right)(\neg\psi \wedge X\psi)\right). \end{aligned}$$

For brevity, let  $X^+\chi$  abbreviate  $\psi \wedge X\chi$  and  $X^-\chi$  abbreviate  $\neg\psi \wedge X\chi$ . Note that the argument to  $\Phi_{\mathcal{H}}$  is  $X^-\psi$  in this respect, and that  $\Phi_{\mathcal{H}}$  can be written as  $\text{rec } \mathcal{H}(x).(X^+x) \vee X^-\mathcal{H}(X^-x)$ . We also have  $X^-(\chi_1 \vee \chi_2) \equiv X^-\chi_1 \vee X^-\chi_2$  in general. Then consider  $\Phi_{\mathcal{H}}(X^-\psi)$ . We have

$$\begin{aligned} \Phi_{\mathcal{H}}(X^-\psi) &\equiv X^+X^-\psi \vee X^-\Phi_{\mathcal{H}}(X^-X^-\psi) \\ &\equiv X^+X^-\psi \vee X^-(X^+X^-X^-\psi \vee X^-\Phi_{\mathcal{H}}(X^-X^-X^-\psi)) \\ &\equiv X^+X^-\psi \vee X^-X^+X^-X^-\psi \vee X^-X^-\Phi_{\mathcal{H}}(X^-X^-X^-\psi) \\ &\equiv X^+X^-\psi \vee X^-X^+X^-X^-\psi \vee X^-X^-(X^+X^-X^-X^-\psi \vee X^-\Phi_{\mathcal{H}}(X^-X^-X^-X^-\psi)) \\ &\equiv \dots \equiv \bigvee_{n \geq 0} \underbrace{X^- \dots X^-}_n X^+ \underbrace{X^- \dots X^-}_{n+1} \psi \end{aligned}$$

The first and second equivalence and every second after that uses Lemma 3. The others simply use the commutation of  $X^-$  with disjunctions.

The first conjuncts in  $\varphi_{\text{steps}}(\psi)$  fix the values of  $\psi$  on a possible model in the first four states, namely to hold at positions 0, 1 and 3. Since  $\psi$  holds at positions 1 and 3 but not at 2,  $G(\psi \rightarrow X\Phi_{\mathcal{H}}(X^-\psi))$  forces  $\psi$  to furthermore hold at position 6 but not at 4 and 5. This can be iterated now with position 3 to see that the next moment at which  $\psi$  holds is 10. Hence,  $\varphi_{\text{steps}}(\psi)$  forces  $\psi$  to hold at the initial point of a model and then at distances increasing by 1 in each step. This can be used in the proof of the next result.

► **Theorem 8.** *The satisfiability problem for RecLTL is undecidable ( $\Sigma_1^1$ -hard).*

**Proof.** The following problem, known as the recurrent octant tiling problem, is  $\Sigma_1^1$ -hard [14]: given a tiling system  $\mathcal{T} = (T, H, V, t_0, t_\infty)$  where  $T$  is a finite set of tile types,  $H, V \subseteq T^2$  and  $t_0, t_\infty \in T$ , is there a tiling  $\tau : \{(i, j) \mid 0 \leq i \leq j\} \rightarrow T$  of the octant plane, such that

- $\tau(0, 0) = t_0$ , (initial tile set properly)
- for all  $i, j$  with  $j > i$ :  $(\tau(i, j), \tau(i+1, j)) \in H$ , (no horizontal mismatch)
- for all  $i, j$  with  $j \geq i$ :  $(\tau(i, j), \tau(i, j+1)) \in V$ , (no vertical mismatch)
- there are infinitely many  $j$  such that  $\tau(0, j) = t_\infty$ ? (recurrence)

Such a tiling  $\tau$  can be represented straight-forwardly as a linear-time model over the set of propositions  $T$  by listing it row-wise:

$$\tau(0, 0), \tau(0, 1), \tau(1, 1), \tau(0, 2), \tau(1, 2), \tau(2, 2), \tau(0, 3), \dots, \tau(3, 3), \tau(0, 4), \dots$$

Moreover, the conditions on a successful tiling can be formalised in RecLTL as follows, using one additional proposition `fst` to mark the beginnings of each row.

$$\begin{aligned} \varphi_{\mathcal{T}} := & t_0 \wedge \mathbf{G} \left( \bigwedge_{t \in T} t \rightarrow \bigwedge_{t' \neq t} \neg t' \right) \wedge \varphi_{\text{steps}}(\text{fst}) \wedge \mathbf{G}(\mathbf{X}\neg\text{fst} \rightarrow \bigvee_{(t,t') \in H} t \wedge \mathbf{X}t') \\ & \wedge \mathbf{G} \left( \text{fst} \rightarrow \neg \bigvee_{(t,t') \in T^2 \setminus V} (\text{rec } H(x,y).(x \wedge \mathbf{X}(\neg\text{fst} \mathbf{U} (\text{fst} \wedge y))) \vee H(\mathbf{X}(\neg\text{fst} \wedge x), \mathbf{X}y))(t, t') \right) \\ & \wedge \mathbf{GF}(\text{fst} \wedge t_{\infty}) \end{aligned}$$

The first conjunct enforces the initiality condition, the second ensures that each position is occupied by exactly one tile. The third conjunct ensures that exactly the positions of the form  $\tau(0, j)$  for any  $j$  are marked using `fst`, using  $\varphi_{\text{steps}}$  constructed above. The fourth ensures horizontal matching by comparing adjacent positions apart from those where the succeeding one is the beginning of the next row. The fifth conjunct states that it is impossible to find the beginning of some row  $j$ , a vertically non-matching pair of tiles  $(t, t')$ , and an  $i \leq j$  such that  $t$  is the tile  $i$  steps after that beginning of the row, and  $t'$  is found  $i$  steps after the next state satisfying `fst`. Note that these are exactly the positions that are vertically adjacent in the octant plane.

The last conjunct ensures the recurrence condition. Now a successful tiling  $\tau$  for  $\mathcal{T}$  induces a linear time model for  $\varphi_{\mathcal{T}}$  in the shape as described and vice-versa. ◀

An immediate consequence of Thms. 4 and 8 is the (high) undecidability of RecCTL.

► **Corollary 9.** *The satisfiability problem for RecCTL is  $\Sigma_1^1$ -hard.*

Also, it is well-known that model checking for linear-time logics under the usual all-paths-semantics is closely related to the validity problem.

► **Corollary 10.** *The model checking problem for RecLTL over transition systems is  $\Pi_1^1$ -hard.*

In contrast, the model checking problem for RecCTL over finite transition systems is decidable.

► **Theorem 11.** *The model checking problem for RecCTL over finite transition systems is EXPTIME-complete.*

**Proof.** A deterministic exponential-time upper bound can be derived from a naïve bottom-up algorithm that computes the semantics  $\llbracket \varphi \rrbracket_{\mathcal{T}}$  of a given formula  $\varphi$  over a given LTS  $\mathcal{T}$  using fixpoint iteration. The EXPTIME upper bound also follows from the (linear) embedding of RecCTL into HFL<sup>1</sup> (Thm. 5) whose model checking problem is known to be EXPTIME-complete [4].

A matching lower bound is inherited from FLC using Thm. 6 since the model checking problem for FLC is known to be EXPTIME-complete as well [24]. ◀

A natural question that arises concerns the decidability of model checking RecCTL over classes of infinite-state transition systems. A consequence of Thm. 6 is the negative result that this problem is already undecidable for the class BPA (Basic Process Algebra) [6] – in some sense the smallest class of context-free processes – as this is undecidable for FLC [28, 22].

► **Corollary 12.** *The model checking problem for RecCTL over classes of infinite-state transition systems subsuming BPA is undecidable.*



**A decidable fragment of RecCTL.** The undecidability of the satisfiability problem for temporal logics beyond regularity can often be seen by observing that such a logic, for example FLC, can both express context-free properties, and is closed under conjunctions. Since closure under Boolean operators is highly desirable, a restriction of the recursion process to a class strictly below the context-free languages is unavoidable if one wants to recuperate decidability. A natural candidate are the *visibly pushdown languages* (VPL) [2], which are closed under intersection and complement and, hence, not problematic if mixed with full closure under Boolean operators. In particular, it is known that PDL[VPL] is decidable and 2EXPTIME-complete [26]. We refer to the literature for a detailed introduction into VPL and PDL[VPL].

► **Definition 13.** *Let  $\mathcal{P}$  be a set of propositions. We write  $\mathbb{B}(\mathcal{P})$  for the set of all Boolean combinations of these variables. Let  $\mathcal{B}_c, \mathcal{B}_r, \mathcal{B}_i \subseteq \mathbb{B}(\mathcal{P})$  be mutually exclusive, i.e. the conjunction of two formulas from these sets is satisfiable only if they are from the same set.*

*Formulas of the fragment vpRecCTL of RecCTL are given by the grammar*

$$\varphi ::= \mathbf{tt} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbf{rec}_i \left( \begin{array}{c} \mathcal{F}_1(x_1) \quad \cdot \quad \psi_1 \\ \vdots \\ \mathcal{F}_n(x_n) \quad \cdot \quad \psi_n \end{array} \right) (\varphi)$$

*where each  $\psi_j$  is a disjunction of formulas of the forms  $x_j$ ,  $\mathbf{EX}(\beta_i \wedge \mathcal{F}_k(x_j))$  or  $\mathbf{EX}(\beta_c \wedge \mathcal{F}_k(\mathbf{EX}(\beta_r \wedge \mathcal{F}_{k'}(x_j))))$ , with  $\beta_c \in \mathcal{B}_c$ ,  $\beta_r \in \mathcal{B}_r$  and  $\beta_i \in \mathcal{B}_i$ . Note that formulas derived from  $\varphi$  contain no free variables from  $\mathcal{V}_2$ .*

► **Theorem 14.** *The satisfiability problem for vpRecCTL is 2EXPTIME-complete.*

**Proof.** (sketch) It is possible to devise a satisfiability-preserving linear translation from vpRecCTL into PDL[VPL]. From  $\mathcal{B}_c, \mathcal{B}_r, \mathcal{B}_i$  we construct a visibly pushdown alphabet  $\mathcal{A} := \mathcal{A}_c \cup \mathcal{A}_r \cup \mathcal{A}_i$  with  $\mathcal{A}_c := \{a_\beta \mid \beta \in \mathcal{B}_c\}$  and likewise for  $\mathcal{A}_r$  and  $\mathcal{A}_i$ .

The key is then to see that the structure of a functional formula  $\Phi = (\mathbf{rec}_i \mathcal{F}_1(x_1). \psi_1, \dots, \mathcal{F}_n(x_n). \psi_n)$  in vectorial form resembles a context-free grammar  $G$  with  $\mathbf{EX}(\beta_c \wedge \mathcal{F}_k(\mathbf{EX}(\beta_r \wedge \mathcal{F}_{k'}(x_j))))$  in the definition of some  $\mathcal{F}_i$  for instance corresponding to a production of the form  $\mathcal{F}_i \rightarrow a_{\beta_c} \mathcal{F}_k a_{\beta_r} \mathcal{F}_{k'}$ . The structure of  $\Phi$  then ensures that the resulting grammar is in fact a visibly-pushdown grammar  $G_\Phi$  [2], and so  $\Phi(\varphi)$  can be translated into  $\langle G_\Phi \rangle(\hat{\varphi})$  where  $\hat{\varphi}$  is the translation of  $\varphi$ .

The lower bound follows equally from PDL[VPL]’s 2EXPTIME-completeness [26], as the translation can easily be reversed into one from PDL[VPL] to vpRecCTL. ◀

An equivalence-preserving translation from vpRecCTL into PDL[VPL] is not possible since PDL[VPL] is defined over edge-labelled LTS, and the partition of edge labels into a visibly pushdown alphabet plays a key role in the definition of the logic. Without the shift from node- to edge-labels – keeping each  $\beta$  as a propositional formula rather than transforming it into an alphabet symbol  $a_\beta$  – the translation would indeed be equivalence preserving but the resulting formula would “only” be in PDL[CFL].

► **Corollary 15.** *Every vpRecCTL can equivalently be expressed in PDL[CFL].*



## 5 Conclusion & Further Work

We have presented an expressive extension of the framework of standard temporal logics. The aim is to make the specification and verification of complex systems and properties beyond regular ones more accessible through temporal logics with a reasonably intuitive syntax and semantics, such as CTL and LTL are.

High expressive power is achieved through the introduction of a recursion operator which takes formulas as arguments. The mathematical concepts underlying the formal semantics are borrowed from higher-order logics like HFL without having to involve rather cumbersome tools like proof systems. Instead, only a relatively simple monotonicity requirement for recursion variables has to be obeyed. This way, RecCTL and RecLTL achieve a reasonable balance between expressive power and pragmatic usability.

We have studied the computational complexities of the most important decision problems of model and satisfiability checking of these logics. The increase compared to CTL and LTL is in line with what one can expect to pay for additional expressiveness.

There are various routes for further work on such logics. For instance, model checking procedures that are optimised for practical purposes need to be sought. There is also potential in extending the fragment vpRecCTL by large amounts without losing the decidability property. Take for instance the infinitary modal formula  $\bigvee_{n \geq i} \diamond^n \square^n q$ , stating that for some  $n$  there is a path of length  $n$  such that all successive paths of that same length end in a  $q$ -state. This cannot be stated in PDL[VPL] even though it intuitively uses the VPL  $\{a^n b^n \mid n \geq 1\}$ , but PDL-based logics can only combine languages with a single modality. The property is, however, easily formalisable in RecCTL as  $\text{EX}(\text{rec } \mathcal{F}(x).x \vee \text{EX } \mathcal{F}(\text{AX } x))(\text{AX } q)$ .

We conjecture that it is possible to allow mixtures of EX- and AX-operators in vpRecCTL, achieving higher expressiveness and yet not losing decidability. We believe that satisfiability in such an extended fragment can also be reduced to the problem of solving a visibly-pushdown game [27].

---

### References

- 1 R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.
- 2 R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th Annual ACM Symp. on Theory of Computing, STOC'04*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.
- 3 A. Arnold and D. Niwiński. *Rudiments of  $\mu$ -calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.
- 4 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.
- 5 H. Bekić. *Programming Languages and Their Definition, Selected Papers*, volume 177 of *LNCS*. Springer, 1984.
- 6 J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1–3):109–137, 1984.
- 7 E. A. Emerson. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 24(2):77–79, 1987.
- 8 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.
- 9 E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.
- 10 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.

- 11 E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 2000.
- 12 M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
- 13 H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
- 14 D. Harel. Recurring dominoes: Making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.
- 15 D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.
- 16 D. Janin and I. Walukiewicz. On the expressive completeness of the propositional  $\mu$ -calculus with respect to monadic second order logic. In *CONCUR*, pages 263–277, 1996. doi:10.1007/3-540-61604-7\_60.
- 17 B. Knaster. Un théorème sur les fonctions d'ensembles. *Annals Soc. Pol. Math*, 6:133–134, 1928.
- 18 R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, November 1990.
- 19 D. Kozen. Results on the propositional  $\mu$ -calculus. *TCS*, 27:333–354, December 1983. doi:10.1007/BFb0012782.
- 20 D. Kozen. A finite model theorem for the propositional  $\mu$ -calculus. *Studia Logica*, 47(3):233–241, 1988.
- 21 L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.
- 22 M. Lange. Local model checking games for fixed point logic with chop. In *Proc. 13th Conf. on Concurrency Theory, CONCUR'02*, volume 2421 of *LNCS*, pages 240–254. Springer, 2002.
- 23 M. Lange. Temporal logics beyond regularity, 2007. Habilitation thesis, University of Munich, BRICS research report RS-07-13.
- 24 M. Lange. Three notes on the complexity of model checking fixpoint logic with chop. *R.A.I.R.O. – Theoretical Informatics and Applications*, 41:177–190, 2007.
- 25 M. Lange and R. Somla. Propositional dynamic logic of context-free programs and fixpoint logic with chop. *Information Processing Letters*, 100(2):72–75, 2006.
- 26 C. Löding, C. Lutz, and O. Serre. Propositional dynamic logic with recursive programs. *J. Log. Algebr. Program*, 73(1-2):51–69, 2007.
- 27 C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proc. 24th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'04*, volume 3328 of *LNCS*, pages 408–420. Springer, 2004.
- 28 M. Müller-Olm. A modal fixpoint logic with chop. In *STACS'99*, volume 1563 of *LNCS*, pages 510–520. Springer, 1999.
- 29 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.
- 30 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.
- 31 C. Stirling. Comparing linear and branching time temporal logics. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proc. Conf. on Temporal Logic in Specification*, volume 398 of *LNCS*, pages 1–20, Berlin, 1989. Springer.
- 32 A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–309, 1955.
- 33 M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th Symp. on Theory of Computing, STOC'85*, pages 240–251, Baltimore, USA, 1985. ACM.
- 34 M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004.

# Parametric Model Checking Continuous-Time Markov Chains

Catalin-Andrei Ilie<sup>1</sup>

Department of Computer Science, University of Bucharest, Romania  
cilie@fmi.unibuc.ro

James Ben Worrell

Department of Computer Science, University of Oxford, UK  
jbw@cs.ox.ac.uk

---

## Abstract

CSL is a well-known temporal logic for specifying properties of real-time stochastic systems, such as continuous-time Markov chains. We introduce PCSL, an extension of CSL that allows using existentially quantified parameters in timing constraints, and investigate its expressiveness and decidability over properties of continuous-time Markov chains. Assuming Schanuel’s Conjecture, we prove the decidability of model checking the one-parameter fragment of PCSL on continuous-time Markov chains. Technically, the central problem we solve (relying on Schanuel’s Conjecture) is to decide positivity of real-valued exponential polynomial functions on bounded intervals. A second contribution is to give a reduction of the Positivity Problem for matrix exponentials to the PCSL model checking problem, suggesting that it will be difficult to give an unconditional proof of the decidability of model checking PCSL.

**2012 ACM Subject Classification** Theory of computation → Logic and verification; Theory of computation → Verification by model checking; Theory of computation → Random walks and Markov chains

**Keywords and phrases** Probabilistic Continuous Stochastic Logic, Continuous-time Markov Chains, model checking, Schanuel’s Conjecture, positivity problem

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.7

**Funding** *James Ben Worrell*: Supported by EPSRC Fellowship EP/N008197/1.

## 1 Introduction

Continuous-time Markov chains (CTMC) have been intensively investigated for a long time, especially because they are simple stochastic models with a wide range of real life applications, being suitable for modelling properties such as expected failure time for systems or expected time between system events. Given the omnipresence of continuous-time Markov chains, it has been natural to seek a logical formalism to describe their properties. A popular example is Continuous Stochastic Logic (CSL), introduced by Aziz et al in [2]. CSL is a branching-time, temporal logic, that allows expressing quantitative bounds on certain properties of continuous-time Markov chains.

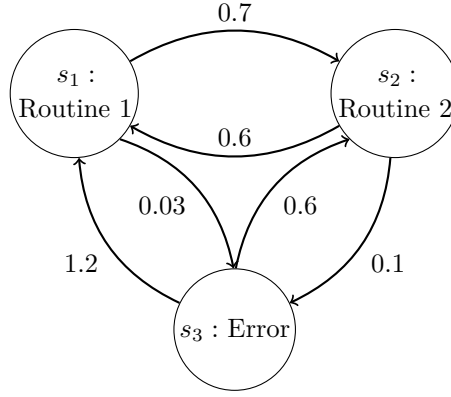
Let us consider the CTMC  $\mathcal{M}$  in Figure 1 modelling the state transitions of a simple system. One can express the property that the probability of encountering an error in the continuous time interval  $[0, 4]$  is greater than 0.5 in CSL by the following state formula:

$$\varphi := \mathbb{P}_{>0.5}(true \mathbf{U}_{[0,4]} s_3). \quad (1)$$

---

<sup>1</sup> Most of the research was done as part of Andrei’s dissertation while he was an undergraduate student at the University of Oxford working under the supervision of James Worrell.





■ **Figure 1** A simple CTMC modelling a system which is considered to run properly in states  $s_1$  and  $s_2$ , and to malfunction in state  $s_3$ .

Another natural property that one might want to express is whether there exists a “dangerous” short period in  $[0, 4]$ , say of length 0.1, in which our example system fails with probability at least 0.3. This could then lead to isolating such periods and taking appropriate action. However, CSL does not allow expressing such properties. This is why we extend CSL to allow existential quantifiers over time bounds, giving rise to the logic Parametric CSL (PCSL), in which we can express the desired property by a state formula:

$$\psi := \exists t \in [0, 3.9] \cdot \mathbb{P}_{>0.3}(\text{true} \mathbf{U}_{[t, t+0.1]} s_3). \quad (2)$$

In general, checking if mathematical models satisfy certain properties is a central part of formal verification. This gives rise to *model checking* problems, in which we want to find procedures to determine if a model verifies properties that are usually expressed formally within a logic. In CSL, the model checking problem consists of deciding if properties expressed by state formulas are true or false in certain states of a CTMC. The main result of [2] is that CSL model checking is decidable. The proof is non-trivial, as it employs results in algebraic and transcendental number theory such as the Lindemann-Weierstrass theorem [10]. There exist state-of-the-art model checking software, such as PRISM [8], which allow verifying properties of systems, including CTMCs, expressed formally by logics like CSL, PCTL. However, in this project we deal theoretically with the fundamental problem regarding PCSL model checking.

We define the model checking problem of PCSL similarly to the one of CSL, with the simple exception that we allow state formulas to be evaluated over initial distributions instead of states. Therefore, we want to decide if a CTMC together with an initial distribution entail a PCSL state formula<sup>2</sup>. We show that the model checking problem for the fragment of PCSL consisting of formulae with only one existential quantifier, such as 2 above, is decidable assuming Schanuel’s Conjecture, a conjecture which generalizes important results in transcendental number theory, including the Lindemann-Weierstrass Theorem. The latter was used in [2] to prove CSL model checking decidability. We also discuss why PCSL model checking decidability is non-trivial and employs a strong number theoretical result.

<sup>2</sup> Note that this simply allows for checking entailment in a certain state by setting its initial probability to 1.

## 2 PCSL Syntax and Semantics

We extend the original CSL formulation of Aziz et al by allowing existentially quantified parameters. Sticking to the terminology in [2], we call a “path” through a CTMC  $M$  a function on domain  $[0, \infty)$  with values in the state set  $S$ , which associates to each time step a state and follows the transitions in  $M$ . For any state  $s$  we denote by  $U^s$  the set of paths starting at  $s$ . Similarly, we denote by  $U^M$  the set of paths starting at any state in  $M$ . For any set of paths  $\Gamma$  starting at the same state  $s$  we denote its probability by  $\mu^s(\Gamma)$ .

For any initial distribution  $\pi$  and for any set of paths  $\Phi$ , not necessarily starting from the same state, we denote its probability by  $\mu^\pi(\Phi)$ , where the probability of the initial vertex is determined according to the initial distribution  $\pi$  of  $M$ :

$$\mu^\pi(\Phi) := \sum_{s \in S} \pi(s) \mu^s(U^s \cap \Phi). \quad (3)$$

We now give the syntax and semantics of our extension, which we name “Parametric Continuous Stochastic Logic” (PCSL). For clarity, we use in PCSL state names instead of state labels in the formulas, while the authors of the original CSL papers use state labels. We also define the satisfaction relation for state formulas over initial distributions instead of states, to allow a wider class of verifiable models. Apart from this, CSL can be seen as the PCSL restriction when using no quantified parameters in the definitions below. We also define  $\text{PCSL}_n$ , for  $n \in \mathbb{N}$ , to be the restriction of PCSL with at most  $n$  nested existential quantifiers.

### 2.1 PCSL Syntax

First, let  $T = \{t_1, t_2, \dots\}$  be a countably infinite set of free variables to which we have access. These variables will represent existentially quantified real numbers. We define a *parametric term* over a finite set of free variables  $T' \subset T$  as a linear combination of free variables in  $T'$  with rational coefficients:

1.  $c$  is a parametric term, for any  $c \in \mathbb{Q}$ ,
2.  $\tau + qt$  is a parametric term, for any parametric term  $\tau$ ,  $q \in \mathbb{Q}$ , and  $t \in T'$ .

Let  $M$  be a CTMC with state set  $S$ . As in CSL, there are two types of PCSL formulas: *state formulas* and *path formulas*.

State formulas are evaluated in states, or over initial distributions<sup>3</sup>, and their syntax is given by:

1.  $s$ , for  $s \in S$  (the atomic state formula),
2. If  $f_1$  and  $f_2$  are state formulas, then so are  $\neg f_1$  and  $f_1 \vee f_2$ ,<sup>4</sup>
3. If  $g$  is a path formula using parametric terms over free variables  $\{t_1, \dots, t_r\}$ , then  $\exists t_1 \in [x_1, y_1] \dots \exists t_r \in [x_r, y_r] \cdot \mathbb{P}_{>c}(g)$  is a state formula, where  $c \in \mathbb{Q}$ , and for  $i = 1, \dots, r$ :  $x_i, y_i \in \mathbb{Q}$ , and  $0 \leq x_i \leq y_i$ .<sup>5</sup>

Path formulas are evaluated along paths, and their syntax is :

<sup>3</sup> In CSL, the state formulas are only evaluated in states. Our extension allows evaluation over initial distributions as well.

<sup>4</sup>  $f_1 \wedge f_2$  and  $f_1 \rightarrow f_2$  can be written using only these definitions

<sup>5</sup> Note that we allow as well no free variables, so no quantifiers at all in such a formula. PCSL differs from CSL specifically by allowing these  $\exists$  operators.

## 7:4 Parametric Continuous Stochastic Logic

1.  $f_1 \mathbf{U}_{[a_1, b_1]} f_2 \mathbf{U}_{[a_2, b_2]} \dots f_n$ , where  $f_1, f_2, \dots, f_n$  are state formulas, and all  $a_1, \dots, a_{n-1}$  and  $b_1, \dots, b_{n-1}$  are parametric terms over a finite set of free variables.

For any  $k \in \mathbb{N}$ , the syntax of  $\text{PCSL}_k$  is the same as of  $\text{PCSL}$ , with the exception of the path formula rule 1, which for  $\text{PCSL}_k$  is:

1.  $f_1 \mathbf{U}_{[a_1, b_1]} f_2 \mathbf{U}_{[a_2, b_2]} \dots f_n$ , where  $f_1, f_2, \dots, f_n$  are state formulas, and all  $a_1, \dots, a_{n-1}$  and  $b_1, \dots, b_{n-1}$  are parametric terms over a set of free variables of size at most  $k$ .

### 2.2 PCSL Semantics

Let  $M$  be a CTMC, with state set  $S$ , and initial distribution  $\pi_0$ . Let  $f$  and  $g$  be PCSL state, respectively path formulas.

We say that a state  $s$  satisfies a state formula  $f$  if, for a distribution  $\pi'$  such that  $\pi'(s) = 1$ , we have  $M, \pi' \models f$  according to the definitions below. Let us denote by  $\llbracket f \rrbracket_M$  the set of states satisfying  $f$ . We also denote by  $g[t \leftarrow d]$  the path formula obtained by substituting the occurrences of the free variable  $t$  in parametric terms of  $g$  by the non-negative real  $d$ . We define the satisfaction relation  $M, \pi \models f$  for a general rational distribution  $\pi$ , using structural induction over the state formula  $f$ :

1.  $f$  is of the form  $s$  ( $s \in S$ ):  $M, \pi \models f$  iff  $\pi(s) = 1$ ,
2.  $f$  is of the form  $\neg f_1$ :  $M, \pi \models f$  iff  $M, \pi \not\models f_1$ ,
3.  $f$  is of the form  $f_1 \vee f_2$ :  $M, \pi \models f$  iff  $M, \pi \models f_1$  or  $M, \pi \models f_2$ ,
4.  $f$  is of the form  $\exists t_1 \in [x_1, y_1] \dots \exists t_r \in [x_r, y_r] \cdot \mathbb{P}_{>c}(g)$ :  $M, \pi \models f$  iff there exist non-negative reals  $c_1 \in [x_1, y_1], \dots, c_r \in [x_r, y_r]$  such that

$$\mu^\pi(\{\rho \in U^M \mid M, \rho \models g[t_1 \leftarrow c_1][t_2 \leftarrow c_2] \dots [t_r \leftarrow c_r]\}) > c,$$

By notation abuse, we define the satisfaction relation  $M, \rho \models g$  for path formulas  $g$  and for any path  $\rho$ :

1.  $g$  is a path formula with no free variables (i.e. all parametric terms are numbers) of the form  $f_1 \mathbf{U}_{[a_1, b_1]} f_2 \mathbf{U}_{[a_2, b_2]} \dots f_n$  and  $\rho$  is a path through  $M$ :  $M, \rho \models g$  iff there exist positive reals  $\alpha_1, \dots, \alpha_{n-1}$  such that for each integer in  $[1, n-1]$  we have  $a_i \leq \alpha_i \leq b_i$  and for any  $\beta \in [\alpha_{i-1}, \alpha_i]$  we have  $\pi(\beta) \in \llbracket f_i \rrbracket_M$ , and  $\pi(\alpha_{n-1}) \in \llbracket f_n \rrbracket_M$ .<sup>6</sup>

We further overwrite the satisfaction relation as follows:

- we define  $M \models f$  iff  $M, \pi_0 \models f$ , where  $\pi_0$  is the initial distribution of  $M$ ,
- for any  $s \in S$ , we define  $M, s \models f$  iff  $M, \pi' \models f$ , where distribution  $\pi'$  is chosen over  $S$  such that  $\pi'(s) = 1$  and  $\pi'(s') = 0$ , for any  $s' \neq s$ .

### 2.3 PCSL Formulas Examples

The PCSL formula

$$\phi_3 := s_1 \wedge \exists t \in [0, 5] \cdot \mathbb{P}_{>0.5}(\text{true} \mathbf{U}_{[t, t]} s_2)$$

expresses the property that the system is initially in state  $s_1$  and there exists an instantaneous moment  $t \leq 5$  during which the probability of being in state  $s_2$  is greater than 0.5. The formula  $\phi_3$  is in  $\text{PCSL}_1$ , but not in  $\text{CSL}$ . Note that it is different from the  $\text{CSL}$  formula  $\phi_4 := s_1 \wedge \mathbb{P}_{>0.5}(\text{true} \mathbf{U}_{[0, 5]} s_2)$ , which expresses the property of being in state  $s_1$  initially and transitioning to  $s_2$  at any moment before 5.0 with probability greater than 0.5.

<sup>6</sup> The real number  $\alpha_0$  is defined to be 0 for convenience. There are other ways to define the semantics for the path formula, but we want to be consistent with [2].

A more interesting example is motivated by the following situation. Suppose we have a system and we want a state formula  $\varphi$  to hold with high probability ( $> 0.8$ ) before time  $t = 5$ , but we do not want the state formula to be too biased towards any short period, i.e., we do not want there to be any continuous time period of length 0.1 such that  $\varphi$  holds with probability greater than 0.2. This can be modelled in PCSL (in PCSL<sub>1</sub>) as:

$$\phi_5 := \mathbb{P}_{>0.8}(\text{true}\mathbf{U}_{[0,5]}\varphi) \wedge \neg\exists t \in [0, 4.9] \cdot \mathbb{P}_{>0.2}(\text{true}\mathbf{U}_{[t,t+0.1]}\varphi).$$

The following formula is in PCSL<sub>2</sub>, but (syntactically) not in PCSL<sub>1</sub>, as it contains a path formula with two free variables:

$$\phi_6 := \exists t_1 \in [0, 1] \exists t_2 \in [3, 4] \cdot \mathbb{P}_{>0.5}(\text{true}\mathbf{U}_{[t_1,t_1]}s_1 \mathbf{U}_{[t_1,t_2]}\text{true}\mathbf{U}_{[t_2,t_2]}s_2).$$

Formula  $\phi_6$  expresses the property that there are some moments  $t_1 \in [0, 1]$  and  $t_2 \in [3, 4]$  such that the probability of being in state  $s_1$  at time  $t_1$  and in state  $s_2$  at time  $t_2$  is greater than 0.5.

### 3 Mathematical Background

#### 3.1 Exponential Polynomials

► **Definition 1.** *An exponential polynomial is a function  $f(t) = \sum_{i=1}^m P_i(t)e^{\alpha_i t}$ , where  $P_i \in \mathbb{C}[t]$  are polynomials with complex coefficients and  $\alpha_1, \dots, \alpha_m$  are complex numbers. We call the coefficients of polynomials  $P_1, \dots, P_m$  and the numbers  $\alpha_1, \dots, \alpha_m$  the coefficients of the exponential polynomial  $f$ .*

Exponential polynomials often arise when writing the explicit solutions of ordinary linear differential equations and when modelling probability distributions in dynamic systems, such as continuous-time Markov chains [4, 2, 3]. We will mainly be concerned with exponential polynomials with algebraic coefficients that are real-valued over reals, i.e., if  $t$  is real, then  $f(t)$  is real. We simply refer to such functions as real-valued.

The following result, a standard linear algebra result (see [3, 2] for a detailed proof), will later on give the relation between transition probabilities of continuous-time Markov chains and exponential polynomials:

► **Lemma 2.** *Let  $A$  be an  $n \times n$  matrix with rational (algebraic) entries. Then, for any  $\alpha, \beta \in \mathbb{Q}$ , the entries of the exponential matrix  $f(t) := \exp(A(\alpha t + \beta))$  are real-valued exponential polynomials with algebraic coefficients.*

#### 3.2 Schanuel's Conjecture

Schanuel's Conjecture is a unifying conjecture in the field of transcendental number theory, having as consequences important results about exponential functions over both real and complex numbers, such as in the work of Zilber [12], and in model theory, such as decidability of the first-order theory  $Th_{\text{exp}}(\mathbb{R})$  of the field of real numbers with exponentials  $\langle \mathbb{R}, +, \times, \exp, 0, 1 \rangle$  [9], and decidability of the Continuous Skolem Problem [4].

Schanuel's Conjecture has the following form:

► **Conjecture 1.** *(Schanuel's Conjecture) Given any  $n$  complex numbers  $z_1, \dots, z_n$  that are linearly independent over  $\mathbb{Q}$ , the extension field  $\mathbb{Q}(z_1, \dots, z_n, e^{z_1}, \dots, e^{z_n})$  has transcendence degree at least  $n$  over  $\mathbb{Q}$ .*

Schanuel's Conjecture states that, for  $z_i$ 's as above, among  $z_1, \dots, z_n, e^{z_1}, \dots, e^{z_n}$  there are at least  $n$  numbers which are not related by any non-trivial polynomial with rational coefficients.

Schanuel's Conjecture is a generalisation of Lindemann-Weierstrass theorem, which lies at the heart of the decidability proof of CSL in [2], the logic that we extend into PCSL.



### 3.3 The Positivity Problem

► **Definition 3.** *An instance of the Positivity Problem for exponential polynomials is a real-valued exponential polynomial  $f(t) = \sum_{j=1}^m P_j(t)e^{\lambda_j t}$  with algebraic coefficients, together with an interval  $[c, d]$ , where  $c, d \in \mathbb{Q}_+$  ( $d \geq c \geq 0$ ). We want to answer the question: does there exist  $t \in [c, d]$  such that  $f(t) > 0$ ?*

We show in the appendix that deciding whether a real-valued exponential polynomial with algebraic coefficients is strictly positive at some point in a given bounded interval with rational endpoints is decidable under Schanuel's Conjecture. This decision problem is of particular interest because it arises naturally in continuous linear dynamical systems, as we will see in our CSL extension. We mainly build our proof on top of the one in [4], which shows that we can decide, subject to Schanuel's Conjecture, whether a real-valued exponential polynomial with algebraic coefficients has a zero in a given bounded interval. The additional complexity in the current problem comes from the fact that detecting sign changes for a real-valued exponential polynomial is based on the behaviour of all its factors together, unlike detecting roots, where only the behaviour of one of its factors matters.

► **Theorem 4.** *The Positivity Problem for exponential polynomials is decidable assuming Schanuel's Conjecture.*

The following is a proof outline of Theorem 4; full details can be found in the Appendix. Suppose that we want to decide whether a given real-valued exponential polynomial  $f$  is positive throughout an interval  $[c, d]$ . We reduce this problem to deciding the existence of zeros of exponential polynomials on bounded intervals, which is known to be decidable conditional on Schanuel's Conjecture [4]. In fact the reduction itself uses several of the ideas developed in [4]. To carry out the reduction we first compute the sign of  $f$  at the endpoints  $c$  and  $d$ . Suppose that  $f$  is negative at both endpoints. We then compute a factorisation of  $f$  in the form  $f = f_1^{\alpha_1} \cdots f_k^{\alpha_k}$ , where the factors  $f_i$  are real-valued exponential polynomials that do not share any common zeros. Then determining whether  $f$  changes sign from negative to positive on  $[c, d]$  reduces to determining whether one of its factors  $f_i$  with odd exponent  $\alpha_i$  changes sign. To solve this last problem we give an effectively decidable categorisation of the factors into two types.

We show that factors of the first type are always nonnegative and factors  $g$  of second type are such that  $g$  and  $g'$  have no common zeros, i.e., they always sign at every zero. Thus  $f$  becomes positive on  $[c, d]$  iff it has a factor of the second type that has a zero in  $[c, d]$ . The role of Schanuel's conjecture in the above argument is to rule out the existence of common zeros of the different factors of  $f$  and common zeros of certain factors and their derivatives.

► **Remark 5.** Given a function  $f$  and an interval  $[c, d]$  that are an instance of the Positivity Problem for exponential polynomials, a decision procedure for this problem trivially implies a decision procedure for checking in a similar setup if there exists  $t \in [c, d]$  such that  $f(t) > q$ , for any given rational  $q$ . This follows as we can let  $g(t) := f(t) - q$ , so  $g$  is an exponential polynomial with algebraic coefficients as well, therefore we can use a decision procedure for the Positivity Problem for exponential polynomials on input function  $g$  and interval  $[c, d]$  and decide if there exists  $t \in [c, d]$  such that  $f(t) > q$ .

The Positivity Problem for exponential polynomials is a hard problem, as it is trivially inter-reducible with the Non-negativity Problem for exponential polynomials [3], which has the same setup as the Positivity Problem for exponential polynomials, but asks whether for all  $t \in [c, d]$  it is true that  $f(t) \geq 0$ . Concretely, decidability of any of the two problems



implies decidability of the other as well:

$$\forall t \in [c, d]. f(t) \geq 0 \Leftrightarrow \neg \exists t \in [c, d]. (-f(t) > 0), \quad (4)$$

$$\exists t \in [c, d]. f(t) > 0 \Leftrightarrow \neg \forall t \in [c, d]. (-f(t) \geq 0). \quad (5)$$

However, decidability of the Non-negativity Problem for exponential polynomials is open [3], so decidability of the Positivity Problem for exponential polynomials is a hard mathematical task. In fact, comparing exponential polynomials with 0 is a hard task [4, 9, 3], and decision problems related to this would have considerable new implications in both model theory and number theory [4]. This motivates us to work under the assumption of Schanuel's Conjecture, which is often assumed in model theory [4, 12], as the unconditional decidability currently seems out of reach.

## 4 Model Checking Decidability of PCSL

A central problem in formal verification for any logic describing dynamic systems is the *model checking problem*. Intuitively, it asks whether a model of a certain system satisfies a specification, usually expressed withing a logical formalism.

We introduce the PCSL model checking problem below.

► **Definition 6** (Model checking problem for PCSL). *An instance of the model checking problem for PCSL is given by a continuous-time Markov chain  $M$ , a distribution  $\pi$  with rational entries over the states of  $M$ , and a PCSL formula  $\varphi$ . We want to answer the question: is it the case that  $M, \pi \models \varphi$ ?*

The model checking problem for  $\text{PCSL}_n$ , for any  $n \in \mathbb{N}$ , is defined similarly, with the exception that  $\varphi$  is a  $\text{PCSL}_n$  formula in Definition 6. We prove in subsection 4.1 that  $\text{PCSL}_1$  model checking is decidable assuming Schanuel's Conjecture. We also show in subsection 4.2 that unconditional  $\text{PCSL}_1$  model checking is hard from a mathematical point of view, by reducing a well-known hard problem to it.

### 4.1 Decidability of the model checking problem for $\text{PCSL}_1$ assuming Schanuel's Conjecture

We show that  $\text{PCSL}_1$  model checking is decidable assuming Schanuel's Conjecture. For this, we prove that the decidability of the Positivity Problem for exponential polynomials implies  $\text{PCSL}_1$  model checking decidability. As discussed in Section 3.3, Schanuel's Conjecture implies decidability of the Positivity Problem for exponential polynomials, therefore we get our result.

Given any Markov chain  $M$  with state set  $S = \{s_1, \dots, s_k\}$  and rational transition rate matrix  $Q$ , and an initial rational distribution  $\pi$ , we proceed by structural induction over  $\text{PCSL}_1$  formula  $\varphi$  to show that there exists a model checking procedure to determine if  $M, \pi \models \varphi$ .

Let us first deal with the trivial cases.

If  $\varphi$  is an atom (state)  $s$ , then  $M, \pi \models \varphi$  iff  $\pi(s) = 1$ .

If  $\varphi = \varphi_1 \vee \varphi_2$ , we have  $M, \pi \models \varphi$  iff  $M, \pi \models \varphi_1$  or  $M, \pi \models \varphi_2$ .

If  $\varphi = \neg \varphi_1$ , we have  $M, \pi \models \varphi$  iff  $M, \pi \not\models \varphi_1$ .

If  $\varphi = \mathbb{P}_{>c}(\varphi_1 \mathbf{U}_{[a_1, b_1]} \varphi_2 \mathbf{U}_{[a_2, b_2]} \dots \mathbf{U}_{[a_{n-1}, b_{n-1}]} \varphi_n)$ , as we can model check formulas  $\varphi_1, \dots, \varphi_n$  by induction, the decidability follows from the same proof used by Aziz et al in [2] to show that classic CSL is decidable, by only using the Lindemann-Weierstrass theorem.

## 7:8 Parametric Continuous Stochastic Logic

Now, we have to deal with the case

$$\varphi = \exists t \in [a, b] \cdot \mathbb{P}_{>c}(\varphi_1 \mathbf{U}_{[a_1, b_1]} \varphi_2 \mathbf{U}_{[a_2, b_2]} \cdots \mathbf{U}_{[a_{n-1}, b_{n-1}]} \varphi_n),$$

where  $a, b, c \in \mathbb{Q}$ , and  $a_1, b_1, \dots, a_{n-1}, b_{n-1}$  are parametric terms over  $\{t\}$  (functions in  $t$  of the form  $\alpha t + \beta$ , with  $\alpha, \beta \in \mathbb{Q}$ ). We therefore need to reason about the quantity

$$f(t) := \mu^\pi(\{\text{paths } \rho \in U^M \mid M, \rho \models \varphi_1 \mathbf{U}_{[a_1, b_1]} \varphi_2 \mathbf{U}_{[a_2, b_2]} \cdots \mathbf{U}_{[a_{n-1}, b_{n-1}]} \varphi_n\}). \quad (6)$$

In fact, we are interested if there exists some  $t \in [a, b]$  such that  $f(t) > c$ . We further show that  $f(t)$  is an exponential polynomial that we can algorithmically compute, which gives us the sought conditional decidability result.

Assume for the moment that for any  $t \in [a, b]$  we have

$$0 \leq a_1 \leq b_1 \leq \cdots \leq a_{n-1} \leq b_{n-1}. \quad (7)$$

By the structural induction hypothesis, we can compute the sets of states  $\llbracket \varphi_1 \rrbracket_M, \dots, \llbracket \varphi_n \rrbracket_M$  satisfying subformulas  $\varphi_1, \dots, \varphi_n$ . For any subset of states  $H \subseteq S$ , let its complement be  $H^c := S \setminus H$ .

We show how to compute the probability function  $f(t)$ , by similar constructions to the ones in [2]. Let us construct the following matrices, where for any matrix  $A$  we refer to its entry on row  $i$  and column  $j$  as  $A(i, j)$ .

- Let  $Q_{i,i}$  be a transition rate matrix that models states in  $\llbracket \varphi_i \rrbracket_M^c$  as absorbing states, and is everywhere else identical to  $Q$ :

$$Q_{i,i}(j, k) := \begin{cases} Q(j, k), & \text{if } s_j \in \llbracket \varphi_i \rrbracket_M, \\ 0, & \text{if } s_j \in \llbracket \varphi_i \rrbracket_M^c. \end{cases}$$

This matrix is used to model a run of  $M$  which remains in states satisfying  $\varphi_i$ . Also, let  $P_{i,i}(t) := \exp(Q_{i,i}t)$  be the transition matrix for time  $t$  corresponding to the Markov chain described by  $Q_{i,i}$ .

- Let  $Q_{i,i+1}$  be a transition rate matrix obtained from  $Q$  that only models transitions from  $\llbracket \varphi_i \rrbracket_M$  to  $\llbracket \varphi_i \rrbracket_M \cup \llbracket \varphi_{i+1} \rrbracket_M$ , and from  $\llbracket \varphi_{i+1} \rrbracket_M$  to  $\llbracket \varphi_{i+1} \rrbracket_M$ :

$$Q_{i,i+1}(j, k) := \begin{cases} Q(j, k), & \text{if } s_j \in \llbracket \varphi_i \rrbracket_M \text{ and } s_k \in \llbracket \varphi_i \rrbracket_M \cup \llbracket \varphi_{i+1} \rrbracket_M, \\ Q(j, k), & \text{if } s_j \in \llbracket \varphi_{i+1} \rrbracket_M \text{ and } s_k \in \llbracket \varphi_{i+1} \rrbracket_M, \\ 0, & \text{otherwise.} \end{cases}$$

This matrix is used to model transitions from states satisfying  $\varphi_i$  to states satisfying  $\varphi_{i+1}$ . Also, let  $P_{i,i+1}(t) := \exp(Q_{i,i+1}t)$  be the transition matrix for time  $t$  corresponding to the Markov chain described by  $Q_{i,i+1}$ .

- Let  $I_i$  be an indicator matrix of states in  $\llbracket \varphi_i \rrbracket_M$ :

$$I_i(j, k) := \begin{cases} 1, & \text{if } s_j = s_k \in \llbracket \varphi_i \rrbracket_M, \\ 0, & \text{otherwise.} \end{cases}$$

This matrix is used to filter out states not satisfying  $\varphi_i$  at certain times.

- Finally, let  $E_n$  be a matrix obtained from  $Q$  which treats states in  $\llbracket \varphi_n \rrbracket_M$  as absorbing states:

$$E_n(j, k) := \begin{cases} 0, & \text{if } s_j \in \llbracket \varphi_n \rrbracket_M, \\ Q(j, k), & \text{otherwise.} \end{cases}$$

This matrix is used at the end of the formula to “collect” all the probability mass of paths which have satisfied the path formula  $\varphi_1 \mathbf{U}_{[a_1, b_1]} \varphi_2 \mathbf{U}_{[a_2, b_2]} \cdots \mathbf{U}_{[a_{n-1}, b_{n-1}]} \varphi_n$ . Also, let  $F_n(t) := \exp(E_n t)$  be the transition matrix for time  $t$  corresponding to the Markov chain described by  $E_n$ .

It is not hard to see that the probability of paths starting in  $M$  according to the initial probability  $\pi$  which satisfy the path formula

$$\varphi_1 \mathbf{U}_{[a_1, b_1]} \varphi_2 \mathbf{U}_{[a_2, b_2]} \cdots \mathbf{U}_{[a_{n-1}, b_{n-1}]} \varphi_n,$$

as defined in (6) above, has the expression:

$$f(t) = \pi^\top \cdot P_{0,0}(a_1) \cdot I_0 \cdot P_{0,1}(b_1 - a_1) \cdot I_1 \cdot P_{1,1}(a_2 - b_1) \cdot I_1 \cdot P_{1,2}(b_2 - a_2) \cdot I_2 \cdots I_{n-1} \cdot F_n(b_n - a_n) \cdot I_n \cdot \mathbf{1}. \quad (8)$$

All matrix functions parameters  $(a_1, b_1 - a_1, a_2 - b_1, \dots)$  are of the form  $\alpha t + \beta$ , for  $\alpha, \beta \in \mathbb{Q}$ . By Lemma 2 we get that all entries implied in the product at (8) are exponential polynomials with algebraic coefficients. As exponential polynomials with algebraic coefficients are closed under product and sum, we get that  $f(t)$  is an exponential polynomial with algebraic coefficients, which we can compute algorithmically, by using classic representation methods of algebraic numbers (see [5, Section 4.2]).

Therefore, by our result - Theorem 4, we get that assuming Schanuel’s Conjecture we can decide if there exists  $t \in [a, b]$  such that  $f(t) - c > 0$ , as  $f(t) - c$  is a real-valued exponential polynomial with algebraic coefficients. Therefore, under Schanuel’s Conjecture, we can also model check  $\varphi$  in the case when

$$\varphi = \exists t \in [a, b] \cdot \mathbb{P}_{>c}(\varphi_1 \mathbf{U}_{[a_1, b_1]} \varphi_2 \mathbf{U}_{[a_2, b_2]} \cdots \mathbf{U}_{[a_{n-1}, b_{n-1}]} \varphi_n).$$

In conclusion, we have covered all forming rules of state formulas in  $\text{PCSL}_1$ , and proved by structural induction that Schanuel’s Conjecture implies the existence of a model checking procedure for  $\text{PCSL}_1$ .

► **Remark 7.** Let us briefly discuss the assumption (7). We assumed that the parametric terms  $a_1, b_1, \dots, a_{n-1}, b_{n-1}$  in  $\{t\}$ , which are linear functions in  $t$ , satisfy for all  $t \in [a, b]$ :  $a_1 \leq b_1 \leq \dots \leq a_{n-1} \leq b_{n-1}$ . Let us write them explicitly as  $a_i = x_i(t), b_i = y_i(t)$ , for  $i = 1, \dots, n - 1$ . First, it is easy to see that, as all parametric terms are linear functions in  $t$  with rational coefficients, there is some maximal interval  $[c, d] \subseteq [a, b]$ , with  $t, c \in \mathbb{Q}$ , such that all  $0 \leq x_1(t) \leq y_1(t), 0 \leq x_2(t) \leq y_2(t), \dots, 0 \leq x_{n-1}(t) \leq y_{n-1}(t)$  hold for all  $t \in [c, d]$ , and at least one of them doesn’t hold for any  $t \in [a, b] \setminus [c, d]$ . Then, we can just seek some value of  $t$  in  $[c, d]$  that satisfies the formula, as outside this interval the formula is not syntactically valid. Now, in order to be able to also assume the inequalities  $y_i(t) \leq x_{i+1}(t)$ , we can split the interval  $[c, d]$  in intervals in which either  $y_i(t) \leq x_{i+1}(t)$ , or  $y_i(t) \geq x_{i+1}(t)$  holds, and deal with all possible cases separately. The full details for this part are rather technical, and mostly follow the technique in [2].

## 4.2 Hardness of the model checking problem for $\text{PCSL}_1$

To show hardness of deciding the model checking problem for  $\text{PCSL}_1$ , therefore showing hardness of deciding the model checking problem for  $\text{PCSL}$  implicitly, we proceed in two steps.

First, we introduce a hard decision problem - *the Positivity Problem for matrix exponentials*. This is a hard problem as it is inter-reducible with the Positivity Problem for exponential polynomials (by simple algebraic manipulation, see [3] for details). We have discussed in

subsection 3.3 why the Positivity Problem for exponential polynomials is a hard problem: it would imply decidability of the Non-negativity Problem for exponential polynomials, which is currently open [3]. We show that the Positivity Problem for matrix exponentials is reducible to a decision problem regarding CTMC properties - *the Threshold Problem for continuous-time Markov chains*.

Second, we show that PCSL<sub>1</sub> model checking decidability implies decidability of the Threshold Problem for continuous-time Markov chains. Therefore, decidability of the model checking problem for PCSL<sub>1</sub> implies decidability of the Positivity Problem for matrix exponentials. This stands as hardness evidence for a PCSL<sub>1</sub> model checking decision procedure, and for a PCSL model checking decision procedure as well, because PCSL<sub>1</sub> is a fragment of PCSL.

#### 4.2.1 Reduction of a Hard Problem to the Threshold Problem for Continuous-Time Markov Chains

We introduce below the Threshold Problem for continuous-time Markov chains and the Positivity Problem for matrix exponentials.

► **Definition 8** (Threshold Problem for continuous-time Markov chains).  $I = (\langle \mathbf{u}, \mathbf{R}, \mathbf{v} \rangle, \langle a, b \rangle)$  is an instance of the **Threshold Problem for continuous-time Markov chains**, where  $\mathbf{u} \in \mathbb{Q}^k$  is a stochastic vector<sup>7</sup>,  $\mathbf{v} \in \{0, 1\}^k$ ,  $\mathbf{R} \in \mathbb{Q}^{k \times k}$  is a rate matrix (for some  $k \in \mathbb{N}$ ), and  $a, b \in \mathbb{Q}$  such that  $0 \leq a \leq b$ . We want to answer the question: does there exist some real  $t \in [a, b]$  such that  $\mathbf{u}^\top e^{\mathbf{R}t} \mathbf{v} > \frac{1}{2}$ ?

Intuitively, in the Threshold Problem for continuous-time Markov chains,  $\mathbf{u}$  represents the initial distribution and  $\mathbf{R}$  represents the rate matrix of a CTMC. Then, we ask if at some moment during a given interval  $[a, b]$  the probability of being in a state from a given set, that is described by 1-entries of  $\mathbf{v}$ , is greater than  $\frac{1}{2}$ .

► **Definition 9** (Positivity Problem for matrix exponentials).  $I = (\langle \mathbf{u}, \mathbf{A}, \mathbf{v} \rangle, \langle a, b \rangle)$  is an instance of the **Positivity Problem for matrix exponentials**, where  $\mathbf{u}, \mathbf{v} \in \mathbb{Q}^k$ ,  $\mathbf{A} \in \mathbb{Q}^{k \times k}$  (for some  $k \in \mathbb{N}$ ), and  $a, b \in \mathbb{Q}^+$ , with  $0 \leq a \leq b$ . We want to answer the question: does there exist some real  $t \in [a, b]$  such that  $\mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v} > 0$ ?

Note that the Positivity Problem for matrix exponentials can be seen as a generalization of the Threshold Problem for continuous-time Markov chains, both because the former has much more general instances, but also because its decidability implies decidability of the latter. To see this, let  $(\langle \mathbf{u}, \mathbf{R}, \mathbf{v} \rangle, \langle a, b \rangle)$  be an instance of the Threshold Problem for continuous-time Markov chains, then  $e^{\mathbf{R}t}$  is a stochastic matrix<sup>8</sup>, so  $\mathbf{u}^\top e^{\mathbf{R}t}$  is a stochastic row vector, therefore  $\mathbf{u}^\top e^{\mathbf{R}t} \mathbf{1} = 1$ . Then, we could obtain a decision procedure for this problem by applying a decision procedure for the Positivity Problem for matrix exponentials on instance  $(\langle \mathbf{u}, \mathbf{R}, \mathbf{v} - \frac{1}{2} \mathbf{1} \rangle, \langle a, b \rangle)$ :

$$\begin{aligned} \exists t \in [a, b] \text{ such that } \mathbf{u}^\top e^{\mathbf{R}t} \mathbf{v} > \frac{1}{2} &\Leftrightarrow \\ \exists t \in [a, b] \text{ such that } \mathbf{u}^\top e^{\mathbf{R}t} (\mathbf{v} - \frac{1}{2} \mathbf{1}) > 0. \end{aligned}$$

<sup>7</sup> Has positive entries that sum up to 1.

<sup>8</sup> Its rows are probability distributions.

Furthermore, as  $e^{\mathbf{R}t}$  is a stochastic matrix, we would expect the Threshold Problem for continuous-time Markov chains to be considerably easier, as eigenvalues of stochastic matrices are well-behaved<sup>9</sup>.

Surprisingly, we show that the Positivity Problem for matrix exponentials is reducible to the Threshold Problem for continuous-time Markov chains, thus making the two decision problems equivalently hard.

► **Theorem 10.** *The Positivity Problem for matrix exponentials is reducible to the Threshold Problem for continuous-time Markov chains.*

**Proof.** The full proof is given in the appendix. Using algebraic manipulations, we construct a rate transition matrix  $\mathbf{O}$  and some vectors  $\mathbf{u}_1$  and  $\mathbf{v}_3$ , and then a rate transition matrix  $\mathbf{R}$ , a stochastic vector  $\tilde{\mathbf{u}}$  and a vector  $\tilde{\mathbf{v}}$  with only 0 and 1 entries such that

$$\begin{aligned} \exists t \in [a, b] \text{ such that } \mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v} > 0 &\iff \\ \exists t \in [a, b] \text{ such that } \mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_3 > \frac{1}{2} &\iff \\ \exists t \in [a, b] \text{ such that } \tilde{\mathbf{u}}^\top e^{\mathbf{R}t} \tilde{\mathbf{v}} > \frac{1}{2}. & \end{aligned} \quad (9)$$

However, a decision procedure for the Threshold Problem for continuous-time Markov chains would specifically allow us to answer queries such as  $\exists t \in [a, b]$  such that  $\tilde{\mathbf{u}}^\top e^{\mathbf{R}t} \tilde{\mathbf{v}} > \frac{1}{2}$ , therefore it would give a decision procedure for the Positivity Problem for matrix exponentials as well. ◀

#### 4.2.2 Expressing the Threshold Problem for Continuous-Time Markov Chains in PCSL<sub>1</sub>

Let  $I = (\langle \mathbf{u}, \mathbf{R}, \mathbf{v} \rangle, \langle a, b \rangle)$  be an instance of the Threshold Problem for continuous-time Markov chains. Recall that  $\mathbf{u} \in \mathbb{Q}^k$  is a stochastic vector,  $\mathbf{R} \in \mathbb{Q}^{k \times k}$  is a rate matrix,  $\mathbf{v} \in \{0, 1\}^k$ , and  $0 \leq a \leq b$  are rationals, and we want to answer whether there exists  $t \in [a, b]$  such that  $\mathbf{u}^\top e^{\mathbf{R}t} \mathbf{v} > \frac{1}{2}$ .

Let  $M$  be the continuous-time Markov chain corresponding to rate matrix  $\mathbf{R}$ , with initial probability distribution  $\pi_0 := \mathbf{u}$ , and with state set  $S$  such that  $|S| = k$ . The probability distribution over states at time  $t$  is given by  $\mathbf{u}^\top e^{\mathbf{R}t}$ . Therefore, as  $\mathbf{v} \in \{0, 1\}^k$ , we can see the expression  $\mathbf{u}^\top e^{\mathbf{R}t} \mathbf{v}$  as summing up the probability distribution at time  $t$  of states corresponding to 1-entries in  $\mathbf{v}$ .

Let the states from  $S$  that correspond to 1-entries of  $\mathbf{v}$  be  $S' = \{s_1, \dots, s_i\}$ . If  $S'$  is empty, then  $\mathbf{v} = \mathbf{0}$ , and we have  $\mathbf{u}^\top e^{\mathbf{R}t} \mathbf{v} = 0$ , so the Threshold Problem for continuous-time Markov chains instance is a negative instance, and we trivially have:

$$\exists t \in [a, b] \text{ such that } \mathbf{u}^\top e^{\mathbf{R}t} \mathbf{v} > \frac{1}{2} \iff M \models s \wedge \neg s, \text{ for some } s \in S. \quad (10)$$

Otherwise, if  $S'$  is not empty, we get:

$$\begin{aligned} \exists t \in [a, b] \text{ such that } \mathbf{u}^\top e^{\mathbf{R}t} \mathbf{v} > \frac{1}{2} &\iff \\ M, \pi_0 \models \exists t \in [a, b] \cdot \mathbb{P}_{>1/2}(\text{true} \mathbf{U}_{[t,t]}(s_1 \vee \dots \vee s_i)). & \end{aligned} \quad (11)$$

<sup>9</sup> Standard linear algebra results imply that 1 is always an eigenvalue of any stochastic matrix, and all the eigenvalues have absolute value less or equal to 1.

Thus, for any instance  $I$  of the Threshold Problem for continuous-time Markov chains, there exists a continuous-time Markov chain  $M$  and some PCSL<sub>1</sub> formula  $\varphi$  such that  $I$  is a yes-instance of the Threshold Problem for continuous-time Markov chains if and only if the PCSL<sub>1</sub> satisfaction relation  $M \models \varphi$  holds.

In conclusion, a PCSL<sub>1</sub> model checking procedure (that decides if PCSL<sub>1</sub> statements of the form  $M \models \varphi$  hold) would yield the existence of a decision procedure for the Threshold Problem for continuous-time Markov chains. As we have shown in Section 4.2.1, this would imply the decidability of the Positivity Problem for matrix exponentials, which, as discussed, is a hard problem and is not currently known to be decidable. Therefore, the unconditional decidability of PCSL<sub>1</sub> model checking, and thus of PCSL model checking, seems to be a hard problem.

## 5 Conclusions

### 5.1 Overview

We introduced PCSL, a powerful parametric logic for formally expressing temporal properties of continuous-time Markov chains. We investigated the model checking problem of our logic, proving that its unconditional decidability is a hard problem, and showed that Schanuel's Conjecture implies decidability of the model checking problem for an expressive fragment of PCSL. The last result relies on a technical proof that Schanuel's Conjecture implies the decidability of the Positivity Problem for exponential polynomials, which is an important achievement in the field.

The logic could have simply been extended to allow operators of the form  $\text{Pr}_{\&c}$ , where  $\&$  could be any of  $\leq, \geq, <, >, =$ , or  $\neq$ , instead of only allowing  $\text{Pr}_{>c}$ . All our results would still hold, as [4] proves the conditional decidability of verifying whether exponential polynomials are equal to a given constant in some given interval, and this together with our proofs would suffice for obtaining the same consequences about model checking PCSL. We restricted our attention to PCSL using only operators of the form  $\text{Pr}_{>c}$ , which makes our arguments more concise, while presenting all the fundamental mathematical problems we have tackled.

### 5.2 Future Work

We propose two main directions for future work on our project.

#### 5.2.1 General Conditional Decidability of PCSL

We have shown decidability of PCSL<sub>1</sub> model checking assuming Schanuel's Conjecture, by proving conditional decidability of the Positivity Problem for exponential polynomials. In general, the decidability of model checking PCSL <sub>$n$</sub>  reduces to the decidability of the Positivity Problem for exponential polynomials in  $n$  variables. In fact, we found out that using the polynomial resultant for eliminating variables in the two variable case reduces the decidability of the model checking problem for PCSL<sub>2</sub> to a purely algebraic problem. We believe that decidability of model checking PCSL <sub>$n$</sub>  also follows assuming Schanuel's Conjecture, and therefore we propose seeking a general proof for conditional decidability of model checking PCSL.

## 5.2.2 Practical Model Checking of PCSL

We have mainly been concerned with the fundamental problem of PCSL decidability, however in practice we expect that the malicious cases we encountered theoretically should not represent too much of a risk in real-life applications. As we have seen interesting classes of properties that are expressible in PCSL, it is worthy to further investigate the practical aspects of model checking PCSL and possible optimizations for an actual procedure.

---

### References

- 1 S Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Information Processing Letters*, 115(2):155–158, 2015.
- 2 Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, 2000.
- 3 Paul Bell, Jean-Charles Delvenne, Raphael Jungers, and Vincent D. Blondel. The continuous Skolem-Pisot problem: On the complexity of reachability for linear ordinary differential equations. *Theoretical Computer Science*, 411(40–42):3625–3634, 2010.
- 4 Ventsislav Chonev, Joel Ouaknine, and James Worrell. On the Skolem problem for continuous linear dynamical systems. *43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 100:1–100:13, 2016.
- 5 Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1993.
- 6 Paul M. Cohn. *Basic Algebra: Groups, Rings and Fields*. Springer, second edition, 2004.
- 7 Bettina Just. Integer relations among algebraic numbers. In *International Symposium on Mathematical Foundations of Computer Science*, pages 314–320. Springer, 1989.
- 8 Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. *Proc. 23rd International Conference on Computer Aided Verification (CAV’11), volume 6806 of LNCS, pages 585–591, Springer, 2011.*
- 9 Angus Macintyre and Alex J Wilkie. On the decidability of the real exponential field. In (ed. Piergiorgio Odifreddi) *Kreiseliana: About and Around Georg Kreisel.*, 1996.
- 10 Ivan Niven. *Irrational Numbers*. The Mathematical Association of America, fifth edition, 2005.
- 11 Paul S Wang. Factoring multivariate polynomials over algebraic number fields. *Mathematics of Computation*, 30(134):324–336, 1976.
- 12 Boris Zilber. Exponential sums equations and the Schanuel conjecture. *Journal of the London Mathematical Society*, 65(1):27–44, 2002.

## A Proof of Theorem 4

Let  $f(t) = \sum_{j=1}^m P_j(t)e^{\lambda_j t}$ , together with the interval  $[c, d]$  be an instance of the Positivity Problem for exponential polynomials. Let  $\mathbb{K}$  be the number field generated by the coefficients of polynomials  $P_1, \dots, P_m$  and by  $\lambda_1, \dots, \lambda_m$  over  $\mathbb{Q}$ . We can algorithmically determine a basis  $\{a_1, \dots, a_r\}$  over  $\mathbb{Q}$  of the real parts of  $\lambda_i$ ’s, and a basis  $\{b_1, \dots, b_s\}$  over  $\mathbb{Q}$  of the imaginary parts of  $\lambda_i$ ’s [7].

Without loss of generality, assume that all real and imaginary parts of  $\lambda_1, \dots, \lambda_m$  can be written as linear combinations of  $\{a_1, \dots, a_r\}$ , respectively of  $\{b_1, \dots, b_s\}$  that use integer coefficients instead of rational coefficients (this follows as we can pick a suitable  $N \in \mathbb{N}$  and write  $f_1(t) := f(Nt) = \sum_{j=1}^m P_j(Nt)e^{(\lambda_j N)t}$ ).

It follows that we can write  $f(t)$  as a polynomial in the field of Laurent polynomials  $\mathcal{R}$ , with multiplicative units the non-zero monomials in  $y_1, \dots, y_r, z_1, \dots, z_s$ :

$$\mathcal{R} := \mathbb{K}[x, y_1, y_1^{-1}, \dots, y_r, y_r^{-1}, z_1, z_1^{-1}, \dots, z_s, z_s^{-1}].$$



We write  $f(t) = P(t, e^{a_1 t} \dots, e^{a_r t}, e^{b_1 t} \dots, e^{b_s t})$ , where  $P$  is a polynomial with non-negative power in its first argument, and with any integer power in the others.

Being a localisation of the polynomial ring  $\mathbb{A} := \mathbb{K}[x, y_1, \dots, y_r, z_1, \dots, z_s]$ ,  $\mathcal{R}$  is a unique factorisation domain (this is a standard result; see [6, Theorem 10.3.7]) and has an effective procedure for factoring it into irreducible polynomials [11]. We extend the conjugation over  $\mathcal{R}$ , by defining a ring automorphism  $(\cdot)^*$  which acts on  $P$  to yield  $P^*$  as below:

$$\begin{aligned} P(x, y_1, \dots, y_r, z_1, \dots, z_s) &= \sum_i \alpha_i x^{\beta_i} y_1^{\gamma_{1,i}} \dots y_r^{\gamma_{r,i}} z_1^{\delta_{1,i}} \dots z_s^{\delta_{s,i}} \\ P^*(x, y_1, \dots, y_r, z_1, \dots, z_s) &:= \sum_i \overline{\alpha_i} x^{\beta_i} y_1^{\gamma_{1,i}} \dots y_r^{\gamma_{r,i}} z_1^{-\delta_{1,i}} \dots z_s^{-\delta_{s,i}}. \end{aligned} \quad (12)$$

The motivation behind this definition is that for  $f(t) = P(t, e^{a_1 t} \dots, e^{a_r t}, e^{b_1 t} \dots, e^{b_s t})$  we have  $f(\bar{t}) = P^*(t, e^{a_1 t} \dots, e^{a_r t}, e^{b_1 t} \dots, e^{b_s t})$ . For such a real-valued  $f$ , we have  $P = P^*$ .

As  $(\cdot)^*$  is a ring automorphism over the unique factorization domain  $\mathcal{R}$ , we get that if a polynomial  $Q$  in  $\mathcal{R}$  divides  $P$ , then there exists some  $R$  in  $\mathcal{R}$  such that  $P = QR$ , so  $P^* = Q^*R^*$ . Therefore,  $Q^*$  also divides  $P^*$ , but as  $P = P^*$  we have that  $Q^*$  divides  $P$ . Therefore, factors of  $P$  come in  $*$ -conjugated pairs.

We will use Schanuel's Conjecture through the following result, which follows from it by using the concept of *resultant* of two polynomials and basic algebraic manipulations [4]:

► **Lemma 11.** *Let  $r, s$  be non-negative integers, and let  $\{a_1, \dots, a_r\}$  and  $\{b_1, \dots, b_s\}$  be  $\mathbb{Q}$ -linearly independent sets of algebraic numbers. Let  $P, Q \in \mathcal{R}$  be polynomials with algebraic coefficients that are coprime in  $\mathcal{R}$ . Then the following equations have no common solution  $t \in \mathbb{R} \setminus \{0\}$ :  $P(t, e^{a_1 t}, \dots, e^{a_r t}, e^{b_1 t}, \dots, e^{b_s t}) = 0, Q(t, e^{a_1 t}, \dots, e^{a_r t}, e^{b_1 t}, \dots, e^{b_s t}) = 0$*

We say that two polynomials  $P, Q \in \mathcal{R}$  are *associates* if  $Q = \mathbf{z}^{\mathbf{u}} P$ , where  $\mathbf{z}^{\mathbf{u}}$  is a monomial in  $z_1, \dots, z_s$  (note that the associate relation is symmetric by the definition of  $\mathcal{R}$ ).

We have seen that we can write the exponential polynomial as a Laurent polynomial in  $\mathcal{R}$ :  $f(t) = P(t, e^{a_1 t} \dots, e^{a_r t}, e^{b_1 t} \dots, e^{b_s t})$ . As  $f(t)$  is real-valued, it can be factored in irreducible polynomials from  $\mathbb{K}$  that are either real valued, or come with their conjugate pair in the factorization of  $f$ . Therefore, there exist some irreducible polynomials  $Q_1, \dots, Q_k$  in  $\mathcal{R}$  that come in pair with their conjugates and some irreducible polynomials  $R_1, \dots, R_l$  in  $\mathcal{R}$  and positive integers  $\alpha_1 \dots \alpha_k, \beta_1, \dots, \beta_l$  such that we can write  $P = \prod_{i=1}^k (Q_i Q_i^*)^{\alpha_i} \cdot \prod_{j=1}^l R_j^{\beta_j}$ .

Define the functions  $u_i(t) := Q_i(t, e^{a_1 t} \dots, e^{a_r t}, e^{b_1 t} \dots, e^{b_s t})$ , which are not real-valued and come in pairs with their conjugates; and  $v_j(t) := R_j(t, e^{a_1 t} \dots, e^{a_r t}, e^{b_1 t} \dots, e^{b_s t})$ , which are real-valued. Let  $w_i(t) := u_i(t) \overline{u_i(t)}$ , for  $t \in \mathbb{R}$ . Then  $f(t) = \prod_{i=1}^k w_i(t)^{\alpha_i} \cdot \prod_{j=1}^l v_j(t)^{\beta_j}$ , where functions  $w_1, \dots, w_k, v_1, \dots, v_l$  are real-valued, analytic functions.

Recall that the decision problem asks whether  $f(t)$  is strictly positive for some value of  $t$  in  $[c, d]$ , for some given  $c, d \in \mathbb{Q}$ . If  $f(t)$  has a trivial form (i.e., if  $f(t)$  is a polynomial in  $\mathbb{K}[x]$ , with no exponentials) we can easily decide this problem by approximating its roots in  $[c, d]$  and classifying the sign on  $f$  between them (using, for example, the Sturm sequence of the polynomial). Otherwise, by Lindemann-Weierstrass Theorem (see [2]), we get that  $f(t) = \sum_{j=1}^m P_j(t) e^{\lambda_j t}$  can be 0 in an algebraic point  $t$  if and only if  $P_j(t) = 0$ , for all  $j \in \{1, \dots, m\}$ . We can use standard factorization algorithms for computing common algebraic roots of the  $P_j$  polynomials. If there is any common algebraic root  $t^*$ , then  $f(t^*) = 0$ . As all the derivatives of  $f$  are also exponential polynomials, we can determine in a similar way the smallest  $M$  such that the  $M^{\text{th}}$  derivative of  $f$  is non-zero at  $t^*$ . By Taylor's Theorem, for any  $t$ , there exists some  $\epsilon$  between  $t$  and  $t^*$  such that  $f(t) = f(t^*) + \frac{f^{(M)}(\epsilon)}{M!} (t - t^*)^M$ .



If  $M$  is odd, then  $f$  changes sign at  $t^*$ . Otherwise, there is no sign change at  $t^*$ . We can therefore deal with common roots of all polynomials  $P_j(t)$ . Then, we can assume without loss of generality that the  $P_j$  polynomials have no common root in  $(c, d)$ .

We can trivially get rid of the case when all the polynomials  $P_j(t)$  have  $c$  as a common root by dividing them by the highest power of  $(t - c)$  that divides all of them. This can be done safely, as changing signs at  $c$  does not make sense within the interval  $[c, d]$  and as  $(t - c)$  is always positive for  $t > c$ , so this division does not affect potential sign changes of  $f(t)$  anywhere in  $[c, d]$ . The same holds for  $d$ . Therefore, we can assume without loss of generality that the  $P_j$  polynomials have no common root in  $[c, d]$ .

As  $P_j$ 's cannot all be 0 at the same (algebraic) point, by Lindemann-Weierstrass Theorem we get that  $f$  is non-zero in any rational point. In particular,  $f(c)$  and  $f(d)$  are non-zero, so we can use any standard approximation procedure until we can compare  $f(c), f(d)$  with 0 (for example, see [2, Lemma 2]). If either of  $f(c), f(d)$  is strictly greater than 0, then we are done. Therefore, assume from now on that both  $f(c)$  and  $f(d)$  are strictly negative.

No two different functions from  $w_1(t), \dots, w_k(t), v_1(t), \dots, v_l(t)$  can have a common real zero in  $[c, d]$ , as this would imply that two of the polynomials  $Q_1, \dots, Q_k, R_1, \dots, R_l$  have a common solution of the form  $(t, e^{a_1 t}, \dots, e^{a_r t}, e^{b_1 i t}, \dots, e^{b_s i t})$ , which contradicts Lemma 11, as all the listed polynomials are irreducible (and not associates) and as  $t = 0$  cannot be a solution of such functions, because of Lindemann-Weierstrass Theorem (as we have dealt with algebraic roots above, which are common roots of all  $P_j$ 's). This means that it is enough to decide whether there exists some function among  $w_i$ 's and  $v_j$ 's with odd exponent in  $f$  that changes its sign in  $[c, d]$ , as we can just consider the one which changes its sign at the least  $\tau \in [c, d]$ . If we let this function be  $g$ , we have  $g(\tau) = 0$  and we then know that no other function among the  $w_i$ 's and  $v_j$ 's has a solution at  $\tau$ , so there is some interval  $I = (\tau - \epsilon, \tau + \epsilon)$  such that  $g$  has exactly opposite signs on  $(\tau - \epsilon, \tau)$  and  $(\tau, \tau + \epsilon)$ , and no other function equals 0 on  $I$ . It is easy then to see that deciding whether  $f(t) > 0$  for some  $t \in [c, d]$  is equivalent to deciding whether any of the real-valued functions with odd exponent in  $f$  among  $w_i$ 's and  $v_j$ 's changes its sign in  $[c, d]$ . This follows easily as the real-valued functions with even exponents are always non-negative and cannot change sign. Therefore, we can assume without loss of generality that all exponents are 1, so  $f(t) = \prod_{i=1}^k w_i(t) \cdot \prod_{j=1}^l v_j(t)$ .

In general, classical numerical algorithms should work in most of the cases for our decision problem. However, when an exponential polynomial has a tangential zero, detecting it through such procedures requires infinite precision. The difficulty in solving our problem comes exactly from dealing with cases of such tangential zeros.

Let us now see how to decide if any of the  $v_j$ 's or  $w_i$ 's changes its sign on  $[c, d]$ .

**Case 1:** Decide if some  $v_j$  changes sign on  $[c, d]$ .

Recall that  $v_j(t) = R_j(t, e^{a_1 t}, \dots, e^{a_r t}, e^{b_1 t}, \dots, e^{b_s t})$  is a real-valued function ( $R_j = R_j^*$ ). Also, recall that we ruled out the case of  $f(c) = 0$ , so we can approximate arbitrarily close  $v_j(c)$  to decide if it is positive or negative. Assume without loss of generality that  $v_j(c) < 0$ . Then, as  $v_j(d) \neq 0$ , if we get by approximating it that  $v_j(d) > 0$ , we are trivially done, so assume that  $v_j(d) < 0$ . We want to decide if there exists some  $t \in [c, d]$  such that  $v_j(t) > 0$ .

In this case, we claim that deciding if there is some zero of  $v_j$  in  $[c, d]$  is equivalent to deciding if it changes sign, i.e., the equations  $v_j(t) = 0, v_j'(t) = 0$  have no common solution. So, if  $v_j(t) = 0$  for some  $t \in [c, d]$ , there is a least such  $t$  (by continuity on a bounded interval), and it is easy to see that, if  $v_j'(t) \neq 0$ , we get that  $v_j'$  changes sign at  $t$ , from negative to positive.

## 7:16 Parametric Continuous Stochastic Logic

To see that  $v_j(t) = 0, v'_j(t) = 0$  have no common solution, write  $v'_j(t)$  as a polynomial in  $t, e^{a_1 t}, \dots, e^{a_r t}, e^{b_1 t}, \dots, e^{b_s t}$  and get by a simple degree chasing argument that it is coprime with the polynomial  $R_j(t, e^{a_1 t}, \dots, e^{a_r t}, e^{b_1 t}, \dots, e^{b_s t}) = v_j(t)$ , thus getting a contradiction with Lemma 11 (see Type-2 polynomial argument in [4] for details).

In conclusion, we can use the decision procedure described in [4] for zero finding for the purpose of deciding sign changing.

**Case 2:** Decide if some  $w_i$  changes sign on  $[c, d]$ .

Recall that, for  $t \in \mathbb{R}$ :

$$w_i(t) = u_i(t) \overline{u_i(t)}.$$

Note that  $w_i(t)$  cannot change sign at any real  $t$ , therefore this case is trivial, as  $w_i(t) \geq 0$ .

In conclusion, the Positivity Problem for exponential polynomials is decidable assuming Schanuel's Conjecture.

### B Proof of Theorem 10

**Proof.** Let  $(\langle \mathbf{u}, \mathbf{A}, \mathbf{v} \rangle, \langle a, b \rangle)$  be an instance of the Positivity Problem for matrix exponentials. Let  $\mathbf{D} \in \mathbb{Q}^{k \times k}$  be a diagonal matrix such that  $\mathbf{D} = \text{diag}(d_1, \dots, d_k)$ , where  $d_i := 1$  if  $\mathbf{v}_i = 0$  and  $d_i := v_i$ , otherwise. Note that  $d_i \geq 0$  for any  $i$ .

Now, by letting  $\bar{\mathbf{v}} \in \mathbb{Q}^k$  be such that if  $\mathbf{v}_i = 0$  then  $\bar{\mathbf{v}}_i := 0$ , and otherwise  $\bar{\mathbf{v}}_i := 1$ , it is clear that  $\mathbf{v} = \mathbf{D}\bar{\mathbf{v}}$ . By denoting  $\mathbf{B} := \mathbf{D}^{-1}\mathbf{A}\mathbf{D}$  and  $\bar{\mathbf{u}} := \mathbf{D}^\top \mathbf{u}$ , we get:

$$\mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v} = \mathbf{u}^\top \mathbf{D} \mathbf{D}^{-1} e^{\mathbf{A}t} \mathbf{D} \bar{\mathbf{v}} = \mathbf{u}^\top \mathbf{D} e^{\mathbf{D}^{-1} \mathbf{A} \mathbf{D} t} \bar{\mathbf{v}} = \bar{\mathbf{u}}^\top e^{\mathbf{B}t} \bar{\mathbf{v}} \quad (13)$$

We adopt the following construction and map used in [1] for a related reduction in the discrete case: let  $\mathbf{P} \in \mathbb{Q}^{2k \times 2k}$  be a matrix obtained by replacing each entry  $b_{ij}$  of  $\mathbf{B}$  by the symmetric matrix  $\begin{bmatrix} p_{ij} & q_{ij} \\ q_{ij} & p_{ij} \end{bmatrix}$ , where  $p_{ij} = \max\{b_{ij}, 0\}$  and  $q_{ij} = \max\{-b_{ij}, 0\}$ . Let  $\rho$  be a map which sends  $\begin{bmatrix} a & b \\ b & a \end{bmatrix}$  to  $a - b$  and, applied to a matrix which can be partitioned in blocks of the form before, sends each block to the according difference. It is easy to check that  $\rho$  is a (surjective) homomorphism from the ring of matrices in  $\mathbb{Q}^{2k \times 2k}$  (which can be partitioned in  $2 \times 2$  blocks of the form  $\begin{bmatrix} a & b \\ b & a \end{bmatrix}$ ) to the ring of matrices in  $\mathbb{Q}^{k \times k}$ .

By looking at the power series expansion of the matrix exponential  $e^{\mathbf{X}}$ , because of its convergence we get  $e^{\rho(\mathbf{M})} = \rho(e^{\mathbf{M}})$ . Recall (13):  $\mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v} = \bar{\mathbf{u}}^\top e^{\mathbf{B}t} \bar{\mathbf{v}}$ . As  $\rho(\mathbf{P}) = \mathbf{B}$ , we get:

$$\mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v} = \bar{\mathbf{u}}^\top e^{\rho(\mathbf{P})t} \bar{\mathbf{v}} = \bar{\mathbf{u}}^\top \rho(e^{\mathbf{P}t}) \bar{\mathbf{v}}. \quad (14)$$

Write  $\bar{\mathbf{u}} := (\alpha_1, \dots, \alpha_k)^\top$  and  $\bar{\mathbf{v}} := (\beta_1, \dots, \beta_k)^\top$ . Given  $w_1, \dots, w_k \in \mathbb{Q}$ , define  $\mathbf{x} \in \mathbb{Q}^{2k}$  by

$$\mathbf{x} := (\alpha_1 + w_1, w_1, \alpha_2 + w_2, w_2, \dots, \alpha_k + w_k, w_k)^\top.$$

Let us also define  $\mathbf{y} \in \mathbb{Q}^{2k}$  by

$$\mathbf{y} := (\beta_1, -\beta_1, \beta_2, -\beta_2, \dots, \beta_k, -\beta_k)^\top.$$

▷ **Claim 12.** For all  $w_1, \dots, w_k \in \mathbb{Q}$  it holds that  $\mathbf{x}^\top e^{\mathbf{P}t} \mathbf{y} = \mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v}$  for all  $t \in \mathbb{R}$ .

Proof. Let us fix a positive real  $t$ . Denote the elements of  $e^{\mathbf{A}t} =: \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ e_{k1} & e_{k2} & \dots & e_{kk} \end{bmatrix}$  and,

as  $e^{\mathbf{A}t} = \rho(e^{\mathbf{P}t})$ , we can write:

$$e^{\mathbf{P}t} = \begin{bmatrix} f_{11} & g_{11} & f_{12} & g_{12} & \dots & f_{1k} & g_{1k} \\ g_{11} & f_{11} & g_{12} & f_{12} & \dots & g_{1k} & f_{1k} \\ \vdots & \vdots & \ddots & \vdots & & & \\ f_{k1} & g_{k1} & f_{k2} & g_{k2} & \dots & f_{kk} & g_{kk} \\ g_{k1} & f_{k1} & g_{k2} & f_{k2} & \dots & g_{kk} & f_{kk} \end{bmatrix}, \quad (15)$$

where  $f_{ij} - g_{ij} = e_{ij}$  for all  $i, j$ . Then, we get:

$$\begin{aligned} \mathbf{x}^\top e^{\mathbf{P}t} \mathbf{y} &= \sum_{i=1}^k \sum_{j=1}^k ((\alpha_i + w_i) f_{ij} \beta_j - (\alpha_i + w_i) g_{ij} \beta_j + w_i g_{ij} \beta_j - w_i f_{ij} \beta_j) \\ &= \sum_{i=1}^k \sum_{j=1}^k (\alpha_i (f_{ij} - g_{ij}) \beta_j) = \sum_{i=1}^k \sum_{j=1}^k (\alpha_i e_{ij} \beta_j) \\ &= \mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v}. \end{aligned}$$

As  $t \in \mathbb{R}$  was arbitrary, we get that the claim holds.  $\square$

Choose  $w_i$ 's such that  $\mathbf{x}$  has only positive entries:  $w_1 = \dots = w_k := \max(|\alpha_1|, \dots, |\alpha_k|) + 1$ . Let  $S > 0$  be the sum of  $\mathbf{x}$ 's entries; and let  $\mathbf{z} := \frac{1}{S} \mathbf{x}$ . Then, by Claim 12, we have:  $\mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v} > 0 \iff \mathbf{x}^\top e^{\mathbf{P}t} \mathbf{y} > 0 \iff (\frac{1}{S} \mathbf{x})^\top e^{\mathbf{P}t} \mathbf{y} > 0 \iff \mathbf{z}^\top e^{\mathbf{P}t} \mathbf{y} > 0$ .

Note that we reduced the Positivity Problem for matrix exponentials to the one above, where  $\mathbf{z}$  is a stochastic vector. Also,  $\mathbf{y}$ 's entries are either  $-1$ ,  $0$ , or  $1$ .

Let the entries of  $\mathbf{P}$  be  $p_{ij}$ . Let us now pick a number  $r$  that is greater than the sum of any row of  $\mathbf{P}$ :  $r > \sum_{j=1}^{2k} p_{ij}$ , for each  $1 \leq i \leq 2k$ . Let  $q_i := r - \sum_{j=1}^{2k} p_{ij}$ , for each  $1 \leq i \leq 2k$ , and let  $\mathbf{Q}$  be a diagonal matrix:  $\mathbf{Q} := \text{diag}(q_1, \dots, q_{2k})$ .

Let us define  $\mathbf{O} := \begin{bmatrix} \mathbf{P} - r\mathbf{I} & \mathbf{Q} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ , where each of the four blocks of  $\mathbf{O}$  is a  $2k \times 2k$  matrix. We note that  $\mathbf{O}$  is a rate matrix. By inspection of the block multiplications, it is easy to see that the top left block of  $\mathbf{O}^n$  is  $(\mathbf{P} - r\mathbf{I})^n$ . Hence, by the power series expansion of  $e^{\mathbf{X}}$  and by setting  $\mathbf{u}_1 := \begin{bmatrix} \mathbf{z} \\ \mathbf{0} \end{bmatrix}$ ,  $\mathbf{v}_1 := \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}$ , we get:  $\mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_1 = \mathbf{z}^\top e^{(\mathbf{P}-r\mathbf{I})t} \mathbf{y} = \frac{1}{e^{rt}} \mathbf{z}^\top e^{\mathbf{P}t} \mathbf{y}$ , so

$$\mathbf{z}^\top e^{\mathbf{P}t} \mathbf{y} > 0 \iff \mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_1 > 0.$$

Thus, we reduced the initial problem to the existence of a  $t$  in  $[a, b]$  such that  $\mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_1 > 0$ , where  $\mathbf{u}_1$  is stochastic,  $\mathbf{O}$  is a rate matrix and  $\mathbf{v}_1$  has entries in  $\{-1, 0, 1\}$ .

By letting  $\mathbf{v}_2 := \mathbf{v}_1 + \mathbf{1}$ :  $\mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_1 > 0 \iff \mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_2 > \mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{1} = 1$ . Furthermore,  $\mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_2 > 1 \iff \mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_3 > \frac{1}{2}$ , where  $\mathbf{v}_3 := \frac{1}{2} \mathbf{v}_2$ , so  $\mathbf{v}_3$ 's entries are in  $\{0, \frac{1}{2}, 1\}$ .

We have reduced the problem whether there exists some  $t$  in  $[a, b]$  such that  $\mathbf{u}^\top e^{\mathbf{A}t} \mathbf{v} > 0$ , where  $\mathbf{u}, \mathbf{v}$  are any vectors and  $\mathbf{A}$  is any matrix, to the problem whether there exists some  $t$  in  $[a, b]$  such that  $\mathbf{u}_1^\top e^{\mathbf{O}t} \mathbf{v}_3 > \frac{1}{2}$ , where  $\mathbf{u}_1$  is stochastic,  $\mathbf{O}$  is a rate matrix and  $\mathbf{v}_3$  has entries in  $\{0, \frac{1}{2}, 1\}$ .

## 7:18 Parametric Continuous Stochastic Logic

This last problem asks for a given continuous-time Markov chain whether there exists a moment  $t$  in  $[a, b]$  such that summing the probabilities of being in certain states at time  $t$  with fixed weights in  $\{0, \frac{1}{2}, 1\}$  yields a result greater than  $\frac{1}{2}$ . We reduce this problem to a similar one with coefficients in  $\{0, 1\}$  by splitting the states in the former problem that have weight  $\frac{1}{2}$  in two identical states that, seen together as a black box, act as the original state.

More formally, if a state  $s_i$  has associated coefficient  $\frac{1}{2}$  in  $\mathbf{v}_3$ , we split it into states  $s_{i,1}$  and  $s_{i,2}$  and modify the transition rates:

- for any state  $s_j$  having a strictly positive transition rate  $r_{j,i}$  to  $s_i$ : delete this transition and add two new transition rates to  $s_{i,1}$  and  $s_{i,2}$ , both with rate  $\frac{r_{j,i}}{2}$ ,
- for any state  $s_j$  such that there is a strictly positive transition rate  $r_{i,j}$  from  $s_i$  to  $s_j$ : delete this transition and add two new transition rates from  $s_{i,1}$  and  $s_{i,2}$  to  $s_j$ , both with rate  $r_{j,i}$ .

Note that by being in state  $s_{i,1}$  or  $s_{i,2}$  in the new Markov chain we get the same behaviour as being in  $s_i$  in the original Markov chain (all the outgoing outgoing rates from  $s_{i,1}$  or  $s_{i,2}$  stay the same as the outgoing rates from  $s_i$ ). We also modify the initial distribution: if the initial probability of  $s_i$  was  $p_i$ , set the new initial probability in  $s_i$  to 0 and both initial probabilities of  $s_{i,1}$  and  $s_{i,2}$  to  $\frac{p_i}{2}$ .

By regarding the cluster of states  $s_{i,1}$  and  $s_{i,2}$  as a “black-box state”, it behaves equivalently to state  $s_i$  in the original Markov chain. Because of the symmetry, the probability of being in  $s_{i,1}$  at time  $t$  equals the probability of being in state  $s_{i,2}$  at time  $t$ , which is equal to half of the probability of being in state  $s_i$  at time  $t$  in the original Markov chain.

Starting from the CTMC with rate transition matrix  $\mathbf{O}$ , initial distribution  $\mathbf{u}_1$  and coefficient vector  $\mathbf{v}_3$ , we can iteratively apply the described splitting process, by going through all the states having weights in the original formulation equal to  $\frac{1}{2}$ . We then get a sequence of new continuous-time Markov chains  $\mathcal{M}_1, \dots, \mathcal{M}_N$  with rate matrices  $\mathbf{R}_1, \dots, \mathbf{R}_N$  and with initial distributions  $\tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_N$ , and new weight vectors  $\tilde{\mathbf{v}}_1, \dots, \tilde{\mathbf{v}}_N$  defined as follows:

- 0/1 in the corresponding positions in  $\tilde{\mathbf{v}}_{i+1}$  of states having previous coefficients 0/1 in  $\tilde{\mathbf{v}}_i$ ,
- 0 in the corresponding positions in  $\tilde{\mathbf{v}}_{i+1}$  of the most recent split state having previous coefficient 1/2 in  $\tilde{\mathbf{v}}_i$ ,
- 1 in the corresponding position in  $\tilde{\mathbf{v}}_{i+1}$  of the first newly created state by splitting (of the form  $s_{i,1}$ ) and 0 to the second such state (of the form  $s_{i,2}$ ),
- $\frac{1}{2}$  in the corresponding position in  $\tilde{\mathbf{v}}_{i+1}$  of all other states having previous coefficients  $\frac{1}{2}$  in  $\tilde{\mathbf{v}}_i$ .

We have the invariant  $\tilde{\mathbf{u}}_i^\top e^{\mathbf{R}_i t} \tilde{\mathbf{v}}_i = \tilde{\mathbf{u}}_{i+1}^\top e^{\mathbf{R}_{i+1} t} \tilde{\mathbf{v}}_{i+1}$ . Let  $\mathcal{M} := \mathcal{M}_N$  be the CTMC obtained after the iterative splitting process described above, with rate matrix  $\mathbf{R} := \mathbf{R}_N$  and initial distribution  $\tilde{\mathbf{u}} := \tilde{\mathbf{u}}_N$ , and let the final coefficient vector be  $\tilde{\mathbf{v}} := \tilde{\mathbf{v}}_N$ . It is clear now that  $\mathbf{u}_1^\top e^{\mathbf{O} t} \mathbf{v}_3 = \tilde{\mathbf{u}}^\top e^{\mathbf{R} t} \tilde{\mathbf{v}}$ .

Consequently,  $\exists t \in [a, b]$  s.t.  $\mathbf{u}^\top e^{\mathbf{A} t} \mathbf{v} > 0 \iff \exists t \in [a, b]$  s.t.  $\tilde{\mathbf{u}}^\top e^{\mathbf{R} t} \tilde{\mathbf{v}} > \frac{1}{2}$ , where  $\tilde{\mathbf{u}}$  is a stochastic vector,  $\mathbf{R}$  is a rate matrix and  $\tilde{\mathbf{u}}$  has entries in  $\{0, 1\}$ . We conclude that the Positivity Problem for matrix exponentials is reducible to the Threshold Problem for continuous-time Markov chains. ◀

# Universal Solutions in Temporal Data Exchange

Zehui Cheng 

University of California Santa Cruz, CA, USA  
zecheng@ucsc.edu

Phokion G. Kolaitis

University of California Santa Cruz, CA, USA  
IBM Research, Almaden, CA, USA  
kolaitis@ucsc.edu

---

## Abstract

---

During the past fifteen years, data exchange has been explored in depth and in a variety of different settings. Even though temporal databases constitute a mature area of research studied over several decades, the investigation of temporal data exchange was initiated only very recently. We analyze the properties of universal solutions in temporal data exchange with emphasis on the relationship between universal solutions in the context of concrete time and universal solutions in the context of abstract time. We show that challenges arise even in the setting in which the data exchange specifications involve a single temporal variable. After this, we identify settings, including data exchange settings that involve multiple temporal variables, in which these challenges can be overcome.

**2012 ACM Subject Classification** Information systems → Data management systems; Theory of computation → Data exchange; Information systems → Temporal data

**Keywords and phrases** temporal databases, database dependencies, data exchange, universal solutions, abstract time, concrete time, Allen's relations

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.8

**Funding** The research of Phokion Kolaitis is partially supported by NSF Grant IIS-1814152.

**Acknowledgements** We thank Jing Ao and Rada Chirkova for numerous fruitful conversations concerning temporal data exchange.

## 1 Introduction and Summary of Results

Data exchange is concerned with the transformation of data structured under one schema, called the *source* schema, into data structured under a different schema, called the *target* schema. Since the original formalization of the data exchange problem between relational schemas in [9] about fifteen years ago, an extensive study of data exchange has been carried out in several different settings, including XML data exchange [4], data exchange between graph databases [6], and relational to RDF data exchange [7]; an overview of the main results in this area can be found in the monograph [3]. Temporal databases constitute a mature area of research that has been studied in depth over several decades; for overviews, see, e.g., the book [13] or the book chapter [8]. Data exchange and temporal databases have advanced independently and, rather surprisingly, their paths did not cross until very recently, when Golshanara and Chomicki [11] published the first paper on *temporal data exchange*, that is, data exchange between temporal databases.

Data exchange is formalized using *schema mappings*, i.e., tuples of the form  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ , where  $\mathbf{S}$  is the source schema,  $\mathbf{T}$  is the target schema, and  $\Sigma$  is a finite set of constraints in some suitable logical formalism that describe the relationship between source and target. Every fixed schema mapping  $\mathcal{M}$  gives rise to the *data exchange problem with respect to*  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ : given a source instance  $I$ , find a *solution* for  $I$ , that is, a target instance  $J$  so that  $(I, J) \models \Sigma$ . In general, no solution for  $I$  may exist or multiple solutions for  $I$  may exist.



© Zehui Cheng and Phokion G. Kolaitis;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 8; pp. 8:1–8:17

Leibniz International Proceedings in Informatics



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

In [9], the concept of a *universal* solution was introduced and a case was made that universal solutions are the “best” solutions to materialize, provided solutions exist. In a precise sense (formalized using homomorphisms), a universal solution is a most general solution, thus it embodies no more and no less information than what the constraints in  $\Sigma$  specify. By now, universal solutions have been widely adopted as the preferred semantics in data exchange; furthermore, a concerted research effort has been dedicated to discovering when universal solutions exist and how to compute them. The main tool for computing universal solutions is the *chase* algorithm [9] and its variants (see [12] for a survey).

In temporal databases, there are two different models of time, namely, *concrete* time and *abstract* time; in the first model, time is represented by time intervals, while in the second by time points [8, 15]. Concrete temporal databases can be converted to abstract temporal databases using the *semantic function*<sup>1</sup>  $\llbracket \cdot \rrbracket$ , which takes as input a concrete temporal database  $D$  and returns as output the abstract temporal database  $\llbracket D \rrbracket$  where intervals of time in  $D$  are replaced by all points of time in them. The semantic function is often deployed to transfer results about concrete temporal databases to results about abstract temporal databases.

As already mentioned, Golshanara and Chomicki [11] are the first to investigate temporal data exchange. Specifically, they considered *temporal* schema mappings  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ , where  $\Sigma_{st}$  is a set of temporal source-to-target tuple-generating dependencies (temporal s-t tgds) and  $\Sigma_t$  is a set of temporal target equality-generating dependencies (temporal target egds) with the restriction that each such constraint contains exactly one temporal variable. This means that each constraint in  $\Sigma_{st}$  is of the form  $\forall \mathbf{x} \forall t (\varphi(\mathbf{x}, t) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}, t))$ , where  $t$  is the only temporal variable,  $\varphi(\mathbf{x}, t)$  is a conjunction of source atoms, and  $\psi(\mathbf{x}, \mathbf{y}, t)$  is a conjunction of target atoms. Also, each constraint in  $\Sigma_t$  is of the form  $\forall \mathbf{x} \forall t (\theta(\mathbf{x}, t) \rightarrow x_k = x_l)$ , where  $t$  is the only temporal variable and  $\theta(\mathbf{x}, t)$  is a conjunction of target atoms.

Let  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  be a temporal schema mapping as above. The main result in [11] is the discovery of a variant of the chase algorithm that has the following properties: (a) it runs in polynomial time; (b) given a concrete source instance  $I$ , it detects if  $I$  has a solution with respect to  $\mathcal{M}$ ; and (c) if  $I$  has such a solution, then it produces a concrete target instance  $J$  such that  $J$  is *semantically adequate* for  $I$ , i.e., the abstract target instance  $\llbracket J \rrbracket$  is a universal solution for the abstract source instance  $\llbracket I \rrbracket$ . In the sequel, we call *normalizing chase* the variant of the chase used in [11]. It is a natural extension of the chase algorithm to temporal dependencies, but with the twist that first a *normalization* step is performed on the given concrete source instance  $I$  and then the temporal s-t tgds are applied to the resulting normalized instance  $\mathcal{N}(I)$ ; after this, a second normalization step is performed on the resulting concrete target instance and then the temporal target egds are applied.

**Summary of Results.** Our investigation began when we noticed that Golshanara and Chomicki [11] do not address the question of whether or not the normalizing chase always produces a universal solution for a given concrete source instance, provided a solution exists (in fact, the notion of a universal solution for a concrete source instance is never introduced in [11]). We first show that the normalizing chase need *not* produce a universal solution for a given concrete source instance. Actually, we establish a stronger negative result: there is a temporal schema mapping  $\mathcal{M}^* = (\mathbf{S}, \mathbf{T}, \Sigma_{st}^*, \Sigma_t^*)$  as above and a concrete source instance  $I^*$  that has a solution with respect to  $\mathcal{M}^*$ , but there is *no* concrete universal solution  $J$  for  $I^*$  or for the normalized instance  $\mathcal{N}(I^*)$  that is semantically adequate for  $I^*$  (in particular, the result of the normalizing chase on  $I^*$  cannot be a universal solution for  $I^*$ ).

<sup>1</sup> In the temporal databases literature,  $\llbracket \cdot \rrbracket$  is called the *semantic mapping*. Here, we chose to call it the *semantic function* to avoid confusion with the term *schema mapping*, which will be used repeatedly throughout this paper.

The preceding state of affairs motivates the following question: which temporal schema mappings admit semantically adequate concrete universal solutions? We make progress towards answering this question by identifying sufficient conditions that guarantee the existence of semantically adequate concrete universal solutions. To this effect, we show that if the temporal target egds have at most one temporal atom in their left-hand side (and any number of non-temporal atoms), then the output of the normalizing chase on a given concrete instance  $I$  is a concrete universal solution for  $\mathcal{N}(I)$  and is also semantically adequate for  $I$ . In a sense, this is an optimal result because the temporal schema mapping  $\mathcal{M}^*$  above contains a temporal target egd with two temporal atoms in its left-hand side, hence this result cannot be extended to the class of schema mappings studied by Golshanara and Chomicki [11].

All aforementioned results concern temporal schema mappings in which each constraint contains at most one temporal variable. Here, we embark on an investigation of temporal data exchange using schema mappings specified by constraints that may contain several different temporal variables. Such constraints may also contain comparisons between temporal variables using the well known Allen's relations, thus they can capture richer data exchange scenarios. This expansion of the landscape, however, comes with a number of complications, since, among other things, constraints in the concrete model of time need to be carefully translated into constraints in the abstract model of time (constraints with at most one temporal variable do not change, only the interpretation of the temporal variables does).

In the setting of multiple temporal variables, we consider temporal *full* schema mappings  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ , i.e., schema mappings in which no existential quantifiers occur in the consequent of constraints in  $\Sigma_{st}$ . We show that if each temporal target egd has at most one temporal atom in its left-hand side, then we can produce concrete target instances that are both universal solutions and semantically adequate, provided solutions exist. Finally, we introduce another variant of the chase, which we call the *coalescing* chase, and show that for arbitrary temporal full schema mappings, the coalescing chase on concrete source instances always produces semantically adequate solutions, provided solutions exist.

## 2 Preliminaries

This section contains the definitions of the basic concepts and some background material.

**Models of Time.** Let  $\mathbb{N} = \{1, 2, \dots\}$  be the set of all natural numbers. In the *abstract* model of time, natural numbers represent *time points*. In the *concrete* model of time, closed-open intervals  $[s, e) = \{t \in \mathbb{N} : s \leq t < e\}$ , where  $s$  and  $e$  are natural numbers with  $s < e$ , represent *time intervals*. Unbounded time intervals of the form  $[s, \infty)$  are also allowed.

**Temporal Databases.** A relational schema is a finite collection  $\mathbf{R}$  of relation symbols of the form  $R(A_1, \dots, A_k)$ , where  $A_1, \dots, A_k$  are the *attributes* of  $R$  and  $k$  is its arity. An  $\mathbf{R}$ -instance  $I$  is a finite collection of finite relations  $R^I$ , one for each relation symbol  $R$  in  $\mathbf{R}$  and such that the arity of  $R^I$  matches that of  $R$ .

A *temporal* relation symbol is a relation symbol  $R$  in which one or more of its attributes are designated as temporal attributes, i.e., they can only take temporal values. In this paper, we assume that every temporal relation symbol has exactly one temporal attribute, which, without loss of generality, is the last attribute in the list. A *temporal* relational schema is a relational schema  $\mathbf{R}$  containing at least one temporal relation symbol. For such a schema  $\mathbf{R}$ , an *abstract R-instance* is an  $\mathbf{R}$ -instance in which the values of the temporal attributes are time points. A *concrete R-instance* is an  $\mathbf{R}$ -instance in which the values of the temporal attributes are time intervals. We will use the term *temporal database* to refer to both abstract instances and concrete instances.



**Constraints and Schema Mappings.** Let  $\mathbf{S}$  and  $\mathbf{T}$  be two relational schemas, called, respectively, the *source* schema and the *target* schema, where  $\mathbf{S}$  and  $\mathbf{T}$  have no relation symbols in common. Data exchange from  $\mathbf{S}$  to  $\mathbf{T}$  is formalized using constraints in some logical formalism that describe the relationship between these two schemas [9]. The most widely used such constraints are *source-to-target tuple-generating dependencies* (s-t tgds) and *target equality-generating dependencies* (target egds). A s-t tgd is a first-order sentence of the form  $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$ , where  $\varphi(\mathbf{x})$  is a conjunction of source atoms, and  $\psi(\mathbf{x}, \mathbf{y})$  is a conjunction of target atoms. Such constraints can express a variety of data transformation tasks, including copying a relation, projecting a relation, augmenting a relation with an extra column, and joining two or more relations, where, in each case, the result of the transformation is moved to the target [14]. A target egd is a first-order sentence of the form  $\forall \mathbf{x}(\theta(\mathbf{x}) \rightarrow x_k = x_l)$ , where  $\theta(\mathbf{x})$  is a conjunction of target atoms and  $x_k, x_l$  are variables occurring in  $\mathbf{x}$ . Target egds include target key constraints as an important special case.

The first step in formalizing data exchange between temporal relational schemas is to extend the concepts of s-t tgds and target egds to incorporate time. As stated in Section 1, Golshanara and Chomicki [11] initiated the study of temporal data exchange by considering temporal s-t tgds of the form  $\forall \mathbf{x}\forall t(\varphi(\mathbf{x}, t) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}, t))$  and temporal target egds of the form  $\forall \mathbf{x}\forall t(\theta(\mathbf{x}, t) \rightarrow x_k = x_l)$ , where  $t$  is the only temporal variable that occurs in these formulas (in particular, the consequent of temporal s-t tgds contains no existentially quantified temporal variables).

In Section 4, we will explore a much richer framework for temporal data exchange in which the constraints considered may contain multiple temporal variables. We introduce the basic notions for this richer framework in this section (of course, these notions apply to the framework studied by Golshanara and Chomicki [11] as well). Specifically, we consider temporal s-t tgds of the form  $\forall \mathbf{x}\forall \mathbf{t}(\varphi(\mathbf{x}, \mathbf{t}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}, \mathbf{t}))$  and temporal target egds of the form  $\forall \mathbf{x}\forall \mathbf{t}(\theta(\mathbf{x}, \mathbf{t}) \rightarrow x_k = x_l)$ , where  $\mathbf{t}$  is a (possibly empty) tuple of temporal variable; all other variables are non-temporal, thus the consequent of such temporal s-t tgds contains no existentially quantified temporal variables. We regard s-t tgds and target egds as the special cases of their temporal counterparts in which no temporal variable occurs (i.e., the tuple  $\mathbf{t}$  is empty). In what follows, we will use the term *temporal schema mapping* for a tuple  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ , where  $\mathbf{S}$  and  $\mathbf{T}$  are disjoint temporal relational schemas,  $\Sigma_{st}$  is a finite set of temporal s-t tgds, and  $\Sigma_t$  is a finite set of temporal target egds, as above.

**Values in Source and Target Instances.** In data exchange between relational schemas, the source instances contain values from a countable domain  $\text{CONST}$  of objects, called *constants*, while the target instances may contain values from the union  $\text{CONST} \cup \text{NULL}$ , where  $\text{NULL}$  is a countable set of distinct *labelled nulls*  $N_1, N_2, \dots$ , which are typically used to witness the existentially quantified variables in the right-hand sides of s-t tgds. Thus, a labelled null represents some unknown value. In temporal data exchange, the values occurring in source and target instances may also be time points or time intervals, depending on the model of time used. Furthermore, the use of null values in target instances requires delicate handling because such null values may need to take into account the temporal context in which they are introduced. For this reason, temporal target instances may contain values that are constants, time points in the abstract model of time (or time intervals in the concrete model of time), labelled nulls  $N_1, N_2, \dots$ , and *time-stamped* nulls, that is, null values of the form  $N_1^{\mathbf{t}}, N_2^{\mathbf{t}}, \dots$ , where  $\mathbf{t}$  is a finite sequence of time points (or a finite sequence of time intervals). Two such time-stamped nulls are equal if and only if they have the same subscript (label) and the same time-stamp. Intuitively, a time-stamped null represents unknown values in the context of its time-stamp. For example, a time-stamped null  $N_j^{[2,5]}$  represents three unknown values, one at time-point 2, one at time-point 3, and one at time-point 4.



**Homomorphisms, Solutions, and Universal Solutions.** Let  $\mathbf{T}$  be a temporal target schema and let  $J$  and  $J'$  be two temporal target databases over the same model of time (i.e., both are abstract or both are concrete). As discussed above, the relations in  $J$  and  $J'$  may contain constants, labelled nulls, and time-stamped nulls as values.

A *homomorphism* from  $J$  to  $J'$  is a function  $h$  from the active domain<sup>2</sup> of  $J$  to the active domain of  $J'$  such that: (a) if  $v$  is a constant or a time value (time point or time interval), then  $h(v) = v$ ; (b) if  $v$  is a labelled null  $N_j$ , then  $h(v)$  is either a constant or a labelled null  $N_k$ ; (c) if  $v$  is a time-stamped null  $N_j^t$ , then  $h(N_j^t)$  is a constant or a null  $N_k^t$  with the same time-stamp or a labelled null  $N_k$  (without a time-stamp); (d) if a tuple  $(v_1, \dots, v_m)$  belongs to a relation  $R^J$  of  $J$ , then  $(h(v_1), \dots, h(v_m))$  belongs to the relation  $R^{J'}$  of  $J'$ .

The intuition behind this definition is that if there is a homomorphism from  $J$  to  $J'$ , then  $J$  is “more general” than  $J'$ . Time-stamped nulls are “more general” than labelled nulls, since the latter represent a single unknown value, while the former may represent multiple unknown values, depending on the time-stamp used. This explains the different treatment of labelled nulls and time-stamped nulls in conditions (b) and (c), respectively, in the definition.

Let  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  be a temporal schema mapping and  $I$  a concrete source instance. A concrete target instance  $J$  is a *solution for  $I$  w.r.t.  $\mathcal{M}$*  if the following conditions hold:

- If  $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$  is a (non-temporal) s-t tgd in  $\Sigma_{st}$  and if  $\mathbf{a}$  is a tuple from the active domain of  $I$  such that  $I \models \varphi(\mathbf{a})$ , then there is a tuple  $\mathbf{b}$  that consists of constants and/or labelled nulls such that  $J \models \psi(\mathbf{a}, \mathbf{b})$ .
- If  $\forall \mathbf{x}\forall \mathbf{t}(\varphi(\mathbf{x}, \mathbf{t}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}, \mathbf{t}))$  is a temporal s-t tgd in  $\Sigma_{st}$  and if  $\mathbf{a}$  is a tuple of constants and  $\mathbf{i}$  is a tuple of intervals such that  $I \models \varphi(\mathbf{a}, \mathbf{i})$ , then there is a tuple  $\mathbf{b}$  that consists of constants, labelled nulls, and time-stamped nulls such that every time-stamped null in  $\mathbf{b}$  has  $\mathbf{i}$  as its time-stamp and  $J \models \psi(\mathbf{a}, \mathbf{b}, \mathbf{i})$ .
- If  $\forall \mathbf{x}\forall \mathbf{t}(\theta(\mathbf{x}, \mathbf{t}) \rightarrow x_k = x_l)$  is a temporal target egd in  $\Sigma_t$  and if  $\mathbf{a}$  and  $\mathbf{i}$  are tuples such that  $J \models \theta(\mathbf{a}, \mathbf{i})$ , then  $a_k = a_l$ , which means that  $a_k$  and  $a_l$  are the same constant or the same labelled null  $N_j$  or the same time-stamped null  $N_j^i$ .

A concrete target instance  $J$  is a *universal solution for  $I$  w.r.t.  $\mathcal{M}$*  if  $J$  is a solution for  $I$  w.r.t.  $\mathcal{M}$  and, for every solution  $J'$  for  $I$  w.r.t.  $\mathcal{M}$ , there a homomorphism from  $J$  to  $J'$ .

**The Chase and its Variants.** In the case of (standard) data exchange, universal solutions are produced using the chase procedure [9]. Intuitively, given a source instance  $I$ , the chase procedure attempts to produce a target instance  $J$  by starting with the empty target instance, repeatedly applying the constraints of the given schema mapping, and generating new tuples in the current target instance as needed, so that eventually either the current target instance satisfies all the constraints of the schema mapping  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  or a conflict arises in which case there is no solution for  $I$  w.r.t.  $\mathcal{M}$ . We now describe at a high level how the chase algorithm can be adapted to the setting of temporal data exchange.

Let  $K$  be the current concrete target instance in the run of the chase.

- If  $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$  is a (non-temporal) s-t tgd in  $\Sigma_{st}$  and if  $\mathbf{a}$  is a tuple from the active domain of  $I$  such that  $I \models \varphi(\mathbf{a})$ , but  $K \not\models \exists \mathbf{y}\psi(\mathbf{a}, \mathbf{y})$ , then the chase generates a tuple  $\mathbf{b}$  of distinct labelled nulls for the variables in  $\mathbf{y}$  and adds tuples to the relations in  $K$  so that the resulting instance  $K'$  satisfies  $\psi(\mathbf{a}, \mathbf{b})$ . (Same as in standard chase.)
- If  $\forall \mathbf{x}\forall \mathbf{t}(\varphi(\mathbf{x}, \mathbf{t}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}, \mathbf{t}))$  is a temporal s-t tgd in  $\Sigma_{st}$  and if  $\mathbf{a}$  and  $\mathbf{i}$  are such that  $I \models \varphi(\mathbf{a}, \mathbf{i})$ , but  $K \not\models \exists \mathbf{y}\psi(\mathbf{a}, \mathbf{y}, \mathbf{i})$ , then the chase generates a tuple  $\mathbf{b}$  of distinct time-stamped labelled nulls for the variables in  $\mathbf{y}$  all of which have the same time-stamp  $\mathbf{i}$  and adds tuples to the relations in  $K$  so that the resulting instance  $K'$  satisfies  $\psi(\mathbf{a}, \mathbf{b}, \mathbf{i})$ .

<sup>2</sup> The *active domain* of a database is the set of all values occurring in the relations of that database.

- After the concrete source instance  $I$  has been chased with the constraints in  $\Sigma_{st}$ , then the concrete target instance  $K$  produced thus far is chased with the constraints in  $\Sigma_t$ . Specifically, if  $\forall \mathbf{x} \forall \mathbf{t} (\theta(\mathbf{x}, \mathbf{t}) \rightarrow x_k = x_l)$  is a temporal target egd in  $\Sigma_t$  and  $\mathbf{a}$  and  $\mathbf{i}$  are tuples such that  $K \models \theta(\mathbf{a}, \mathbf{i})$ , then the following cases are considered: (1) if both  $a_k$  and  $a_l$  are labelled nulls or both are time-stamped nulls with the same time-stamp, then one of the two is replaced by the other throughout  $K$ ; (2) if one of  $a_k$  and  $a_l$  is a constant and the other is a labelled null or a time-stamped null, then the labelled null or the time-stamped null is replaced by the constant throughout  $K$ ; (3) if one of  $a_k, a_l$  is a labelled null and the other is a time-stamped null, then the time-stamped null is replaced by the labelled null throughout  $K$ ; (4) if  $a_k$  and  $a_l$  are time-stamped nulls with different time-stamps or if  $a_k$  and  $a_l$  are different constants, then the chase fails.

In what follows, we will use the term the *concrete chase algorithm* to refer to the algorithm just described. In their study of temporal data exchange, Golshanara and Chomicki [11] considered a variant of the chase algorithm, which here we will call the *concrete n-chase algorithm*. There are two main differences between these two algorithms:

- In [11], all temporal schema mappings have s-t tgds with exactly one temporal variable, which implies that (standard) s-t tgds are not allowed. As a result, the target instances produced by the concrete n-chase algorithm contain no labelled nulls, but, of course, they may contain time-stamped nulls in which the time-stamp is a single interval.
- The concrete n-chase algorithm performs a *normalization* step before the constraints in  $\Sigma_{st}$  are applied and another normalization step before the constraints in  $\Sigma_t$  are applied. In particular, the concrete n-chase algorithm does not chase the given concrete source instance  $I$  with  $\Sigma_{st}$ , but, instead, chases the normalized instance  $\mathcal{N}(I)$  with  $\Sigma_{st}$ . We refer the reader to Section 4.2 in [11] for the definition of normalization.

In what follows, if  $\mathcal{M}$  is a temporal schema mapping and  $I$  is a concrete source instance, we will write  $c\text{-chase}_{\mathcal{M}}(I)$  and  $n\text{-chase}_{\mathcal{M}}(I)$  to denote the concrete target instance produced by the concrete chase algorithm and, respectively, the concrete n-chase algorithm on  $I$ .

**Semantic Functions and Semantic Adequacy.** As mentioned in Section 1, concrete instances are converted to abstract instances using the semantic function  $\llbracket \cdot \rrbracket$ .

- If  $\mu = (c_1, \dots, c_m, [s, e])$  is a tuple in which each  $c_k$  is a constant and  $[s, e]$  is an interval, then  $\llbracket \mu \rrbracket = \{(c_1, \dots, c_m, t) : s \leq t < e\}$ .
- If  $I = (R_1, \dots, R_n)$  is a concrete source instance, then  $\llbracket I \rrbracket$  is the abstract source instance  $\llbracket I \rrbracket = (\llbracket R_1 \rrbracket, \dots, \llbracket R_n \rrbracket)$ , where  $\llbracket R_l \rrbracket = \bigcup_{\mu \in R_l} \llbracket \mu \rrbracket$ , for  $1 \leq l \leq n$ .
- We say that a tuple  $\nu = (a_1, \dots, a_m, [s, e])$  is *compatible* if each  $a_k$  is a constant or a labelled null or a time-stamped null  $N_j^{[s_1, e_1], \dots, [s_p, e_p]}$  such that  $[s, e]$  is one of the intervals in the time-stamp, and all time-stamped nulls in  $\nu$  have the same time-stamp. If  $\nu$  is a compatible tuple, then  $\llbracket \nu \rrbracket$  is the set of all tuples  $(b_1, \dots, b_m, t)$  such that the following conditions hold: if  $a_l$  is a constant or a labelled null, then  $b_l = a_l$ ; if  $a_l$  is a time-stamped null  $N_j^{[s_1, e_1], \dots, [s_p, e_p]}$ , then  $b_j$  is a time-stamped null  $N_j^{t_1, \dots, t_p}$ , where  $s_1 \leq t_1 < e_1, \dots, s_p \leq t_p < e_p$ ; and, finally,  $s \leq t < e$ .
- Let  $J = (T_1, \dots, T_m)$  be the concrete target instance produced by the concrete chase algorithm or by the concrete n-chase algorithm on a source instance  $I$ . It is easy to verify that every tuple occurring in one of the relations of  $J$  is compatible. Then  $\llbracket J \rrbracket$  is the abstract target instance  $\llbracket J \rrbracket = (\llbracket T_1 \rrbracket, \dots, \llbracket T_m \rrbracket)$ , where  $\llbracket T_l \rrbracket = \bigcup_{\nu \in T_l} \llbracket \nu \rrbracket$ , for  $1 \leq l \leq m$ .
- Let  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  be a temporal schema mapping with exactly one temporal variable per constraint and let  $I$  be a concrete source instance. We say that a concrete target instance  $J$  is *semantically adequate* for  $I$  if the abstract target instance  $\llbracket J \rrbracket$  is a universal solution for  $\llbracket I \rrbracket$  w.r.t.  $\mathcal{M}$ .

We are now ready to state the main result in [11].

► **Theorem 1.** (Theorem 19 in [11]) *Let  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  be a temporal schema mapping, such that each relational symbol in  $\mathbf{S}$  and  $\mathbf{T}$  has one temporal attribute and each constraint in  $\Sigma_{st} \cup \Sigma_t$  has exactly one temporal variable. If  $I$  is a concrete source instance, then the following statements are true.*

- *If the concrete n-chase algorithm on  $I$  fails, then there is no solution for  $I$  w.r.t.  $\mathcal{M}$ .*
- *If the concrete n-chase algorithm on  $I$  does not fail, then the concrete target instance  $n\text{-chase}_{\mathcal{M}}(I)$  produced by the algorithm is semantically adequate for  $I$ .*

We note that the normalization steps in the concrete n-chase algorithm guarantee that there is a homomorphism from the left-hand side of a constraint in  $\Sigma_{st}$  or in  $\Sigma_t$  to a concrete instance  $K$ , provided there is a homomorphism from the left-hand side of that constraint to the abstract instance  $\llbracket K \rrbracket$ .

### 3 Temporal Data Exchange with a Single Temporal Variable

In this section, we explore aspects of data exchange for temporal schema mappings  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  in which each constraint in  $\Sigma_{st} \cup \Sigma_t$  contains at most one temporal variable. In what follows, we will also assume that all concrete source instances  $I$  are *coalesced*, that is, if  $c_1, \dots, c_m$  are constants and  $i, i'$  are intervals such that  $(c_1, \dots, c_m, i)$  and  $(c_1, \dots, c_m, i')$  belong to the same relation of  $I$ , then  $i$  and  $i'$  are disjoint intervals. Clearly, every concrete source instance can be easily transformed to an “equivalent” coalesced one [8].

#### 3.1 No Semantically Adequate Concrete Universal Solutions

We begin by focusing more narrowly on schema mappings in the setting of Golshanara and Chomicki [11], that is, temporal schema mappings  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  such that each relational symbol in  $\mathbf{S}$  and  $\mathbf{T}$  has one temporal attribute and each constraint in  $\Sigma_{st} \cup \Sigma_t$  has *exactly* one temporal variable (hence, this variable occurs in every atom of the consequent of every s-t tgd). This class of schema mappings does not contain standard (non-temporal) schema mappings as a special case. Several remarks are in order now.

1. Such a schema mapping  $\mathcal{M}$  is meaningful in both the concrete model of time and the abstract model of time without changing the constraints in  $\Sigma_{st} \cup \Sigma_t$ . In the first case, the temporal variable is ranging over time intervals and in the second over time points.
2. Every abstract source instance can be viewed as a sequence of *snapshots*, that is, as a sequence of non-temporal source instances parameterized by time points. One can then drop the temporal variable from the constraints in  $\Sigma_{st} \cup \Sigma_t$ , chase each snapshot with the resulting standard schema mapping, produce a universal solution for each snapshot (if a solution exists for each snapshot), and then consolidate the resulting target snapshots into an abstract target instance, which is an abstract universal solution for the given abstract source instance<sup>3</sup> - see [11] for formal details.
3. Let  $I$  be a concrete source instance. The concrete chase algorithm described in Section 2 produces a concrete universal solution for  $I$  w.r.t.  $\mathcal{M}$ , if a solution exists; if the concrete chase fails, no solution for  $I$  w.r.t.  $\mathcal{M}$  exists. This follows from Theorem 5 in Section 4. As mentioned in Section 1, Golshanara and Chomicki [11] do not address the question of whether or not their concrete n-chase algorithm produces a concrete universal solution. In fact, the notion of a concrete universal solution is not introduced in [11]. Our first result provides a strong negative answer to this question.

<sup>3</sup> If the chase fails on one of the snapshots, then no solution for the given abstract source instance exists.

► **Theorem 2.** *There is a temporal schema mapping  $\mathcal{M}^* = (\mathbf{S}, \mathbf{T}, \Sigma_{st}^*, \Sigma_t^*)$  with one temporal variable in each constraint in  $\Sigma_{st}^* \cup \Sigma_t^*$  and there is a concrete source instance  $I^*$  such that the following properties hold:*

1. *The concrete target instance  $n\text{-chase}_{\mathcal{M}^*}(I^*)$  returned by the concrete  $n$ -chase algorithm on  $I^*$  is neither a solution for  $I^*$  nor for the normalized instance  $\mathcal{N}(I^*)$  w.r.t.  $\mathcal{M}^*$ .*
2. *There is a concrete universal solution for  $I^*$  w.r.t.  $\mathcal{M}^*$ , but there is no concrete universal solution for  $I^*$  w.r.t.  $\mathcal{M}^*$  that is semantically adequate for  $I^*$ .*
3. *There is a concrete universal solution for  $\mathcal{N}(I^*)$  w.r.t.  $\mathcal{M}^*$ , but there is no concrete universal solution for  $\mathcal{N}(I^*)$  w.r.t.  $\mathcal{M}^*$  that is semantically adequate for  $\mathcal{N}(I^*)$ .*

**Proof.** Let  $\mathcal{M}^* = (\mathbf{S}, \mathbf{T}, \Sigma_{st}^*, \Sigma_t^*)$  be the schema mapping where  $\Sigma_{st}^*$  consists of the constraints

$$\begin{aligned} \forall n, s, c, t (E(n, c, t) \wedge S(n, s, t) \rightarrow \text{Emp}(n, c, s, t)) \\ \forall n, c, p, t (P(n, p, t) \rightarrow \exists c \text{EmpPos}(n, c, p, t)) \end{aligned}$$

and  $\Sigma_t^*$  consists of the constraint

$$\forall n, c_1, c_2, s, p, t (\text{Emp}(n, c_1, s, t) \wedge \text{EmpPos}(n, c_2, p, t) \rightarrow c_1 = c_2).$$

Let  $I^*$  be the concrete source instance whose relations are depicted in Table 1. After normalizing  $I^*$  w.r.t.  $\Sigma_{st}^*$  (see [11] for the precise definition of normalization), we obtain the normalized instance  $\mathcal{N}(I^*)$  whose relations are depicted in Table 2.

■ **Table 1** The relations  $E$ ,  $S$ , and  $P$  of the concrete source instance  $I^*$ .

(a)  $E$ .

Name	Company	Time
Ada	IBM	[2013, 2018]
Bob	IBM	[2012, 2015]

(b)  $S$ .

Name	Salary	Time
Ada	18000	[2014, 2018]
Bob	13000	[2013, 2015]

(c)  $P$ .

Name	Position	Time
Ada	Manager	[2015, 2017]
Bob	Consultant	[2012, 2015]

■ **Table 2** The relations  $E$ ,  $S$ , and  $P$  of the normalized instance  $\mathcal{N}(I^*)$ .

(a)  $E$ .

Name	Company	Time
Ada	IBM	[2013, 2014]
Ada	IBM	[2014, 2018]
Bob	IBM	[2012, 2013]
Bob	IBM	[2013, 2015]

(b)  $S$ .

Name	Salary	Time
Ada	18000	[2014, 2018]
Bob	13000	[2013, 2015]

(c)  $P$ .

Name	Position	Time
Ada	Manager	[2015, 2017]
Bob	Consultant	[2012, 2015]

Let  $n\text{-chase}_{\mathcal{M}^*}(I^*)$  be the concrete target instance produced by the concrete  $n$ -chase algorithm on  $I^*$ ; its relations are depicted in Table 3. It is easy to see that  $n\text{-chase}_{\mathcal{M}^*}(I^*)$  is neither a solution for  $I^*$  nor a solution for  $\mathcal{N}(I^*)$ . This proves the first part of the theorem.

■ **Table 3** The relations  $\text{Emp}$  and  $\text{EmpPos}$  of the concrete target instance  $n\text{-chase}_{\mathcal{M}^*}(I^*)$ .

(a)  $\text{Emp}$ .

Name	Company	Salary	Time
Ada	IBM	18000	[2014, 2015]
Ada	IBM	18000	[2015, 2017]
Ada	IBM	18000	[2017, 2018]
Bob	IBM	13000	[2013, 2015]

(b)  $\text{EmpPos}$ .

Name	Company	Position	Time
Ada	IBM	Manager	[2015, 2017]
Bob	$N_2^{[2012, 2013]}$	Consultant	[2012, 2013]
Bob	IBM	Consultant	[2013, 2015]

Let  $c\text{-chase}_{\mathcal{M}^*}(I^*)$  and  $c\text{-chase}_{\mathcal{M}^*}(\mathcal{N}(I^*))$  be the concrete target instances produced by the concrete chase algorithm on  $I^*$  and on  $\mathcal{N}(I^*)$ . The relations of  $c\text{-chase}_{\mathcal{M}^*}(I^*)$  are depicted in Table 4, and those of  $c\text{-chase}_{\mathcal{M}^*}(\mathcal{N}(I^*))$  in Table 5. Note that  $c\text{-chase}_{\mathcal{M}^*}(I^*)$  is a universal solution for  $I^*$ , while  $c\text{-chase}_{\mathcal{M}^*}(\mathcal{N}(I^*))$  is a universal solution for  $\mathcal{N}(I^*)$ .

■ **Table 4** The relations  $Emp$  and  $EmpPos$  of the concrete target instance  $c\text{-chase}_{\mathcal{M}^*}(I^*)$ .

(a)  $Emp$ .

Name	Company	Salary	Time
Ada	IBM	18000	[2014, 2018]
Bob	IBM	13000	[2013, 2015]

(b)  $EmpPos$ .

Name	Company	Position	Time
Ada	$N_1^{[2015, 2017]}$	Manager	[2015, 2017]
Bob	$N_2^{[2012, 2015]}$	Consultant	[2012, 2015]

■ **Table 5** The relations  $Emp$  and  $EmpPos$  of the concrete target instance  $c\text{-chase}_{\mathcal{M}^*}(\mathcal{N}(I^*))$ .

(a)  $Emp$ .

Name	Company	Salary	Time
Ada	IBM	18000	[2014, 2018]
Bob	IBM	13000	[2013, 2015]

(b)  $EmpPos$ .

Name	Company	Position	Time
Ada	$N_1^{[2015, 2017]}$	Manager	[2015, 2017]
Bob	$N_2^{[2012, 2015]}$	Consultant	[2012, 2015]

Let  $a\text{-chase}_{\mathcal{M}^*}(\llbracket I^* \rrbracket)$  be the abstract target instance produced by chasing the snapshots of  $\llbracket I^* \rrbracket$ ; its relations are depicted in Table 6.

■ **Table 6** The relations  $Emp$  and  $EmpPos$  of the abstract target instance  $a\text{-chase}_{\mathcal{M}^*}(\llbracket I^* \rrbracket)$ .

(a)  $Emp$ .

Name	Company	Salary	Time
Ada	IBM	18000	2014
Ada	IBM	18000	2015
Ada	IBM	18000	2016
Ada	IBM	18000	2017
Bob	IBM	13000	2013
Bob	IBM	13000	2014

(b)  $EmpPos$ .

Name	Company	Position	Time
Ada	IBM	Manager	2015
Ada	IBM	Manager	2016
Bob	$N_3^{2012}$	Consultant	2012
Bob	IBM	Consultant	2013
Bob	IBM	Consultant	2014

As shown in [11],  $a\text{-chase}_{\mathcal{M}^*}(\llbracket I^* \rrbracket)$  is a universal solution for  $\llbracket I^* \rrbracket$  w.r.t.  $\mathcal{M}^*$ . It is now easy to verify that  $\llbracket c\text{-chase}_{\mathcal{M}^*}(I^*) \rrbracket$  is *not* homomorphically equivalent to  $a\text{-chase}_{\mathcal{M}^*}(\llbracket I^* \rrbracket)$ . It follows that  $c\text{-chase}_{\mathcal{M}^*}(I^*)$  is *not* semantically adequate for  $I^*$ . Furthermore, it is not hard to show that if  $J$  and  $J'$  are universal solutions for  $I^*$  w.r.t.  $\mathcal{M}^*$ , then  $\llbracket J \rrbracket$  and  $\llbracket J' \rrbracket$  are homomorphically equivalent. Therefore, no concrete universal solution for  $I^*$  is semantically adequate for  $I^*$ . This proves the second part of the theorem. A similar argument with  $c\text{-chase}_{\mathcal{M}^*}(\mathcal{N}(I^*))$  in place of  $c\text{-chase}_{\mathcal{M}^*}(I^*)$  proves the third part of the theorem. ◀

### 3.2 Semantically Adequate Concrete Universal Solutions

Theorem 2 tells that in the temporal data exchange setting studied in [11], there are rather simple temporal schema mappings and temporal source instances for which no concrete universal solution is semantically adequate for these instances or for their normalized versions. A close scrutiny of the proof of Theorem 2 reveals that the root cause for this state of affairs appears to be the presence of two temporal atoms in the antecedent of the temporal target egd in  $\Sigma_t^*$ . Our next result tells that if the temporal target egds contain at most one temporal atom in the antecedent, then normalized instances have concrete universal solutions that are also semantically adequate concrete. Moreover, this result holds if each temporal constraint has *at most* one temporal variable, instead of *exactly* one temporal variable as in [11]; such constraints contain standard (non-temporal) s-t tgds and target egds as a special case.

► **Theorem 3.** *Let  $\mathcal{M} = (\mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t)$  be a temporal schema mapping such that (a) each s-t tgd contains at most one temporal variable; (b) if a s-t tgd contains a temporal variable, then that temporal variable occurs in every atom of its consequent; (c) each target egd contains at most one temporal atom in its antecedent. If  $I$  is a concrete source instance, then the following statements hold:*

## 8:10 Universal Solutions in Temporal Data Exchange

1. If a solution for  $I$  w.r.t.  $\mathcal{M}$  exists, then  $n\text{-chase}_{\mathcal{M}}(I) = c\text{-chase}_{\mathcal{M}}(\mathcal{N}(I))$ , that is, the concrete target instance returned by the concrete  $n$ -chase algorithm coincides with the concrete target instance returned by the concrete chase algorithm on  $\mathcal{N}(I)$ . Consequently,  $\mathcal{N}(I)$  has a semantically adequate concrete universal solution.
2. If the concrete chase algorithm fails on  $\mathcal{N}(I)$ , then there is no solution for  $\llbracket I \rrbracket$  w.r.t.  $\mathcal{M}$ .

**Proof.** (*Hint*) The key observation is that if every constraint in  $\Sigma_t$  contains at most one temporal atom in its antecedent, then the second normalization step in the concrete  $n$ -chase algorithm does not change the temporal target instance produced by chasing  $\mathcal{N}(I)$  with the constraints in  $\Sigma_{st}$ . It follows that  $n\text{-chase}_{\mathcal{M}}(I) = c\text{-chase}_{\mathcal{M}}(\mathcal{N}(I))$ . It can also be shown that  $n\text{-chase}_{\mathcal{M}}(I)$  is semantically adequate, even in this setting where each constraint in  $\Sigma_{st} \cup \Sigma_t$  contains at most one temporal variable (instead of exactly one such variable as in [11]). ◀

It should be pointed out that there are a schema mapping  $\mathcal{M}'$  that satisfies the hypothesis in Theorem 3 and a concrete source instance  $I'$  such that no semantically adequate concrete universal solution for  $I'$  w.r.t.  $\mathcal{M}'$  exists. This is shown in the next proposition.

► **Proposition 4.** *There is a temporal schema mapping  $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma'_{st}, \Sigma'_t)$  where each constraint in  $\Sigma'_{st} \cup \Sigma'_t$  contains at most one temporal variable and each constraint in  $\Sigma'_t$  contains at most one temporal atom in its antecedent, and there is a concrete source instance  $I'$ , such that there exists a concrete universal solution for  $I'$  w.r.t.  $\mathcal{M}'$ , but there is no concrete universal solution for  $I'$  w.r.t.  $\mathcal{M}'$  that is semantically adequate for  $I'$ .*

**Proof.** Let  $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma'_{st}, \Sigma'_t)$  be the schema mapping where  $\Sigma'_{st}$  consists of the constraints

$$\begin{aligned} \forall n, s, c, t (E(n, c, t) \wedge S(n, s, t) \rightarrow \text{Emp}(n, c, s, t)) \\ \forall n, c, p (P(n, p) \rightarrow \exists c \text{EmpPos}(n, c, p)) \end{aligned}$$

and  $\Sigma'_t$  consists of the constraint

$$\forall n, c_1, c_2, s, p, t (E(n, c_1, s, t) \wedge \text{EmpPos}(n, c_2, p) \rightarrow c_1 = c_2).$$

Let  $I'$  be the concrete source instance whose relations are depicted in Table 7. After applying the semantic function on  $I'$ , we obtain the abstract source instance  $\llbracket I' \rrbracket$  whose relations are depicted in Table 8.

■ **Table 7** The relations  $E$ ,  $S$ , and  $P$  of the concrete source instance  $I'$ .

(a)  $E$ .

Name	Company	Time
Ada	IBM	[2013, 2018)
Bob	IBM	[2012, 2015)

(b)  $S$ .

Name	Salary	Time
Ada	18000	[2014, 2018)
Bob	13000	[2013, 2015)

(c)  $P$ .

Name	Position
Ada	Manager
Bob	Consultant

■ **Table 8** The relations  $E$ ,  $S$ , and  $P$  of the abstract source instance  $\llbracket I' \rrbracket$ .

(a)  $E$ .

Name	Company	Time
Ada	IBM	2013
Ada	IBM	2014
Ada	IBM	2015
Ada	IBM	2016
Ada	IBM	2017
Bob	IBM	2012
Bob	IBM	2013
Bob	IBM	2014

(b)  $S$ .

Name	Salary	Time
Ada	18000	2014
Ada	18000	2015
Ada	18000	2016
Ada	18000	2017
Bob	13000	2013
Bob	13000	2014

(c)  $P$ .

Name	Position
Ada	Manager
Bob	Consultant

Let  $c\text{-chase}_{\mathcal{M}'}(I')$  be the concrete target instances produced by the concrete chase algorithm on  $I'$ . The relations of  $c\text{-chase}_{\mathcal{M}'}(I')$  are depicted in Table 9. Note that  $c\text{-chase}_{\mathcal{M}'}(I')$  is a universal solution for  $I'$ .

■ **Table 9** The relations  $Emp$  and  $EmpPos$  of the concrete target instance  $c\text{-chase}_{\mathcal{M}'}(I')$ .

(a)  $Emp$ .

Name	Company	Salary	Time
Ada	IBM	18000	2014
Ada	IBM	18000	2015
Ada	IBM	18000	2016
Ada	IBM	18000	2017
Bob	IBM	13000	2013
Bob	IBM	13000	2014

(b)  $EmpPos$ .

Name	Company	Position
Ada	$N_1$	Manager
Bob	$N_2$	Consultant

Let  $a\text{-chase}_{\mathcal{M}'}(\llbracket I' \rrbracket)$  be the abstract target instance produced by chasing the snapshots of  $\llbracket I' \rrbracket$ ; its relations are depicted in Table 10.

■ **Table 10** The relations  $Emp$  and  $EmpPos$  of the abstract target instance  $a\text{-chase}_{\mathcal{M}'}(\llbracket I' \rrbracket)$ .

(a)  $Emp$ .

Name	Company	Salary	Time
Ada	IBM	18000	2014
Ada	IBM	18000	2015
Ada	IBM	18000	2016
Ada	IBM	18000	2017
Bob	IBM	13000	2013
Bob	IBM	13000	2014

(b)  $EmpPos$ .

Name	Company	Position
Ada	IBM	Manager
Bob	IBM	Consultant

As shown in [11],  $a\text{-chase}_{\mathcal{M}'}(\llbracket I' \rrbracket)$  is a universal solution for  $\llbracket I' \rrbracket$  w.r.t.  $\mathcal{M}'$ . It is now easy to verify that  $\llbracket c\text{-chase}_{\mathcal{M}'}(I') \rrbracket$  is *not* homomorphically equivalent to  $a\text{-chase}_{\mathcal{M}'}(\llbracket I' \rrbracket)$ . From this, it follows that  $c\text{-chase}_{\mathcal{M}'}(I')$  is *not* semantically adequate for  $I'$ . Furthermore, it is not hard to show that if  $J$  and  $J'$  are universal solutions for  $I'$  w.r.t.  $\mathcal{M}'$ , then  $\llbracket J \rrbracket$  and  $\llbracket J' \rrbracket$  are homomorphically equivalent. Consequently, no concrete universal solution for  $I'$  is semantically adequate for  $I'$ . This completes the proof of the proposition. ◀

## 4 Temporal Data Exchange with Multiple Temporal Variables

In this section, we initiate the study of temporal data exchange for schema mappings whose constraints may contain multiple temporal variables. Such constraints make it possible to model more complex transformations of temporal data. In the presence of multiple temporal variables, it is natural to also allow comparisons between different temporal variables. In the concrete model of time, this means that the antecedents of the s-t tgds and the target egds may also contain Boolean combinations of the well known Allen's relations between time intervals [1, 2], such as  $m$  (meets),  $o$  (overlaps),  $<$  (before),  $>$  (after), and  $=$ . Thus, in this section, we consider temporal schema mappings  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  in which each constraint in  $\Sigma_{st}$  is of the form  $\forall \mathbf{x} \forall \mathbf{t} (\varphi(\mathbf{x}, \mathbf{t}) \wedge \pi(\mathbf{t}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}, \mathbf{t}))$ , where the only temporal variables are those in  $\mathbf{t}$ ;  $\varphi(\mathbf{x}, \mathbf{t})$  is a conjunction of source atoms;  $\pi(\mathbf{t})$  is a Boolean combination of Allen's relations involving variables from  $\mathbf{t}$ ; and  $\psi(\mathbf{x}, \mathbf{y}, \mathbf{t})$  is a conjunction of target atoms (in particular, no temporal variable is existentially quantified). By the same token, each constraint in  $\Sigma_t$  is of the form  $\forall \mathbf{x} \forall \mathbf{t} (\theta(\mathbf{x}, \mathbf{t}) \wedge \rho(\mathbf{t}) \rightarrow x_k = x_l)$ , where the only temporal variables are those in  $\mathbf{t}$ ;  $\theta(\mathbf{x}, \mathbf{t})$  is a conjunction of target atoms;  $\rho(\mathbf{t})$  is a Boolean combination of Allen's relations involving variables from  $\mathbf{t}$ ; and  $x_k, x_l$  are among the variables in  $\mathbf{x}$ .

The next result extends Theorem 3.3 in [9] from the case of (standard) data exchange to a restricted case of temporal data exchange.



► **Theorem 5.** Let  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  be a temporal schema mapping, such that one of the following two conditions holds: (a) Every s-t tgd is full (i.e., its consequent contains no existential quantifiers); (b) If a s-t tgd is not full and if it contains a temporal variable, then this is the only temporal variable in that s-t tgd, and it occurs in every atom of the consequent of the s-t tgd; moreover, every target egd contains at most one temporal variable. If  $I$  is a concrete source instance, then the following statements hold:

1. If the concrete chase algorithm does not fail on  $I$ , then the concrete target instance  $c\text{-chase}_{\mathcal{M}}(I)$  returned by this algorithm is a concrete universal solution for  $I$  w.r.t.  $\mathcal{M}$ .
2. If the concrete chase algorithm fails on  $I$ , there is no solution for  $I$  w.r.t.  $\mathcal{M}$ .

The running time of the concrete chase algorithm is bounded by a polynomial in the size of  $I$ .

Next, we explore the interplay between the concrete and the abstract models of time with focus on the existence of semantically adequate concrete universal solutions. In the presence of multiple temporal variables, concrete s-t tgds and concrete target egds must be converted to “essentially equivalent” abstract s-t tgds and to abstract target egds, respectively, because the concrete ones involve Allen’s relations while the abstract ones involve suitable formulas of first-order logic over time points compared with the  $<$  relation. Due to space limitations, we do not include here the precise definition of this conversion. Instead, we describe the precise sense in which this conversion transforms concrete constraints to “essentially equivalent” abstract constraints, and also illustrate this conversion in the proof of Proposition 7.

We will use the terms *concrete schema mapping* and *abstract schema mapping* for a schema mapping consisting of concrete constraints and, respectively, of abstract constraints. If  $\sigma$  is a concrete s-t tgd or a concrete target egds, then we write  $a(\sigma)$  for the abstract s-t tgd or the abstract target egd resulting from  $\sigma$  via the aforementioned conversion. Every concrete schema mapping  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  gives rise to an abstract schema mapping  $\mathcal{M}^a = (\mathbf{S}, \mathbf{T}, \Sigma_{st}^a, \Sigma_t^a)$ , where  $\Sigma_{st}^a = \{a(\sigma) : \sigma \in \Sigma_{st}\}$  and  $\Sigma_t^a = \{a(\sigma) : \sigma \in \Sigma_t\}$ .

Let  $\mathbf{x} = (x_1, \dots, x_m)$  be a tuple of non-temporal variables and let  $\mathbf{t} = (t_1, \dots, t_k)$  be a tuple of temporal variables. A *concrete* (respectively, an *abstract*) assignment to the tuple  $(\mathbf{x}, \mathbf{t})$  is a function  $p$  defined on the set  $\{x_1, \dots, x_m, t_1, \dots, t_k\}$  such that  $p(x_i) = c_i$  is a constant,  $1 \leq i \leq m$ , and  $p(t_j) = [s_j, e_j]$  is an interval (respectively,  $p(t_j) = \alpha_j$  is a time point),  $1 \leq j \leq k$ . If  $p$  is a concrete assignment as above, we will use the notation  $p(\mathbf{x}, \mathbf{t}) = (c_1, \dots, c_m, [s_1, e_1], \dots, [s_k, e_k])$  to denote it. Similarly, if  $p$  is an abstract assignment, it will be denoted as  $p(\mathbf{x}, \mathbf{t}) = (c_1, \dots, c_m, \alpha_1, \dots, \alpha_k)$ .

The semantic function  $\llbracket \cdot \rrbracket$  on concrete assignments is defined as follows: if  $p(\mathbf{x}, \mathbf{t}) = (c_1, \dots, c_m, [s_1, e_1], \dots, [s_k, e_k])$  is a concrete assignment, then  $\llbracket p(\mathbf{x}, \mathbf{t}) \rrbracket$  is the set of all abstract assignments  $q(\mathbf{x}, \mathbf{t}) = (c_1, \dots, c_m, \alpha_1, \dots, \alpha_k)$ , where  $s_j \leq \alpha_j < e_j$  and  $1 \leq j \leq k$ . The next proposition describes the properties of the conversion from concrete formulas to “essentially equivalent” abstract formulas.

► **Proposition 6.** Assume that  $\psi(\mathbf{x}, \mathbf{t})$  is a formula of the form  $\psi(\mathbf{x}, \mathbf{t}) = \varphi(\mathbf{x}, \mathbf{t}) \wedge \pi(\mathbf{t})$ , where the variables in  $\mathbf{t}$  are the only temporal variables,  $\varphi(\mathbf{x}, \mathbf{t})$  is a conjunction of atoms over a temporal schema  $\mathbf{S}$ , and  $\pi(\mathbf{t})$  is a Boolean combination of Allen’s relations involving variables from  $\mathbf{t}$ . Let  $\psi^a(\mathbf{x}, \mathbf{t})$  be the formula obtained by converting  $\psi(\mathbf{x}, \mathbf{t})$  from the concrete model of time to the abstract model of time. Given a coalesced concrete instance  $I$  and a concrete assignment  $p(\mathbf{x}, \mathbf{t})$  taking values in  $I$ , the following statements are equivalent:

- $I, p(\mathbf{x}, \mathbf{t}) \models \psi(\mathbf{x}, \mathbf{t})$ .
- For every abstract assignment  $q(\mathbf{x}, \mathbf{t}) \in \llbracket p(\mathbf{x}, \mathbf{t}) \rrbracket$ , we have that  $\llbracket I \rrbracket, q(\mathbf{x}, \mathbf{t}) \models \psi^a(\mathbf{x}, \mathbf{t})$ . Furthermore, for every abstract assignment  $q(\mathbf{x}, \mathbf{t})$  such that  $\llbracket I \rrbracket, q(\mathbf{x}, \mathbf{t}) \models \psi^a(\mathbf{x}, \mathbf{t})$ , there is a unique concrete assignment  $p(\mathbf{x}, \mathbf{t})$  such that  $q(\mathbf{x}, \mathbf{t}) \in \llbracket p(\mathbf{x}, \mathbf{t}) \rrbracket$  and  $I, p(\mathbf{x}, \mathbf{t}) \models \psi(\mathbf{x}, \mathbf{t})$ .



Earlier, we defined the notion of semantic adequacy for temporal schema mappings in which each constraint had (at most) one temporal variable. We now extend this notion to temporal schema mappings in which constraints may have any number of temporal variables. If  $\mathcal{M}$  is a concrete schema mapping and  $I$  is a concrete source instance, then a concrete target instance  $J$  is *semantically adequate* for  $I$  if  $\llbracket J \rrbracket$  is a universal solution for  $\llbracket I \rrbracket$  w.r.t.  $\mathcal{M}^a$ . Ideally, we would like to have concrete universal solutions for  $I$  that are also semantically adequate for  $I$ . As we have seen in Section 3, however, this is not possible in general, even for temporal schema mappings  $\mathcal{M}$  with a single temporal variable (where we have  $\mathcal{M} = \mathcal{M}^a$ ). In what follows, we identify a sufficient condition for semantic adequacy.

A concrete s-t tgds is *full* if its consequent contains no existentially quantified variables, i.e., it is of the form  $\forall \mathbf{x} \forall \mathbf{t} (\varphi(\mathbf{x}, \mathbf{t}) \wedge \pi(\mathbf{t}) \rightarrow \psi(\mathbf{x}, \mathbf{t}))$ . A concrete schema mapping is *full* if all its concrete s-t tgds are full. Full schema mappings are also known as *Global-as-View* or *GAV* schema mappings, because each full s-t tgds is logically equivalent to a finite set of s-t tgds with a single atom in their consequents.

As an example of a concrete full schema mapping, let  $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$  be the schema mapping in which  $\Sigma_{st}$  consists of the concrete full s-t tgds

$$\sigma_{st}^1 = \forall x_1, x_2, x_3, t_1 (R_1(x_1, x_2, x_3, t_1) \rightarrow T_1(x_1, x_2, t_1)),$$

$$\sigma_{st}^2 = \forall x_1, x_2, x_3, x_4, t_1, t_2 (R_2(x_1, x_2, x_3, t_1) \wedge R_3(x_1, x_4, t_2) \wedge (t_2 \text{ m } t_1) \rightarrow T_2(x_1, x_3, t_2))$$

and  $\Sigma_t$  consists of the concrete target egds

$$\sigma_t^1 = \forall x_1, x_2, x_3, t_1 (T_1(x_1, x_2, t_1) \wedge T_1(x_1, x_3, t_1) \rightarrow x_2 = x_3),$$

$$\sigma_t^2 = \forall x_1, x_2, x_3, t_1, t_2 (T_1(x_1, x_2, t_1) \wedge T_2(x_1, x_3, t_2) \wedge (t_1 \text{ o } t_2) \rightarrow x_2 = x_3).$$

In standard data exchange, full schema mappings have been extensively studied and have been shown to possess a variety of good structural and algorithmic properties (see, e.g., [10, 14]). Unfortunately, as our next result shows, these good properties do not include semantic adequacy.

► **Proposition 7.** *There are a concrete full schema mapping  $\mathcal{M}^+ = (\mathbf{S}, \mathbf{T}, \Sigma_{st}^+, \Sigma_t^+)$  and a concrete source instance  $I^+$  such that the following statements hold:*

1. *There is a concrete universal solution for  $I^+$  w.r.t.  $\mathcal{M}^+$ .*
2. *There is no solution for  $\llbracket I^+ \rrbracket$  w.r.t.  $\mathcal{M}^{+a}$ ; therefore, no concrete universal solution for  $I^+$  w.r.t.  $\mathcal{M}^+$  is semantically adequate for  $I^+$ .*

**Proof.** Let  $\mathcal{M}^+ = (\mathbf{S}, \mathbf{T}, \Sigma_{st}^+, \Sigma_t^+)$  be a concrete schema mapping where  $\Sigma_{st}^+$  consists of the concrete s-t tgds

$$\sigma_{st}^1 = \forall x_1, x_2, x_3, t_1 (R_1(x_1, x_2, x_3, t_1) \rightarrow T_1(x_1, x_2, t_1))$$

$$\sigma_{st}^2 = \forall x_1, x_2, x_3, x_4, t_1, t_2 (R_2(x_1, x_2, x_3, t_1) \wedge R_3(x_1, x_4, t_2) \wedge (t_2 \text{ m } t_1) \rightarrow T_2(x_1, x_3, t_2))$$

and  $\Sigma_t^+$  consists of the concrete target egd

$$\sigma_t = \forall x_1, x_2, x_3, t_1 (T_1(x_1, x_2, t_1) \wedge T_2(x_1, x_3, t_1) \rightarrow x_2 = x_3).$$

Let  $\mathcal{M}^{+a} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}^{+a}, \Sigma_t^{+a})$  be the abstract schema mapping obtained from  $\mathcal{M}^+$  by converting the concrete constraints of  $\mathcal{M}$  to abstract constraints. In this case,  $\Sigma_{st}^{+a}$  consists of the following abstract s-t tgds  $a(\sigma_{st}^1)$  and  $a(\sigma_{st}^2)$  obtained from  $\sigma_{st}^1$  and  $\sigma_{st}^2$ , respectively:

$$a(\sigma_{st}^1) = \forall x_1, x_2, x_3, t_1 (R_1(x_1, x_2, x_3, t_1) \rightarrow T_1(x_1, x_2, t_1))$$

$$\begin{aligned}
a(\sigma_{st}^2) = & \forall x_1, x_2, x_3, x_4, t_1, t_2 \left( R_2(x_1, x_2, x_3, t_1) \wedge R_3(x_1, x_4, t_2) \wedge \exists t_1^-, t_1^+, t_2^-, t_2^+ \left( \right. \right. \\
& R_2(x_1, x_2, x_3, t_1^-) \wedge R_3(x_1, x_4, t_2^-) \wedge R_2(x_1, x_2, x_3, t_1^+) \wedge R_3(x_1, x_4, t_2^+) \wedge \\
& (t_1^- \leq t_1 \leq t_1^+) \wedge (t_2^- \leq t_2 \leq t_2^+) \wedge \forall t_1', t_2' \left( ((t_1^- \leq t_1' \leq t_1^+) \wedge (t_2^- \leq t_2' \leq t_2^+) \rightarrow \right. \\
& R_2(x_1, x_2, x_3, t_1') \wedge R_3(x_1, x_4, t_2') \wedge (R_2(x_1, x_2, x_3, t_1') \wedge R_3(x_1, x_4, t_2') \rightarrow \\
& (t_1' \neq t_1^- - 1) \wedge (t_1' \neq t_1^+ + 1) \wedge (t_2' \neq t_2^- - 1) \wedge (t_2' \neq t_2^+ + 1) \left. \right) \wedge (t_2^+ + 1 = t_1^-) \\
& \left. \left. \rightarrow T_2(x_1, x_3, t_2) \right) \right).
\end{aligned}$$

Moreover,  $\Sigma_t^{+a}$  consists of the following abstract target egd  $a(\sigma_t)$  obtained from  $\sigma_t$ :

$$\begin{aligned}
a(\sigma_t) = & \forall x_1, x_2, x_3, t_1, t_2 \left( T_1(x_1, x_2, t_1) \wedge T_2(x_1, x_3, t_2) \wedge \exists t_1^-, t_1^+, t_2^-, t_2^+ \left( T_1(x_1, x_2, t_1^-) \wedge \right. \right. \\
& T_2(x_1, x_3, t_2^-) \wedge T_1(x_1, x_2, t_1^+) \wedge T_2(x_1, x_3, t_2^+) \wedge (t_1^- \leq t_1 \leq t_1^+) \wedge (t_2^- \leq t_2 \leq t_2^+) \\
& \wedge \forall t_1', t_2' \left( ((t_1^- \leq t_1' \leq t_1^+) \wedge (t_2^- \leq t_2' \leq t_2^+) \rightarrow T_1(x_1, x_2, t_1') \wedge T_2(x_1, x_3, t_2') \right) \\
& \wedge (T_1(x_1, x_2, t_1') \wedge T_2(x_1, x_3, t_2') \rightarrow (t_1' \neq t_1^- - 1) \wedge (t_1' \neq t_1^+ + 1) \\
& \wedge (t_2' \neq t_2^- - 1) \wedge (t_2' \neq t_2^+ + 1) \left. \right) \wedge (t_1^- = t_2^- \wedge t_1^+ = t_2^+) \left. \right) \rightarrow x_2 = x_3 \left. \right).
\end{aligned}$$

Before completing the proof of the proposition, we provide some intuition about the conversion of the concrete temporal constraints of  $\mathcal{M}$  to the abstract temporal constraints of  $\mathcal{M}^{+a}$ . To begin with,  $a(\sigma_{st}^1)$  is the same as  $\sigma_{st}^1$  because  $\sigma_{st}^1$  has a single temporal variable and no Allen's relations. In contrast,  $a(\sigma_{st}^2)$  is quite different from  $\sigma_{st}^2$  because it has two temporal variables and one atomic formula involving Allen's relation **m** (meets). The sub-formula  $\exists t_1^-, t_1^+, t_2^-, t_2^+ \left( R_2(x_1, x_2, x_3, t_1^-) \wedge \dots \wedge (t_2^+ + 1 = t_1^-) \right)$  of  $a(\sigma_{st}^2)$  asserts that: (i) the abstract variables  $t_1$  and  $t_2$  belong to intervals that meet each other (this is the purpose of the sub-formula  $(t_2^+ + 1 = t_1^-)$ ); (ii) all temporal values  $t_1'$  and  $t_2'$  in these intervals have the property that  $R_2(x_1, x_2, x_3, t_1')$  and  $R_3(x_1, x_4, t_2')$  hold; and (iii) there are no bigger intervals for which (i) and (ii) hold. A similar intuition applies to the construction of the abstract target egd  $a(\sigma_t)$ . The correctness of this conversion (i.e., that the abstract constraints  $a(\sigma_{st}^1)$ ,  $a(\sigma_{st}^2)$ , and  $a(\sigma_t)$  are “essentially equivalent” to the concrete constraints  $\sigma_{st}^1$ ,  $\sigma_{st}^2$ , and  $\sigma_t$ ) uses the fact that we use coalesced concrete source instances.

Let  $I^+$  be the concrete source instance whose relations are depicted in Table 11. By applying the semantic function on  $I^+$ , we obtain the abstract instance  $\llbracket I^+ \rrbracket$ , whose relations are depicted in Table 12.

Let  $\text{c-chase}_{\mathcal{M}^+}(I^+)$  be the target instance produced by the concrete chase algorithm on  $I^+$ . The relations of  $\text{c-chase}_{\mathcal{M}^+}(I^+)$  are depicted in Table 13. According to Theorem 5,  $\text{c-chase}_{\mathcal{M}^+}(I^+)$  is a universal solution for  $I^+$  w.r.t.  $\mathcal{M}^+$ .

■ **Table 11** The relations  $R_1$ ,  $R_2$ , and  $R_3$  in the coalesced concrete source instance  $I^+$ .

(a)  $R_1$ .

name	school	position	Ptime
$a_1$	$c_1$	$d_1$	[1, 3]
$a_1$	$c_1$	$d_2$	[2, 4]

(b)  $R_2$ .

name	address	school	Stime
$a_1$	$b_1$	$c_2$	[4, 6]

(c)  $R_3$ .

name	city	Ctime
$a_1$	$e_1$	[1, 4]

We claim that the abstract chase algorithm w.r.t.  $\mathcal{M}^+$  fails on  $\llbracket I^+ \rrbracket$ . To see this, let  $\text{a-chase}_{\Sigma_{st}^{+a}}(\llbracket I^+ \rrbracket)$  be the target instance produced by chasing  $\llbracket I^+ \rrbracket$  with the abstract s-t tgds in  $\Sigma_{st}^{+a}$ . The relations of  $\text{a-chase}_{\Sigma_{st}^{+a}}(\llbracket I^+ \rrbracket)$  are depicted in Table 14. If we now chase

■ **Table 12** The relations  $R_1$ ,  $R_2$  and  $R_3$  in the abstract source instance  $\llbracket I^+ \rrbracket$ .

(a)  $R_1$ .

name	school	position	Ptime
$a_1$	$c_1$	$d_1$	1
$a_1$	$c_1$	$d_1$	2
$a_1$	$c_1$	$d_2$	2
$a_1$	$c_1$	$d_2$	3

(b)  $R_2$ .

name	address	school	Stime
$a_1$	$b_1$	$c_2$	4
$a_1$	$b_1$	$c_2$	5

(c)  $R_3$ .

name	city	Ctime
$a_1$	$e_1$	1
$a_1$	$e_1$	2
$a_1$	$e_1$	3

■ **Table 13** The relations  $T_1$  and  $T_2$  in the target instance  $\text{c-chase}_{\mathcal{M}^+}(I^+)$ .

(a)  $T_1$ .

name	school	Ptime
$a_1$	$c_1$	[1, 3]
$a_1$	$c_1$	[2, 4]

(b)  $T_2$ .

name	school	Ctime
$a_1$	$c_2$	[1, 4]

■ **Table 14** The relations  $T_1$  and  $T_2$  in the abstract target instance  $\text{a-chase}_{\Sigma_{st}^{+a}}(\llbracket I^+ \rrbracket)$ .

(a)  $T_1$ .

name	school	Ptime
$a_1$	$c_1$	1
$a_1$	$c_1$	2
$a_1$	$c_1$	3

(b)  $T_2$ .

name	school	Ctime
$a_1$	$c_2$	1
$a_1$	$c_2$	2
$a_1$	$c_2$	3

$\text{a-chase}_{\Sigma_{st}^{+a}}(\llbracket I^+ \rrbracket)$  with the abstract target  $\text{egd}$  in  $\Sigma_t^{+a}$ , then the abstract chase algorithm fails. This is because the tuple  $(a_1, b_1, c_1, 1)$  in the relation  $T_1$  and the tuple  $(a_1, c_2, 1)$  in the relation  $T_2$  of  $\text{a-chase}_{\Sigma_{st}^{+a}}(\llbracket I^+ \rrbracket)$  trigger the antecedent of the abstract target  $\text{egd } a(\sigma_t)$  in  $\Sigma_t^{+a}$ , hence the abstract chase algorithm fails because it attempts to equate the distinct constants  $c_1$  and  $c_2$ . It follows that there is no solution for  $\llbracket I^+ \rrbracket$  w.r.t.  $\mathcal{M}^{+a}$ . Furthermore, it is not hard to show that if  $J$  and  $J'$  are universal solutions for  $I^+$  w.r.t.  $\mathcal{M}^+$ , then  $\llbracket J \rrbracket$  and  $\llbracket J' \rrbracket$  are homomorphically equivalent. Consequently, no concrete universal solution for  $I^+$  w.r.t.  $\mathcal{M}^+$  is semantically adequate for  $I^+$  (in particular, the concrete universal solution  $\text{c-chase}_{\mathcal{M}^+}(I^+)$  of  $I^+$  w.r.t.  $\mathcal{M}^+$  is not semantically adequate for  $I^+$ ). This completes the proof of the proposition. ◀

Observe that the temporal target  $\text{egd } \sigma_t$  of  $\mathcal{M}^+$  had two temporal atoms in its antecedent. Our next result tells that semantically adequate universal solutions exist for full schema mappings whose temporal target  $\text{egds}$  have at most one temporal atom in their antecedent.

► **Theorem 8.** *Let  $\mathcal{M} = (\mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t)$  be a concrete full schema mapping such that each constraint in  $\Sigma_t$  contains at most one temporal atom. If  $I$  is a concrete source instance, then the following statements hold:*

1. *If a solution for  $I$  w.r.t.  $\mathcal{M}$  exists, then the concrete target instance  $\text{c-chase}_{\mathcal{M}}(I)$  returned by the concrete chase algorithm is semantically adequate for  $I$ .*
2. *If the concrete chase algorithm fails on  $I$ , then there is no solution for  $\llbracket I \rrbracket$  w.r.t. to the abstract schema mapping  $\mathcal{M}^a$ .*

According to Proposition 7, if  $\mathcal{M}$  is a concrete full schema mapping, then there may exist concrete source instances  $I$  for which no concrete universal solution is semantically adequate. As discussed earlier, Golshanara and Chomicki [11] used the concrete n-chase algorithm to construct semantically adequate concrete target instances in the setting of temporal schema mappings with exactly one temporal variable. It is not all clear whether

or not the concrete  $n$ -chase algorithm can be extended to temporal schema mappings with multiple temporal variables. Instead, we introduce a different variant of the chase, which we call the *coalescing* chase algorithm. This algorithm proceeds along the lines of the concrete chase algorithm by introducing labelled nulls or time-stamped nulls as needed when temporal s-t tgds are considered or by equating two values when temporal target egds are considered. However, after each such chase step, the resulting target instance is transformed to a coalesced one before the next chase step is applied (in general, a chase step on a coalesced instance may produce a non-coalesced instance). Note that the concrete  $n$ -chase algorithm applies only two normalization steps, while the number of coalescing steps applied by the coalescing chase algorithm is not fixed.

Our final result asserts that the coalescing chase algorithm produces semantically adequate target instances in the setting of concrete full schema mappings.

► **Theorem 9.** *Let  $\mathcal{M} = (\mathcal{S}, \mathcal{T}, \Sigma_{st}, \Sigma_t)$  be a concrete full schema mapping. If  $I$  is a concrete source instance, then the following statement hold:*

1. *If the coalescing chase does not fail on  $I$ , then the concrete target instance returned by the coalescing chase is semantically adequate for  $I$ .*
2. *If the coalescing chase fails on  $I$ , then there is no solution for  $\llbracket I \rrbracket$  w.r.t. to the abstract schema mapping  $\mathcal{M}^a$ .*

## 5 Concluding Remarks

The work reported here contributes to the development of temporal data exchange. Our main focus was on the pursuit of semantically adequate universal solutions. We showed that such solutions may not exist even for temporal schema mappings with a single temporal variable. Nonetheless, we identified classes of schema mappings for which such solutions exist and also classes of schema mappings for which semantically adequate target instances exist. Along the way, we expanded the original framework of temporal data exchange studied in [11] by considering temporal schema mappings with multiple temporal variables and exploring some of the issues involved in the translation from the concrete model of time to the abstract.

We conclude by describing two directions for further research in this area.

- Explore temporal data exchange for schema mappings that also have target tuple-generating dependencies. Several challenges arise in this case, including the translation of the constraints from the concrete model of time to the abstract model of time, the management of time-stamped nulls, and the design of a suitable chase algorithm.
- Explore temporal data exchange for schema mappings in which the constraints have existentially quantified variables. Several challenges of different nature arise in this case, some of which are similar to challenges in answering queries over temporal data with the help of ontologies (see [5] for a comprehensive survey of that area).

---

## References

- 1 James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.
- 2 James F. Allen. Time and time again: The many ways to represent time. *Int. J. Intell. Syst.*, 6(4):341–355, 1991. doi:10.1002/int.4550060403.
- 3 Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014. URL: <http://www.cambridge.org/9781107016163>.
- 4 Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2):7:1–7:72, 2008. doi:10.1145/1346330.1346332.

- 5 Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyashev. 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 fuer Informatik, 2017. doi:10.4230/LIPICs.TIME.2017.1.
- 6 Pablo Barceló, Jorge Pérez, and Juan L. Reutter. Schema mappings and data exchange for graph databases. In Wang-Chiew Tan, Giovanna Guerrini, Barbara Catania, and Anastasios Gounaris, editors, *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 189–200. ACM, 2013. doi:10.1145/2448496.2448520.
- 7 Iovka Boneva, Jose Lozano, and Slawomir Staworko. Relational to RDF data exchange in presence of a shape expression schema. In Dan Olteanu and Barbara Poblete, editors, *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018*, volume 2100 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: <http://ceur-ws.org/Vol-2100/paper6.pdf>.
- 8 Jan Chomicki and David Toman. Temporal databases. In Michael Fisher, Dov M. Gabbay, and Lluís Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 429–467. Elsevier, 2005. doi:10.1016/S1574-6526(05)80016-1.
- 9 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.
- 10 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005. doi:10.1145/1114244.1114249.
- 11 Ladan Golshanara and Jan Chomicki. Temporal data exchange. *Inf. Syst.*, 87, 2020. doi:10.1016/j.is.2019.07.004.
- 12 Gösta Grahne and Adrian Onet. Anatomy of the chase. *Fundam. Inform.*, 157(3):221–270, 2018. doi:10.3233/FI-2018-1627.
- 13 Abdullah Uz Tansel, James Clifford, Shashi K. Gadia, Sushil Jajodia, Arie Segev, and Richard T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.
- 14 Balder ten Cate and Phokion G. Kolaitis. Structural characterizations of schema-mapping languages. *Commun. ACM*, 53(1):101–110, 2010. doi:10.1145/1629175.1629201.
- 15 David Toman. Point vs. interval-based query languages for temporal databases. In Richard Hull, editor, *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, pages 58–67. ACM Press, 1996. doi:10.1145/237661.237676.



# Knowledge Extraction with Interval Temporal Logic Decision Trees

Guido Sciavicco 

Department of Mathematics and Computer Science, University of Ferrara, Italy  
guido.sciavicco@unife.it

Ionel Eduard Stan 

Department of Mathematics and Computer Science, University of Ferrara, Italy  
Department of Mathematical, Physical, and Computer Sciences, University of Parma, Italy  
ioneleduard.stan@unife.it

---

## Abstract

Multivariate temporal, or time, series classification is, in a way, the temporal generalization of (numeric) classification, as every instance is described by multiple time series instead of multiple values. Symbolic classification is the machine learning strategy to extract explicit knowledge from a data set, and the problem of symbolic classification of multivariate temporal series requires the design, implementation, and test of ad-hoc machine learning algorithms, such as, for example, algorithms for the extraction of temporal versions of decision trees. One of the most well-known algorithms for decision tree extraction from categorical data is Quinlan's ID3, which was later extended to deal with numerical attributes, resulting in an algorithm known as C4.5, and implemented in many open-sources data mining libraries, including the so-called Weka, which features an implementation of C4.5 called J48. ID3 was recently generalized to deal with temporal data in form of timelines, which can be seen as discrete (categorical) versions of multivariate time series, and such a generalization, based on the interval temporal logic HS, is known as Temporal ID3. In this paper we introduce Temporal C4.5, that allows the extraction of temporal decision trees from undiscretized multivariate time series, describe its implementation, called Temporal J48, and discuss the outcome of a set of experiments with the latter on a collection of public data sets, comparing the results with those obtained by other, classical, multivariate time series classification methods.

**2012 ACM Subject Classification** Theory of computation; Theory of computation → Logic

**Keywords and phrases** Interval Temporal Logic, Decision Trees, Explainable AI, Time series

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.9

**Acknowledgements** Computational resources have been offered by the University of Udine, Italy, supported by the PRID project *Efforts in the uNderstanding of Complex interActing SystEms* (ENCASE) and the authors acknowledge the partial support by the Italian INDAM GNCS project *Strategic Reasoning and Automated Synthesis of Multi-Agent Systems*.

## 1 Introduction

A *labelled data set*  $\mathcal{D} = \{D_1, \dots, D_m\}$  is a set of instances described by a set of numerical and/or categorical attributes  $\mathcal{A} = \{A_1, \dots, A_n\}$  and associated to a set of classes  $\mathcal{C} = \{C_1, \dots, C_q\}$ . *Supervised symbolic classification* is the machine learning strategy for extracting an explicit (logical) theory that describes a labelled data set  $\mathcal{D}$ . It is usually opposed to *supervised functional classification*, which includes linear regression, logistic regression, neural networks, and many other black-box and function-extraction mechanisms. There are many symbolic classification methods, which can be broadly distinguished into *tree-based* and *rule-based*. Tree-based classification models can be *single* or *multiple*; multiple tree-based models, however, while still symbolic in nature, are not interpretable in the same sense as single trees are. Tree-based classification models are also known as *decision trees*, and they



© Guido Sciavicco and Ionel Eduard Stan;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 9; pp. 9:1–9:16

Leibniz International Proceedings in Informatics



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

can be described in a very abstract (inductive) way: a decision tree is a class, or it is a  $k$ -ways decision followed by  $k$  decision trees. The introduction of decision trees can be dated back to [10, 11]. The problem of extracting the *optimal* decision tree from a data set is NP-hard [9], which justifies the use of sub-optimal approaches. The *ID3* algorithm [10] is a greedy approach to decision tree extraction; it is based on a simple concept: given a labelled data set  $\mathcal{D}$ , one takes a *decision* by choosing the attribute  $A_i$  and the value(s) on the domain  $A_i$  that return(s) the highest information, obtaining, in general, a  $k$ -ways partition of  $\mathcal{D}$  in  $\mathcal{D}_1, \dots, \mathcal{D}_k$ . The information is measured in terms of the classes that occur in  $\mathcal{D}$  and in  $\mathcal{D}_1, \dots, \mathcal{D}_k$ . Each decision can be expressed as a propositional letter; since alternative branches can be seen as logical (exclusive) disjunctions, and successive decisions on the same branches can be seen as logical conjunctions, a decision tree as a whole can be seen as a set of propositional formulas. In other words, a decision tree is a propositional description of the data set on which it is learned.

A *time series* is a set of variables that change over time, and they can be *univariate* or *multivariate*. Each variable of a multivariate time series is an ordered collection of  $N$  real values, instead of a single value. So, a *labelled temporal data set*  $\mathcal{T} = \{T_1, \dots, T_m\}$  is a set of temporal instances described by a set of temporal attributes  $\mathcal{A} = \{A_1, \dots, A_n\}$ , each being a  $N$ -points time series, and associated to a set of classes  $\mathcal{C} = \{C_1, \dots, C_q\}$ . Multivariate time (or temporal) series emerge in many application contexts. The temporal history of some hospitalized patient can be described by the time series of the values of his/her temperature, blood pressure, and oxygenation; the pronunciation of a word in sign language can be described by the time series of the relative and absolute positions of the ten fingers w.r.t. some reference point; different sport activities can be distinguished by the time series of some relevant physical quantities. In the current literature, time series classification algorithms can be instance-based, feature-based, and timeline-based. *Instance-based* methods are essentially all built on the notion of distance between two time series, by means of which a time series can be classified using, e.g., the *Nearest Neighbor* (NN) algorithm. The most widely accepted notions of distances are the *Euclidean Distance* (ED) and the *Dynamic Time Warping* (DTW) [14]. Intuitively, ED is a one-to-one alignment method, while DTW is one-to-many, as it allows one to compare time series even of different scales. Such methods have been systematically applied to a variety of multivariate time series data sets in [2] (the univariate case is dealt with by the same authors in [3]). *Feature-based* methods, on the other hand, consist of flattening the time series, and describe each one of them via a set of values (e.g., mean, variance, maximum, minimum). These descriptions, in turn, can be used as the input of a static learning algorithm. Feature-based techniques are widespread in the data science community, because they are conceptually simple, and allow one to use familiar learning methods; unfortunately, the theories obtained in this way are not always interpretable, and the quality of the models in term of performances is not always acceptable. An extensive comparison between instance-based and feature-based methods can be found in [7], in which the authors also present an algorithm that allows one to automatically choose the best features to represent a time series data set for it to be classified. Finally, A *timeline* can be considered as the discretized version of a multivariate time series. For each single variable, one produces a set of propositional letters that describe the values of that variable, or its (first, second, ...) derivative, on every possible interval of time, and then describes a multivariate time series on a single line by joining the propositional, interval description of all variables. A general method to translate a multivariate time series into a timeline is described in [13], and *Temporal ID3* [5] can be considered an example of *timeline-based* classification of multivariate time series, that uses the interval temporal logic HS [8] to describe a decision tree.



HS	Allen's relations	Graphical representation
$\langle A \rangle$	$[x, y]R_A[x', y'] \Leftrightarrow y = x'$	
$\langle L \rangle$	$[x, y]R_L[x', y'] \Leftrightarrow y < x'$	
$\langle B \rangle$	$[x, y]R_B[x', y'] \Leftrightarrow x = x', y' < y$	
$\langle E \rangle$	$[x, y]R_E[x', y'] \Leftrightarrow y = y', x < x'$	
$\langle D \rangle$	$[x, y]R_D[x', y'] \Leftrightarrow x < x', y' < y$	
$\langle O \rangle$	$[x, y]R_O[x', y'] \Leftrightarrow x < x' < y < y'$	

■ **Figure 1** Allen's interval relations and HS modalities.

In this paper we design a decision tree learning algorithm, *Temporal C4.5*, that generalizes Temporal ID3 in the same way in which the algorithm C4.5 [12] generalizes ID3, that is, by introducing the possibility of learning from continuous attributes. In this case, however, (non-discretized) time series are described *only* by continuous (time-changing) values, so, in the current version, Temporal C4.5 does not admit categorical attributes. We consider one of the most representative implementations of C4.5, called *J48*, available in the Weka open-source learning suite [15], and we modify it by introducing decisions based on the interval temporal logic HS. The main contributions of this paper are:

- (i) the definition of a general theory of decision trees, which is used to guide our generalization from the static to the temporal case;
- (ii) the first implementation of a symbolic classification algorithm for time series that deals with the raw data (i.e., without applying any pre-abstraction method), whose extracted theory is expressed in the interval temporal logic HS;
- (iii) a comparison of the performances of our implementation against existing methods on the public data used in [2].

## 2 Preliminaries

**Classification of multivariate temporal series.** A *time series* is a set of variables that change over time, and they can be *univariate* or *multivariate*. Each variable (or *channel*) of a multivariate time series is an ordered collection of  $N$  real values, instead of a single value, so that a single time series can be described as follows:

$$T = \begin{cases} A_1 & = a_{1,1}, a_{1,2}, \dots, a_{1,N} \\ A_2 & = a_{2,1}, a_{2,2}, \dots, a_{2,N} \\ \dots & \dots \\ A_n & = a_{n,1}, a_{n,2}, \dots, a_{n,N}. \end{cases} \quad (1)$$

So, a *labelled temporal data set*  $\mathcal{T} = \{T_1, \dots, T_m\}$  is a set of temporal instances described by a set of temporal attributes  $\mathcal{A} = \{A_1, \dots, A_n\}$ , each being a  $N$ -points time series, and associated to a set of classes  $\mathcal{C} = \{C_1, \dots, C_q\}$ . A temporal data set can be viewed as a  $m \times n$  matrix where the  $i$ th row,  $1 \leq i \leq m$ , is a multivariate time series and the  $j$ th column,  $1 \leq j \leq n$ , is an attribute. The *multivariate time series supervised classification problem* is the problem of finding a formula (symbolic classification) or a function (functional classification) that associates multivariate time series to classes.

**Timelines and interval temporal logic.** A multivariate time series can be discretized without eliminating the temporal component. In [13] the authors introduce the notion of *timeline* and present a procedure that transforms multivariate time series into timelines. Time series describe continuous processes; when discretized, it makes little sense to model their values at each point, but, instead, they are naturally represented in an interval-based ontology. Thus, if a static numerical data set is naturally represented in propositional logic, a multivariate time series is naturally represented in an interval temporal logic.

Let  $[N]$  an initial subset of  $\mathbb{N}$  of length  $N$ . An *interval* over  $[N]$  is an ordered pair  $[x, y]$ , where  $x, y \in [N]$  and  $x < y$ , and we denote by  $\mathbb{I}([N])$  the set of all intervals over  $[N]$ . If we exclude the identity relation, there are 12 different Allen's relations between two intervals in a linear order [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 Figure 1, and their inverses, that is,  $R_{\bar{X}} = (R_X)^{-1}$ , for each  $X \in \mathcal{X}$ , where  $\mathcal{X} = \{A, L, B, E, D, O\}$ . Halpern and Shoham's modal logic of temporal intervals (HS) is defined from a set of propositional letters  $\mathcal{AP}$ , and by associating a universal modality  $[X]$  and an existential one  $\langle X \rangle$  to each Allen's relation  $R_X$ . Formulas of HS are obtained by

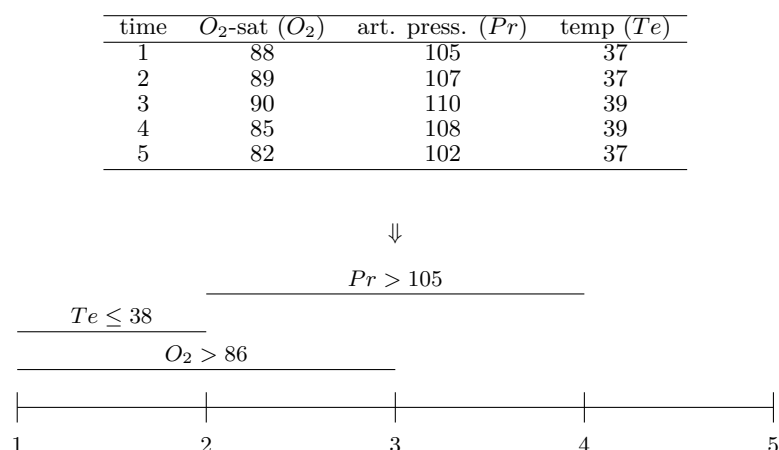
$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle X \rangle\varphi \mid \langle \bar{X} \rangle\varphi, \quad (2)$$

where  $p \in \mathcal{AP}$  and  $X \in \mathcal{X}$ . The other Boolean connectives and the logical constants, e.g.,  $\rightarrow$  and  $\top$ , as well as the universal modalities  $[X]$ , can be defined in the standard way, i.e.,  $[X]p \equiv \neg\langle X \rangle\neg p$ . For each  $X \in \mathcal{X}$ , the modality  $\langle \bar{X} \rangle$  (corresponding to the inverse relation  $R_{\bar{X}}$  of  $R_X$ ) is said to be the *transpose* of the modalities  $\langle X \rangle$ , and vice versa. The semantics of HS formulas is given in terms of *timelines*  $T = (\mathbb{I}([N]), V)^1$ , where  $V : \mathcal{AP} \rightarrow 2^{\mathbb{I}([N])}$  is a *valuation function* which assigns to each atomic proposition  $p \in \mathcal{AP}$  the set of intervals  $V(p)$  on which  $p$  holds. The *truth* of a formula  $\varphi$  on a given interval  $[x, y]$  in an interval model  $T$  is defined by structural induction on formulas as follows:

$$\begin{aligned} T, [x, y] \Vdash p & \quad \text{if } [x, y] \in V(p), \text{ for } p \in \mathcal{AP}; \\ T, [x, y] \Vdash \neg\psi & \quad \text{if } T, [x, y] \not\Vdash \psi; \\ T, [x, y] \Vdash \psi \vee \xi & \quad \text{if } T, [x, y] \Vdash \psi \text{ or } T, [x, y] \Vdash \xi; \\ T, [x, y] \Vdash \langle X \rangle\psi & \quad \text{if there is } [w, z] \text{ s.t. } [x, y]R_X[w, z] \text{ and } T, [w, z] \Vdash \psi; \\ T, [x, y] \Vdash \langle \bar{X} \rangle\psi & \quad \text{if there is } [w, z] \text{ s.t. } [x, y]R_{\bar{X}}[w, z] \text{ and } T, [w, z] \Vdash \psi. \end{aligned} \quad (3)$$

A time series can be seen as a timeline based on a suitable propositional signature. As for example, consider a time series that records medical values of some hospitalized patient: temperature, blood pressure, and oxygenation, as in Figure 2, top. The information can be adequately subsumed into a timeline such as in Figure 2, bottom, provided that a suitable propositional signature is given. In our example, we (arbitrarily) decided that, for instance, the value 38 is an informative splitting point, and a propositional letter ( $Te \leq 38$ ) can be created. In [13], a methodology that allows one to perform such discretization is presented, and in [5], a temporal decision tree extraction method (Temporal ID3), that takes a temporal data set in form of timelines and extracts a decision tree whose edges are labelled with decisions written in HS, is studied. The main limitation of such an approach is that the discretization method does not take into account the predictive capabilities of the decisions (that is, of the propositional letters), because it is run off-line, so to say: in machine learning terms, it is a *filter*. Temporal C4.5 is aimed to close precisely this gap, allowing the propositional signature to emerge *during* the decision tree extraction, not before, precisely as C4.5 does on non-temporal data.

<sup>1</sup> We deliberately use the symbol  $T$  to indicate both a timeline and a time series.



■ **Figure 2** Using timelines to discretize multivariate time series: an example with a single time series.

### 3 A Theory of Decision Trees

**A theory of static decision trees.** Decision trees are a very well-known construct. While in the literature there has been a great effort to improve their implementation, versatility, and applicability, a formal definition of the structure and semantics of decision trees is necessary to correctly define their temporal generalization. In this section, for the sake of simplicity of explanation, we restrict our attention to the case of binary decision trees for binary classification, both in the static and the temporal case. Generalizing our approach to the case of  $k$ -ary trees and multiple classes is immediate.

Consider a labelled static data set  $\mathcal{D} = \{D_1, \dots, D_m\}$  described by the set of attributes  $\mathcal{A} = \{A_1, \dots, A_n\}$  and associated to the classes  $\mathcal{C} = \{Yes, No\}$ . We denote the *domain* of an attribute  $A$  by  $dom(A)$ . The language of static decision trees encompasses a set  $\mathcal{S}$  of *propositional decisions*:

$$\mathcal{S} = \{A \bowtie a \mid A \in \mathcal{A}, a \in dom(A)\}, \quad (4)$$

where  $\bowtie \in \{\leq, =\}$ . Binary static decision trees are formulas of the following grammar:

$$\hat{\varphi} ::= (S \wedge \hat{\varphi}) \vee (\neg S \wedge \hat{\varphi}) \mid C. \quad (5)$$

where  $C \in \mathcal{C}$  and  $S \in \mathcal{S}$ . The symbol  $\vee$  indicates the exclusive *or*, while the symbol  $\wedge$  indicates the classical propositional *and*. Every non-leaf node of a decision tree has two children and every edge is decorated with a decision. Leaves are decorated with a class. A decision  $S$  is interpreted over a single instance  $D$  using classical propositional logic. We say that  $D$  *satisfies* the decision  $A \leq a$  (resp.,  $A = a$ ) if the attribute  $A$  has a value less than or equal to (resp., equal to)  $a \in dom(A)$  in  $D$ , and we use the symbol  $D \models (A \leq a)$  (resp.,  $D \models (A = a)$ ).

A decision tree is interpreted over a labelled data set  $\mathcal{D}$  via the semantic relation  $\hat{\models}_\theta$ , which generalizes  $\models$  from single instances to data sets: we need to define the notion of a data set satisfying  $\hat{\varphi}$  with parameter  $\theta$ , that is,  $\mathcal{D} \hat{\models}_\theta \hat{\varphi}$ . Unlike the classical propositional

logic, in this case the truth relation is parametric; the parameter  $\theta$  formalizes the notion of how well a decision tree  $\hat{\varphi}$  represents  $\mathcal{D}$ . Let  $D$  an instance of  $\mathcal{D}$ . We denote by  $C(D)$  the true class of  $D$ , and by  $\hat{\varphi}(D)$  the class that  $\hat{\varphi}$  predicts for  $D$ . The generic performance of  $\hat{\varphi}$  on  $\mathcal{D}$  can be measured in terms of its *confusion matrix*, which, for each given instance, expresses one of four possible, mutually exclusive, indicators (*true positive*, *true negative*, *false positive*, and *false negative*) by comparing  $C(D)$  and  $\hat{\varphi}(D)$ :

$$\begin{array}{l} \hat{\varphi}(D) = No \\ \hat{\varphi}(D) = Yes \end{array} \begin{array}{cc} C(D) = No & C(D) = Yes \\ \hline \text{True Negative (TN)} & \text{False Negative (FN)} \\ \hline \text{False Positive (FP)} & \text{True Positive (TP)} \end{array} \quad (6)$$

The root of a tree  $\hat{\varphi}$  is associated with the data set  $\mathcal{D}$  on which it is interpreted, and, in general, each node of the tree is associated with a subset  $\mathcal{D}' \subset \mathcal{D}$  and a binary decision  $S$ . A set  $\mathcal{D}'$  is partitioned into two subsets  $\mathcal{D}'_1$  and  $\mathcal{D}'_2$ , that contain, respectively the instances that satisfy  $S$  and those that do not. The subset of  $\mathcal{D}$  associated with a leaf is also labelled with a class  $C$ , meaning that every instance in it is classified with  $C$ , generating a certain amount of misclassifications. From the leaves, one can inductively compute the confusion matrix of each node. The confusion matrix of the root is the one we associate with the tree itself. The rules for  $\hat{\models}_\theta$  are now immediate:

$$\begin{array}{l} \mathcal{D} \hat{\models}_\theta No \\ \mathcal{D} \hat{\models}_\theta Yes \\ \mathcal{D} \hat{\models}_\theta (S \wedge \hat{\varphi}_1) \vee (\neg S \wedge \hat{\varphi}_2) \end{array} \quad \begin{array}{l} \text{if } \theta = \frac{\begin{array}{|c|c|} \hline |\mathcal{D}_{No}| & |\mathcal{D}| - |\mathcal{D}_{No}| \\ \hline 0 & 0 \\ \hline \end{array}}{\quad}, \text{ where} \\ \mathcal{D}_{No} = \{D \in \mathcal{D} \mid C(D) = No\}, \\ \text{if } \theta = \frac{\begin{array}{|c|c|} \hline 0 & 0 \\ \hline |\mathcal{D}| - |\mathcal{D}_{Yes}| & |\mathcal{D}_{Yes}| \\ \hline \end{array}}{\quad}, \text{ where} \\ \mathcal{D}_{Yes} = \{D \in \mathcal{D} \mid C(D) = Yes\}, \\ \text{if } \theta = \theta_1 + \theta_2, \mathcal{D}_1 \hat{\models}_{\theta_1} \hat{\varphi}_1, \text{ and } \mathcal{D}_2 \hat{\models}_{\theta_2} \hat{\varphi}_2, \text{ where} \\ \mathcal{D}_1 = \{D \in \mathcal{D} \mid D \models S\}, \mathcal{D}_2 = \{D \in \mathcal{D} \mid D \models \neg S\}, \\ \mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2, \text{ and } \mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset. \end{array} \quad (7)$$

Observe that computing the confusion matrix on a node generalizes every classical notion of performance, such as accuracy, precision, recall, among others.

**A theory of temporal decision trees.** On the basis of the above notions, we can now define the concept of temporal decision tree.

Let us now consider a labelled temporal data set  $\mathcal{T} = \{T_1, \dots, T_n\}$ , in which every instance is described by the time series  $\mathcal{A} = \{A_1, \dots, A_n\}$  (recall that each attribute is a univariate time series, in this case) and, as before, classified in one of two classes from the set  $\mathcal{C} = \{Yes, No\}$ . We assume that all attributes have the same temporal length,  $N$ . Because time series describe continuous processes, our decisions will be taken on the value of a single channel over an interval of time, and we describe such decisions using propositional interval temporal logic. Unlike static attributes, however, temporal attributes can be analyzed over multiple dimensions. Consider an interval of time  $[x, y]$  and an attribute  $A$  that varies on it. As in the static case we can ask the question  $A \bowtie a$  over the entire interval, where  $\bowtie \in \{\leq, =\}$ , which is positively answered if *every value* of  $A$  in the interval  $[x, y]$  respects the given constraint. But unlike the static case, we do not ask if  $A \bowtie a$  only in the *current interval* but also if *there exists an interval*, related to the current one, in which that holds, so that the decision becomes  $\langle X \rangle (A \bowtie a)$ . This implies, among other things, that the relation  $>$  cannot be defined as the negation of  $\leq$ : when we apply the negation, indeed, we negate *both*  $\langle X \rangle$  and  $\bowtie$ , which amounts to say that if we want to check if  $\langle X \rangle (A > a)$  we have

to do it explicitly. Therefore, in this case,  $\bowtie \in \{\leq, =, >\}$ . Moreover, in order to allow a certain degree of uncertainty, we may relax the requirement  $A \bowtie a$  over a given interval  $[x, y]$  by asking that *at least a certain fraction* of the values of  $A$  in the interval  $[x, y]$  meet this condition; we denote this relaxed decision by  $A \bowtie_\alpha a$ , where  $\alpha \in (0, 1]$  is a real parameter. Finally, we need to remember that in certain applications the values may not be as important as the trends, that is, the value of the *discrete derivative* of  $A$  at a certain degree. We denote the discrete derivative of  $A$  at degree  $z$  by  $A^z$  (identifying  $A^0$  with  $A$ ), and, consequently, a generic temporal decision by  $\langle X \rangle(A^z \bowtie_\alpha a)$ , with  $a \in \text{dom}(A^z)$ . Thus, the language of temporal decision trees encompasses the following set of *temporal decisions*:

$$\mathcal{S} = \{ \langle X \rangle(A^z \bowtie_\alpha a), \langle \bar{X} \rangle(A^z \bowtie_\alpha a) \mid X \in \mathcal{X}, A \in \mathcal{A}, a \in \text{dom}(A^z) \} \cup \{ A^z \bowtie_\alpha a \mid A \in \mathcal{A}, a \in \text{dom}(A^z) \}, \quad (8)$$

where  $\mathcal{X} = \{A, L, B, E, D, O\}$  are interval operators of the language of HS. Temporal decision trees are formulas obtained from (5), in which propositional decisions have been replaced by temporal decisions. A temporal decision is interpreted over a single multivariate time series  $T$  and interval  $[x, y]$ , by using the notion of semantic relation  $\Vdash$  recalled in Section 2; therefore, we use the notation  $T, [x, y] \Vdash \langle X \rangle(A^z \bowtie_\alpha a)$  or  $T, [x, y] \Vdash \langle \bar{X} \rangle(A^z \bowtie_\alpha a)$ . Formally, given a point  $t$ , we denote by  $A^z(t)$  the value of the  $z$ -th discrete derivative of the attribute  $A$  at the point  $t$ , and given an interval  $[x, y]$ , we denote by  $[x, y]^{A^z \bowtie_\alpha a}$  the following set:

$$[x, y]^{A^z \bowtie_\alpha a} = \{t \mid x \leq t \leq y, A^z(t) \bowtie_\alpha a\}. \quad (9)$$

Therefore we have:

$$T, [x, y] \Vdash \langle X \rangle(A^z \bowtie_\alpha a) \quad \text{if} \quad \text{there is } [w, z] \text{ s.t. } [x, y] R_X [w, z] \text{ and} \quad (10) \\ |[w, z]^{A^z \bowtie_\alpha a}| \geq \lceil \alpha \cdot (z - w + 1) \rceil.$$

We now want to define the notion of a temporal data set satisfying  $\hat{\varphi}$  with parameter  $\theta$ , that is,  $\mathcal{T} \hat{\Vdash}_\theta \hat{\varphi}$ . In the static case, from a data set  $\mathcal{D}$  one computes immediately the two data sets  $\mathcal{D}_1$  and  $\mathcal{D}_2$  entailed by a decision  $S$ . In the temporal case, however, this requires more effort. Every instance of the temporal data set  $\mathcal{T}$  associated with the root of  $\hat{\varphi}$  is assigned the reference interval  $[0, 1]$  by default; observe that, since the language of HS encompasses a set of jointly exhaustive operators, this requirement does not decrease the expressive power of temporal decisions. Given a temporal decision  $S$  over  $\mathcal{T}$ , computing the set  $\mathcal{T}_1$  of all instances that do satisfy  $S$  implies assigning to each instance  $T \in \mathcal{T}_1$  a potentially different reference interval; on the contrary, computing  $\mathcal{T}_2$  implies leaving the reference interval of its members unchanged. So, for example, given  $\mathcal{T}$ , in which every instance  $T$  is assigned a reference interval  $[x_T, y_T]$ , and given the decision  $S = \langle A \rangle(A \leq a)$ , we say that:

$$\begin{aligned} \mathcal{T}_1 &= \{T \in \mathcal{T} \mid \exists [y_T, z_T](y_T < z_T \wedge T, [y_T, z_T] \Vdash (A \leq a))\} \\ \mathcal{T}_2 &= \{T \in \mathcal{T} \mid \forall [y_T, z_T](y_T < z_T \rightarrow T, [y_T, z_T] \not\Vdash (A \leq a))\}. \end{aligned} \quad (11)$$

In the particular case in which  $S$  is static, the reference interval does not change for  $\mathcal{T}_1$ , either. For a temporal decision  $S$ , we use the notation  $T \Vdash S$  (resp.,  $T \Vdash \neg S$ ) to identify the members of  $\mathcal{T}_1$  (resp.,  $\mathcal{T}_2$ ). Observe that  $S$  entails unique  $\mathcal{T}_1$  and  $\mathcal{T}_2$ , but not unique (new) reference intervals for the members of  $\mathcal{T}_1$ ; however, this choice is implementative, not theoretical. At this point, the notion of how well a temporal data set  $\mathcal{T}$  satisfies a decision

tree  $\hat{\varphi}$  becomes immediate, and completely equivalent to the static case:

$$\begin{aligned}
\mathcal{T} \Vdash_{\theta} No & \quad \text{if } \theta = \frac{\begin{array}{|c|c|} \hline |\mathcal{T}_{No}| & |\mathcal{T}| - |\mathcal{T}_{No}| \\ \hline 0 & 0 \\ \hline \end{array}}{0}, \text{ where} \\
& \quad \mathcal{T}_{No} = \{T \in \mathcal{T} \mid C(T) = No\}, \\
\mathcal{T} \Vdash_{\theta} Yes & \quad \text{if } \theta = \frac{\begin{array}{|c|c|} \hline 0 & 0 \\ \hline |\mathcal{T}| - |\mathcal{T}_{Yes}| & |\mathcal{T}_{Yes}| \\ \hline \end{array}}{0}, \text{ where} \\
& \quad \mathcal{T}_{Yes} = \{T \in \mathcal{T} \mid C(T) = Yes\}, \\
\mathcal{T} \Vdash_{\theta} (S \wedge \hat{\varphi}_1) \vee (\neg S \wedge \hat{\varphi}_2) & \quad \text{if } \theta = \theta_1 + \theta_2, \mathcal{T}_1 \Vdash_{\theta_1} \hat{\varphi}_1, \text{ and } \mathcal{T}_2 \Vdash_{\theta_2} \hat{\varphi}_2, \text{ where} \\
& \quad \mathcal{T}_1 = \{T \in \mathcal{T} \mid T \Vdash S\}, \mathcal{T}_2 = \{T \in \mathcal{T} \mid T \Vdash \neg S\}, \\
& \quad \mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2, \text{ and } \mathcal{T}_1 \cap \mathcal{T}_2 = \emptyset.
\end{aligned} \tag{12}$$

Analogously to the static case,  $C(T)$  indicates the true class of the multivariate time series  $T$ , and  $\hat{\varphi}(T)$  indicates the class assigned by  $\hat{\varphi}$ .

## 4 Temporal J48

**Entropy-based learning.** In [9], it has been proved that computing the optimal decision tree is an NP-hard problem, where the notion of optimality is expressed as the relation between the height and the performance of the tree. In the perspective of practical applications of decision trees to real-life data, this justifies the use of algorithms that return sub-optimal trees, such as ID3 [10]. ID3 is designed for static, categorical data, at it encompasses  $k$ -ary splits, but, without loss of generality, we focus our attention on binary splits only.

ID3 is able to learn a tree from a purely categorical data set (i.e.,  $\bowtie \in \{=\}$  in Equation (4)), and it uses the concepts of information gain and entropy to select the best decision at every node. By identifying frequencies with probabilities, one defines the *information conveyed* by  $\mathcal{D}$  (or *entropy* of  $\mathcal{D}$ ) by first measuring the amount of instances in  $\mathcal{D}$  that belong to each class  $C_i$  (let us denote this subset with  $\mathcal{D}_{C_i}$ ), and, then, by computing:

$$\text{Info}(\mathcal{D}) = - \sum_{i=1}^{i=2} \left( \frac{|\mathcal{D}_{C_i}|}{|\mathcal{D}|} \log \left( \frac{|\mathcal{D}_{C_i}|}{|\mathcal{D}|} \right) \right). \tag{13}$$

Intuitively, the entropy is inversely proportional to the purity degree of  $\mathcal{D}$  with respect to the class values. *Splitting*, which is the main *greedy* operation in learning a decision tree with ID3, is performed over a specific attribute  $A$ . When restricted to categorical attributes and binary splits, a split depends on a particular attribute  $A$  and a particular value  $a \in \text{dom}(A)$ , which entail two subsets  $\mathcal{D}_1$  and  $\mathcal{D}_2$ ; the former (resp. the latter) contains all those instances  $D$  such that  $D \models (A = a)$  (resp.,  $D \models (A \neq a)$ ) - see Equation (7). Thus, we can compute the *splitting information* of the pair  $A, a$  as follows:

$$\text{InfoSplit}(A, a, \mathcal{D}) = \sum_{i=1}^{i=2} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \text{Info}(\mathcal{D}_i), \tag{14}$$

which implies that the *entropy of attribute*  $A$  is defined as:

$$\text{InfoAtt}(A, \mathcal{D}) = \min_{a \in \text{dom}(A)} \{ \text{InfoSplit}(A, a, \mathcal{D}) \}, \tag{15}$$

and, finally, that the *information gain* of  $A$  is defined as:

$$\text{Gain}(A, \mathcal{D}) = \text{Info}(\mathcal{D}) - \text{InfoAtt}(A, \mathcal{D}). \tag{16}$$

The algorithm ID3 is based on the idea of recursively splitting the data set over the attribute and the value of its domain that guarantee the greatest information gain, until a certain stopping criterion is met. When non-binary splits are allowed, the concept of splitting information must be slightly modified, but the underlying ideas remain. Given a temporal data set  $\mathcal{T}$ , one can use the abstraction algorithm presented in [13] to obtain a discretized version of  $\mathcal{T}$ , in which every multivariate time series becomes a timeline (as explained in Section 2). The algorithm *Temporal ID3* [5] is able to extract a temporal decision tree that follows the general theory explained in the previous section using the same principles of entropy and information gain. As time series are represented in the form of timelines, Temporal ID3 takes decisions of the type  $\langle X \rangle (A = a)$ , where  $A$  is a discretized attribute and  $a$  is one of the possible propositions that emerged from the discretization, and establishes the interval relation, attribute, and propositional letter over which the split is performed according to the entropy principle. In other words, we have:

$$InfoSplit(A, X, a, \mathcal{T}) = \sum_{i=1}^{i=2} \frac{|T_i|}{|D|} Info(T_i), \quad (17)$$

where  $X$  takes values in  $\mathcal{X} \cup \{eq\}$ ,  $eq$  being the interval temporal relation that captures the current interval only, and  $T_1, T_2$  are computed as explained in the previous section (with  $\alpha = 1$  and  $z = 0$ ), and:

$$InfoAtt(A, \mathcal{T}) = \min_{X \in \mathcal{X} \cup \{eq\}, a \in dom(A)} \{InfoSplit(A, X, a, \mathcal{T})\}. \quad (18)$$

The algorithm C4.5 [11] is designed to allow ID3 to cope with numerical data. C4.5 uses exactly the same principles of entropy and information gain introduced for ID3. The main difference, in the static case, lies in allowing  $\bowtie$  to take values in  $\{\leq, =\}$  in propositional decisions; indeed, if  $A$  is numeric, the natural propositional decision is of the type  $A \leq a$ :

$$InfoSplit(A, a, \bowtie, \mathcal{D}) = \sum_{i=1}^{i=2} \frac{|D_i|}{|D|} Info(D_i), \quad (19)$$

where  $\bowtie \in \{\leq, =\}$ , and:

$$InfoAtt(A, \mathcal{D}) = \min_{\bowtie \in \{\leq, =\}, a \in dom(A)} \{InfoSplit(A, a, \bowtie, \mathcal{D})\}. \quad (20)$$

Incidentally, the obtained tree gives a new kind of information, that is, the values of the splitting points that give the most information. If we consider a temporal data set  $\mathcal{T}$ , in which multivariate time series are non-discretized, we obtain a similar result in the temporal case by simply allowing, as above,  $\bowtie \in \{\leq, =, >\}$ ; we may call the resulting algorithm *Temporal C4.5*. Observe that, as explained in the previous section, Temporal C4.5 has two new parameters, that is:

$$InfoSplit(A, X, a, \bowtie, \alpha, z, \mathcal{T}) = \sum_{i=1}^{i=2} \frac{|T_i|}{|D|} Info(T_i), \quad (21)$$

where  $X$  takes values in  $\mathcal{X} \cup \{eq\}$ , and  $T_1, T_2$  are computed as explained in the previous section (but, this time, varying  $\alpha \in (0, 1]$  and  $z \geq 0$ ), and

$$InfoAtt(A, \mathcal{T}) = \min_{\substack{X \in \mathcal{X} \cup \{eq\}, \alpha \in (0, 1], \\ 0 \leq z \leq Max_z, a \in dom(A)}} \{InfoSplit(A, X, a, \bowtie, \alpha, z, \mathcal{T})\}, \quad (22)$$

where  $Max_z$  must be fixed beforehand.

■ **Table 1** A summary of resampled datasets from [2].

Dataset	Train cases	Test cases	Channels	Length	Classes
<b>AtrialFibrillation (AF)</b>	24	6	2	150	3
<b>FingerMovements (FM)</b>	104	26	28	50	2
<b>Libras (LI)</b>	180	45	2	45	15
<b>LSST (LS)</b>	168	42	6	36	14
<b>NATOPS (NA)</b>	96	24	24	51	6
<b>RacketSports (RS)</b>	96	24	6	30	4
<b>SelfRegulationSCP1 (S1)</b>	96	24	6	150	2
<b>SelfRegulationSCP1 (S2)</b>	96	24	7	150	2
<b>UWaveGestureLibrary (UW)</b>	96	24	3	150	8

**A working implementation.** The open-source learning suite Weka [15] offers the implementation the algorithm C4.5 in Java, called *J48*. The most important distinctive characteristic of one specific implementation over the others is the stopping condition; different stopping conditions may lead to different trees. J48 uses a very intuitive principle: having decided a minimal *purity degree* (i.e., a minimal value of  $Info(\mathcal{D})$ , where  $\mathcal{D}$  is associated to a leaf), the base case of the learner is fired on a node if its purity degree is high enough.

Being object-oriented, such an implementation is ideal to test the predictive capabilities of the trees learned following the schema in Section 3 with minimal (yet, non-trivial) modifications. As a matter of fact, we can keep the entire structure of J48: model construction, stopping condition, and training/test performance indicator calculation and displaying. There are two main modifications required:

- (i) input data representation, and
- (ii) splitting management.

As far as the first point is concerned, we used an internal representation based on the string data type, that implicitly assumes that all temporal attributes have the same length and that there are no missing data. Strings have a simple internal structure, in which each value is separated from the next one by a semicolon. Splitting, on the other hand, is taken care by simply building the necessary Java classes that take care of the possible cases. *Temporal J48*, as we call the resulting implementation, requires the following parameters in addition to those already required by J48:

- (i) the value of  $\alpha$  (which in this first experiment we did not optimized at each decision, unlike the general theory suggests);
- (ii) the value of  $Max_z$ ;
- (iii) the reference intervals policy;
- (iv) the subset of the language HS that one allows during the learning phase.

## 5 Experiments and Results

**Data sets.** In order to design a first systematic test aimed to establish the predictive capabilities of Temporal J48, we considered the public temporal data set from [2]. From it, we have extracted nine data sets, which contain problems that vary from the medical context, to automatic recognition of sing language words, to classification of different racket sports based on the movements performed by the athletes. Some adaptations were necessary, taking into account two aspects:



■ **Table 2** Test results in terms of accuracy. Underlined results are the best ones in the group, and starred results are the absolute best ones.

Dataset	AF	FM	LI	LS	NA	RS	S1	S2	UW
J48 1, 0, 0, 0	<u>83.33</u>	<u>50.00</u>	40.00	30.95	<u>79.17</u>	70.83	<u>66.67</u>	50.00	<u>66.67</u>
J48 1, 1, 0, 0	<u>83.33</u>	42.31	51.11	30.95	75.00	<u>87.50*</u>	<u>66.67</u>	54.17	62.50
J48 1, 1, 1, 1	<u>83.33</u>	42.31	<u>64.44</u>	<u>38.10</u>	62.50	79.17	<u>66.67</u>	<u>62.50</u>	54.17
$ED_I$	83.33	<u>76.92</u>	86.67	<u>42.86*</u>	70.83	79.17	66.67	<u>66.67</u>	87.50
$DTW_I$	<u>100.00*</u>	65.38	<u>91.11*</u>	33.33	<u>87.50*</u>	75.00	66.67	<u>66.67</u>	91.67
$DTW_D$	83.33	57.69	<u>91.11*</u>	40.48	<u>87.50*</u>	<u>83.33</u>	<u>83.33*</u>	<u>66.67</u>	<u>95.83*</u>
T. J48 0.5	66.67	57.69	<u>80.00</u>	23.81	<u>83.33</u>	70.83	<u>83.33*</u>	54.17	62.50
T. J48 0.6	66.67	57.69	71.11	<u>26.19</u>	79.17	<u>79.17</u>	66.67	<u>75.00*</u>	58.33
T. J48 0.7	66.67	53.85	73.33	23.81	75.00	66.67	66.67	66.67	62.50
T. J48 0.8	<u>83.33</u>	<u>80.77*</u>	75.56	<u>26.19</u>	75.00	62.50	66.67	62.50	<u>66.67</u>
T. J48 0.9	66.67	<u>80.77*</u>	71.11	23.81	66.67	62.50	66.67	70.83	<u>66.67</u>

- (i) the intrinsic computational inefficiency of Temporal J48 compared with existing methods, due to the substantial amount of information that can be extracted from its results, and
- (ii) the intrinsic unbalance between training and test cardinalities in the original settings in [2].

We modified the data sets as the result of several initial tests by:

- (i) trimming the number of temporal points for those data sets with too long time series, and, in particular, by limiting all time series to  $N = 150$ , and
- (ii) re-sampling training and test instances to obtain a more standard 80% – 20% ratio.

The resulting situation is summarized in Table 1.

**Experimental settings, results, and discussion.** We tested the effectiveness of Temporal J48 against feature-based and distance-based methods, in terms of test accuracy only. This is a highly limited comparison, as the major strength of our approach lies in the interpretability of the (temporal component of the) resulting model, and such a characteristics does not emerge from a purely numeric performance test. Yet, it is interesting to see how good Temporal J48 performs against non-interpretable methods. The results are summarized in Table 2: underlined results are the best ones for the category (feature-based, distance-based, or symbolic), and starred results are the absolute best ones. As for feature-based models, we considered the standard J48 executed on three combinations of abstractions of the temporal data set; for each channel or attribute we computed mean, standard deviation, skewness, and kurtosis, and we combined them in three different ways, each expressed in Table 2 as a bit mask. So, for example, J48 with mask 1,1,0,0 means running the standard decision tree extraction algorithm on a abstracted data set with exactly two attributes per channel, namely mean and standard deviation. As for distance-based methods, we considered the standard, open-source available methods  $ED_I$ ,  $DTW_I$ , and  $DTW_D$ , which require no parametrization. Finally, the parameter that we have used for Temporal J48 are:  $0.5 \leq \alpha \leq 0.9$ , with 0.1 step,  $Max_z = 0$ , and full HS.

As it can be seen, different temporal data set are best dealt with different approaches. In five out nine cases distance-based methods behaved best; in one case feature-based behaved best, using, in particular, only mean and standard deviation. In all other cases, three, there was a run of Temporal J48 that performed better than every other method. As we have explained, obtaining the highest accuracy was *not* our initial motivation; nonetheless,

comparing our method against the others in terms of (test) accuracy is a good proxy for its performance. Yet, as a matter of fact, our approach overcame our expectations: we obtained an interpretable classification model of each of the data sets, and in three cases our model was also the most performing one, indicating that our approach is worth pursuing further. In this first experiment we did not examine all possible parametrization that Temporal J48 offers; in particular, decisions were taken only on the 0-th derivative (so no trends, and no acceleration/decelerations of trends were taken into account), and the uncertainty value  $\alpha$  was fixed at the same value for all intervals. This suggests that a further analysis of the predictive capabilities of Temporal J48 may result in even better performances. More importantly, in some of the cases, Temporal J48 obtained a nearly perfect classification of some of the classes (i.e., individual ROC curve close to 1); these cases give rise to temporal formulas (which can be read on the resulting tree) which, in a way, describe those classes from the temporal logic point of view.

## 6 Conclusions

In this paper we approached the problem of multivariate time series classification. Existing methods for classification of multivariate time series present good performances in terms of accuracy, but the extracted models are not interpretable, in particular in the temporal component. Distance-based methods are based on the concept of distance between series, and feature-based methods, while compatible with interpretable classifiers, flatten the temporal component of the data set. Based on a recently proposed algorithm (Temporal ID3), which is able to classify previously discretized multivariate time series, we developed Temporal C4.5 and realized its implementation, Temporal J48, following the same principle of describing time series using interval temporal logic. Temporal J48 is compatible with the well-known data mining suite Weka.

The initial results show that the interval temporal logic HS is able to correctly describe the behaviour of multivariate time series. As a matter of fact, the predictive capabilities of Temporal J48 are comparable with those of existing methods, and superior to them in some cases, notwithstanding the fact that temporal decision tree models are interpretable even in the temporal component, and therefore undergo a very constrained learning phase (unlike non-interpretable methods, which are notoriously more adaptable). This suggests that temporal symbolic learning may be a promising topic, taking into account that, by examining the temporal component in an explicit way, several new learning parameters emerge that can be adjusted to improve the performances of the extracted models. Future developments of this line include, but are not limited to, exploring the predictive capabilities of variants of the language HS, and studying the possibility of adapting other well-known symbolic learning algorithms in the same way.

---

## References

- 1 J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- 2 A. Bagnall, H.A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh. The UEA multivariate time series classification archive, 2018. [arXiv:1811.00075](https://arxiv.org/abs/1811.00075).
- 3 A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

- 4 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.
- 5 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 Intelligence (JELIA)*, volume 11468 of *Lecture Notes in Computer Science*, pages 778–793. Springer, 2019.
- 6 T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
- 7 B.D. Fulcher and N.S. Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.
- 8 J.Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.
- 9 L. Hyafil and R.L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.
- 10 J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- 11 J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- 12 J.R. Quinlan. Simplifying decision trees. *International Journal of Human-Computer Studies*, 51(2):497–510, 1999.
- 13 G. Sciavicco, I.E. Stan, and A. Vaccari. Towards a general method for logical rule extraction from time series. In *Proc. of the 8th International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC)*, volume 11487 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2019.
- 14 M. Shokoohi-Yekta, J. Wang, and E. Keogh. On the non-trivial generalization of dynamic time warping to the multi-dimensional case. In *Proc. of the 15th SIAM International Conference on Data Mining*, pages 289–297, 2015.
- 15 I.H. Witten, E. Frank, and M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edition, 2017.
- 16 Junfeng Wu, Li Hua Yao, and Bin Liu. An overview on feature-based classification algorithms for multivariate time series. In *Proc. of the 3rd IEEE International Conference on Cloud Computing and Big Data Analysis*, pages 32–38, 2018.

## A Appendix

■ **Listing 1** One of the Temporal J48 models trained on data set RacketSports.

```

1 <L> var5 <= -2.756591
2   <InvA> var5 <= 0.308951
3     <=> var2 > -0.916901
4       <InvB> var0 <= 2.832243: Badminton_Clear (6.0)
5       [InvB] var0 > 2.832243: Badminton_Smash (1.0)
6       [=] var2 <= -0.916901
7         <B> var3 <= -0.207743
8           <InvB> var0 > 4.115426
9           <D> var0 > 1.452113: Squash_ForehandBoast (3.0)
10          [D] var0 <= 1.452113: Squash_BackhandBoast (1.0)
11          [InvB] var0 <= 4.115426
12          <InvB> var0 <= -0.215688: Badminton_Smash (2.0)
13          [InvB] var0 > -0.215688: Badminton_Clear (3.0)
14          [B] var3 > -0.207743: Squash_ForehandBoast (14.0)
15        [InvA] var5 > 0.308951
16          <InvB> var5 <= -2.27452
17            <InvA> var0 <= -1.044682: Squash_BackhandBoast (3.0/1.0)
18            [InvA] var0 > -1.044682: Squash_ForehandBoast (7.0)
19            [InvB] var5 > -2.27452: Squash_BackhandBoast (21.0)
20 [L] var5 > -2.756591
21   <A> var0 <= 0.098773
22     <InvB> var0 > -0.960139
23       <B> var4 <= 0.625893: Badminton_Smash (16.0)
24       [B] var4 > 0.625893: Badminton_Clear (1.0)
25       [InvB] var0 <= -0.960139: Badminton_Clear (2.0)
26 [A] var0 > 0.098773
27   <L> var4 > 8.703901: Badminton_Smash (4.0)
28   [L] var4 <= 8.703901: Badminton_Clear (12.0)

```

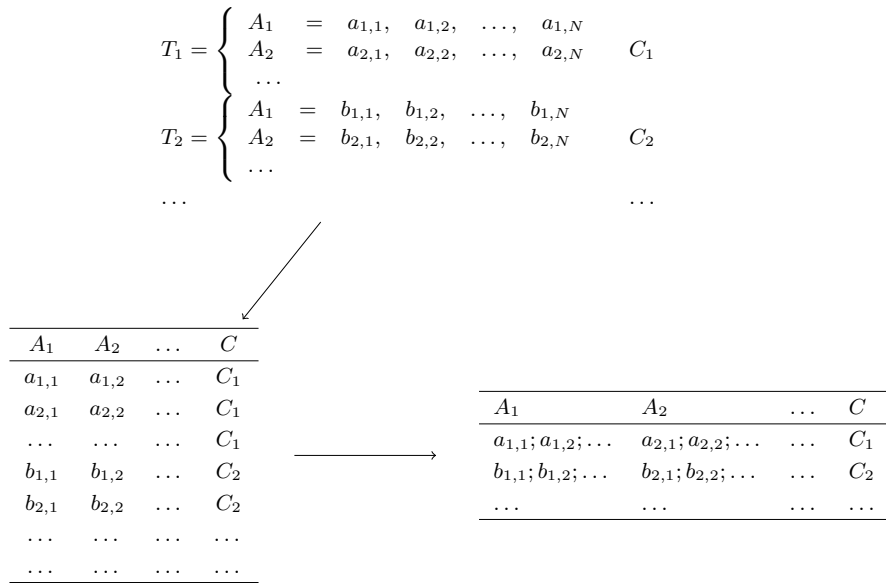
**Time series classification methods.** We briefly review the literature of time series classification. We used some of the available techniques, described here, for a comparison against Temporal J48.

Univariate time series classification is a well-studied problem in the literature; the reader can refer to [3] for an in-depth analysis on state-of-the-art methods for univariate time series classification. For the multivariate case, the classical accepted methods in the literature are *feature-based* or *distance-based*. Feature-based methods are very simple to understand as they are based on extracting numerical or categorical descriptions from each channel. Such descriptions can be simple statistical values (e.g., mean, minimum, maximum, variance, skewness) or yes/no values that refer to the presence of certain patterns (e.g., shapelets). The collection of all descriptions can be then used as input to a classical, static learning algorithm [16]. In some cases, algorithms can be adapted to natively extract patterns from the temporal data sets, as it is the case of [4]. The most widely accepted distance-based methods for multivariate time series classification is the classical *Nearest Neighbour (NN)* algorithm [6] equipped with a proper notion of distance. In the univariate case, given two time series  $T_1 = a_1, a_2, \dots, a_N$  and  $T_2 = b_1, b_2, \dots, b_N$ , the *Euclidean distance (ED)* between  $T_1$  and  $T_2$  is simply the sum of the Euclidean distance between each pair  $(a_i, b_i)$ . The *dynamic time warping distance (DTW)* generalizes such concept by means of an alignment procedure that consists in constructing a  $N \times N$  *distance matrix*, computed via dynamic programming, that allows one to find the alignment that minimizes the point-to-point Euclidean distance. In other words, DTW generalizes the notion of Euclidean distance from single points to single time series. In the multivariate case, in [14] this notion of distance is further generalized in two versions, named  $DTW_I$  (*independent DTW*) and  $DTW_D$  (*dependent DTW*), which differ by how the different channels are combined into a single distance. Thus, distance-based multivariate time series classification is traditionally solved via  $ED_I$  (the independent multivariate generalization of the Euclidean distance),  $DTW_I$ , or  $DTW_D$ . Feature-based and distance-based methods present similar drawbacks: feature-based extract an explicit theory of a temporal data set, but such a theory is hardly interpretable, as it is written in the language of abstract indicators, and non-temporal, as the temporal component plays no role, while distance-based methods solve the classification problem in a black-box way, without extracting any symbolic theory at all. So, the methods of both groups are, in a way, non-interpretable. Finally, using Temporal ID3 (Section 2) to classify previously discretized multivariate time series can be thought of as a *timeline-based* classification method. In this particular taxonomy, Temporal C4.5 is a *symbolic* time series classification method.

**Temporal J48 internal representation.** In Figure 3, we see how data are represented in Temporal J48. Data are abstractly represented as at the top of the figure; the same data present naturally as a matrix, as at the bottom-left of the figure. Finally, we internally represent using a *string* data type, as in the bottom-right of the figure.

**A decision tree learned by Temporal J48.** Let analyze, more in depth, some of the results of our test with Temporal J48.

Consider the temporal data set RacketSports (see Section 5), in which each multivariate time series describes the movements of an athlete playing badminton or squash whilst wearing a smart watch, which relayed the  $X, Y, Z$  coordinates for both a gyroscope and an accelerometer to a smart phone. More in particular, four classes are identified, two movements during the activity of playing squash, that is, *back-hand boast* and *fore-hand*

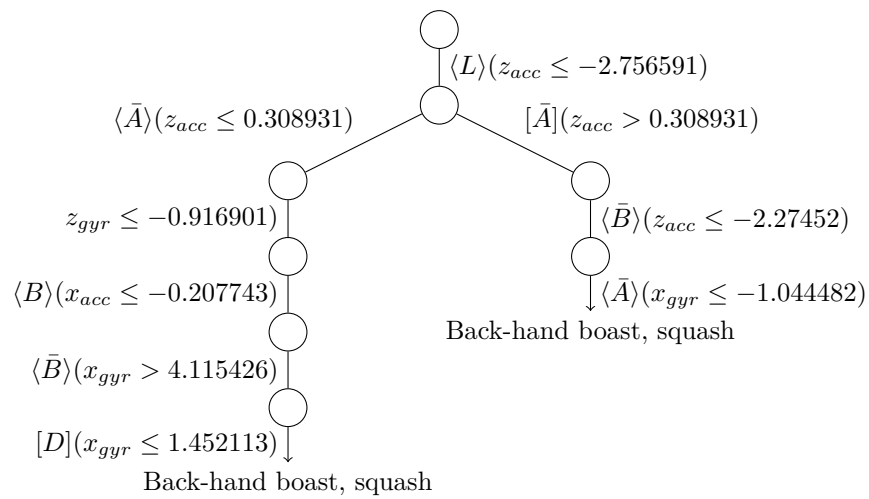


■ **Figure 3** Representation of a temporal data set: original data set (top), natural, tabular representation (bottom, left), and Temporal J48 internal representation (bottom, right).

■ **Table 3** Test performances for the RacketSports data set.

TP	FP	Prec.	Rec.	F-M	MCC	ROC	PRC	Class
0.66	0.05	0.80	0.66	0.72	0.65	0.80	0.61	Smash, badminton
0.83	0.11	0.71	0.83	0.76	0.68	0.86	0.63	Clear, badminton
0.66	0.00	1.00	0.66	0.80	0.77	0.83	0.75	Fore-hand boast, squash
1.00	0.11	0.75	1.00	0.85	0.81	0.94	0.75	Back-hand boast, squash

*boast*, and two movement during the activity of playing badminton, that is, *smash* and *clear*. These movements are described by six channels, which contain the values of the sensors attached for each physical dimension at each moment of time. These variables are codified as follows:  $Var_0, Var_1$ , and  $Var_2$  (resp.,  $Var_3, Var_4$ , and  $Var_5$ ) are the gyroscope (resp., accelerometer) values for  $X, Y$  and  $Z$ . Run with  $\alpha = 0.6$  on this data set, Temporal J48 returned the tree in Listing 1, which has, in test phase, the performances shown in Tab. 3. By focusing on the squash back-hand boast movement only, which has a 0.94 ROC area, one can extract from Listing 1 a formula of HS, shown in Figure 4, that describes such a movement along the temporal component. This proves that our model extraction method allows one to *interpret* the underlying theory. In opposition, feature-based methods flatten the temporal component, so while a symbolic theory is extracted, it is not temporal, and distance-based methods do not extract a theory at all.



■ **Figure 4** Extracted theory for the movement of back-hand boast during squash.

# Window-Slicing Techniques Extended to Spanning-Event Streams

Aurélie Suzanne 

Université de Nantes, France  
Expandium, Saint-Herblain, France  
aurelie.suzanne@ls2n.fr

Guillaume Raschia 

Université de Nantes, France  
guillaume.raschia@ls2n.fr

José Martinez

Université de Nantes, France  
jose.martinez@ls2n.fr

Damien Tassetti

Université de Nantes, France  
damien.tassetti@etu.univ-nantes.fr

---

## Abstract

---

Streaming systems often use slices to share computation costs among overlapping windows. However they are limited to instantaneous events where only one point represents the event. Here, we extend streams to events that come with a duration, denoted as spanning events. After a short review of the new constraints ensued by event lifespan in a temporal sliding-window context, we propose a new structure for dealing with slices in such an environment, and prove that our technique is both correct and effective to deal with such spanning events.

**2012 ACM Subject Classification** Information systems → Stream management

**Keywords and phrases** Data Stream, Spanning-events, Temporal Aggregates, Sliding Windows

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.10

## 1 Introduction

Windows have become a pillar of streaming systems. By keeping only the most recent data, they transform infinite flows of data into finite data sets, allowing aggregate functions. These aggregates continuously summarize the data, providing useful insights on the data at a low memory cost. Sliding windows advance across time, and, in many cases, two successive windows share events, leading to redundancy in computation between consecutive or intersecting windows. This redundancy can be avoided. Techniques such as slicing allow to pre-compute aggregates on sub-parts of the windows which can then be shared among several others.

However, these optimisations, and streaming systems in general, were, up to now, limited to instantaneous events only, i.e., points in time (denoted PES). In this paper, we extend streaming systems and their slicing techniques to spanning-event streams (denoted SES) where events are not assigned to a single point in time but rather to a time interval. This extension cannot be done trivially, as lifespan of events imposes a cost on slice computations. Indeed, one spanning event can appear in several slices, which implies that aggregates, such as the count of events, cannot be deduced from the associated slices.

Figure 1 illustrates this problem, the first window contains five events, while the sum of events we can deduce from the associated slices is nine. With point events, both the direct sum of events and the sum from the slices return three.



© Aurélie Suzanne, Guillaume Raschia, José Martinez, and Damien Tassetti;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

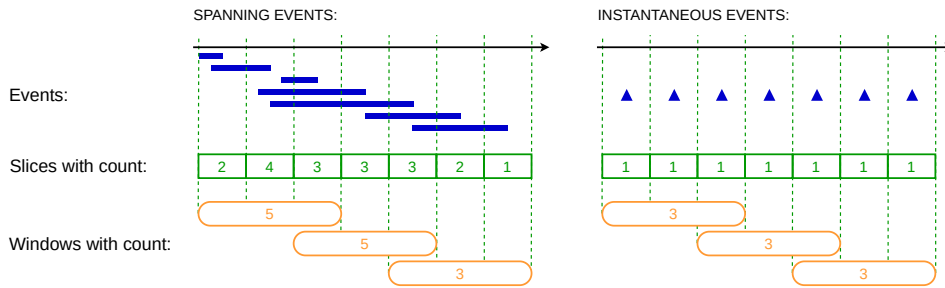
Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 10; pp. 10:1–10:14

Leibniz International Proceedings in Informatics



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

## 10:2 Window-Slicing Techniques Extended to Spanning-Event Streams



■ **Figure 1** Slices and windows with point vs. spanning events.

Applications of these spanning events can be found in particular in monitoring systems dealing with spanning events like telecommunications or transportation. Let us consider a telecommunication network with spanning events like phone calls, and antennas transmitting them. Antenna monitoring consists in retrieving, every five minutes, the number of devices connected to an antenna during the last fifteen minutes. With point events we have either the connection or disconnection of a device to an antenna as an indicator, but cannot use both at the same time. With spanning events we can use the phone call interval, hence improving the accuracy of our metric. In that case, events with device connection, disconnection, or on-going call, can all three be counted correctly as connected to the antenna.

In this paper, we propose a novel slicing technique designed for SES. This technique (1) adapts its structure to aggregate functions, (2) changes the workflow for event insertion to comply with the constraints of event lifespan, and (3) can be plugged into various stream slicer techniques. In order to do so, Section 2 reviews prior work done in the stream optimization field. Then, Section 3 provides background information and extends streams and windows to spanning events. This allows for presenting and extending slice structure to spanning events in Section 4. In Section 5 we study algorithms and complexity which can be achieved with this new structure, and experiment with it in Section 6. We conclude in Section 7.

## 2 Related Work

Commonly, in a stream, data is processed in an append-only continuous flow of point events which cannot be stored. To compute aggregates on this stream, we need to bound it in time, which allows for finite data set [2]. This is the rationale for windows. Many window flavors exist [5, 14] and this paper focuses on *temporal sliding windows*. These windows have the particularity to advance with time independently from the stream, using two parameters: the size or range  $\omega$ , and the step  $\beta$  which determines how fast the window advances in time. Overlaps can happen in such windows as soon as  $\omega > \beta$ , e.g., a window of size  $\omega = 15$  minutes advancing each  $\beta = 5$  minutes.

Many techniques have been proposed to improve the performances of sliding windows on PES systems: *buffers*, *buckets*, *aggregate trees*, *slices*, and their compositions [21]. Naïve techniques keep all the events: *buffers* do just that, whereas *Buckets* [12] split them into sets (e.g., one per window). *Buckets* are especially used for out-of-order processing [13]. With overlapping windows, these methods lead to redundancy in computations as well as to spikes in the system when aggregates are released. Other techniques improve the system efficiency with shared computations among windows. *Aggregate trees* store partial aggregates in a hierarchical data structure. *Slices* divide the stream into finite non-overlapping sets of data, keeping only one aggregate value for each slice, on which the final aggregates are then computed. Slices can then be shared among windows. Further applicable optimization techniques depend on the window type used and on the presence of out-of-order events [19, 21].



In this paper, we are particularly interested in slicing techniques for which inner window optimizations can be adapted to spanning events, and their specific constraints. Several methods have been proposed for slices, which differ mainly in the way they create and release slices. *Panes* [11] provides a first “naïve” implementation, which partitions the stream into constant size slices, equal to  $\gcd(\beta, \omega)$ . This technique generates a high number of slices when the range is not divisible by the step. This led to defining new methods. *Pairs* [10] creates at most two slices per step. Compared to *Panes*, this method reduces by a factor of two the number of slices [17]. These two techniques allow to deal with out-of-order streams [22]. *Cutty* [5] starts new slices at the beginning of windows, and final aggregation executes when needed, without closing any slice, which reduces the number of slices per window by a factor of two compared to *Pairs*. However, this comes at a cost and reduces effective bandwidth of the stream by sending additional events called punctuation which mark each slice start. As this technique does not separate slices when window ends, it does not allow for out-of-order events [22]. Last but not least, *Scotty* [22] takes into account out-of-order events with another enrichment of the stream that indicates the start of slices as well as the release of windows, and a system to update slices for which end time has passed when out-of-order events arrive. It also has the specificity of being able to deal with both sliding and session windows, where window bounds are defined by activity and inactivity.

These techniques can be further improved with final aggregation techniques. They define how to merge slice sub-aggregates. Naïve techniques iterate over all the slices (as done in this communication), however this leads to bottlenecks when computing final results. Other techniques compensate this by using aggregate trees or indexes, e.g., *B-Int* [3], *FlatFAT* [20], *FlatFIT* [16], *TwoStacks* [7], and *DABA* [18], *SlickDeque* [17].

### 3 Preliminaries

#### 3.1 Time, time intervals, and time-interval comparisons

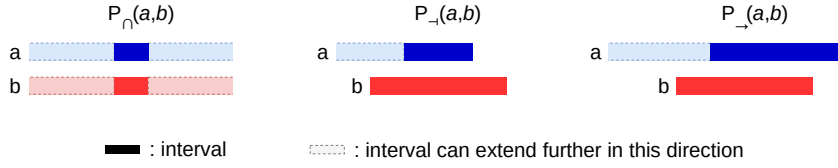
In this paper, we introduce and use time intervals. Time is represented as an infinite, totally ordered, discrete set  $(\mathbb{T}, \prec_{\mathbb{T}})$ , where each time point  $c$  is called a *chronon* [4]. (Discrete) intervals are expressed with a lower and an upper bound, as pairs  $(\ell, u) \in \mathbb{T} \times \mathbb{T}$  with  $\ell \prec_{\mathbb{T}} u$ . By convention, an interval is written as a left-closed–right-opened interval  $[\ell, u)$ . We denote by  $\mathbb{I} \subset \mathbb{T} \times \mathbb{T}$  the set of time intervals. For any  $t \in \mathbb{I}$ ,  $\ell(t) \in t$  and  $u(t) \notin t$  are respectively the lower and upper bounds of the interval  $t$ . A chronon  $c$  can be represented by the interval  $[c, c + 1)$ .

Two intervals can be compared with the thirteen Allen’s predicates [1]. We introduce three new predicates as a combination of Allen’s base predicates, which will prove useful hereafter. They are illustrated in Figure 2. Their corresponding formal definitions are:

- $P_{\cap}(a, b) := \ell(a) < u(b) \wedge \ell(b) < u(a)$ , i.e., time intervals  $a$  and  $b$  have at least one chronon in common;
- $P_{\rightarrow}(a, b) := \ell(b) < u(a) \leq u(b)$ , i.e., time interval  $a$  ends in  $b$ , an asymmetric relation;
- $P_{\leftarrow}(a, b) := \ell(a) < u(b) < u(a)$ , i.e.,  $a$  overlaps and goes beyond  $b$ , asymmetric too.

#### 3.2 Spanning-event Stream

Within our SES framework, we consider that each event comes with a time interval, adding the notion of lifespan to events. Instantaneous events can still be modelled with a single-chronon interval. We consider that events are received after their ending.



■ **Figure 2** The three interval comparison predicates used in this paper.

Spanning-event stream (SES) is defined in Equation 1 shown below.  $\Omega$  corresponds to any set of values, either structured or not, that brings the contents of each event  $e \in S$ , and  $t$  is the time interval of an event  $e$ . We denote by  $t(e)$  this value for an event  $e$ .

$$S = (e_i)_{i \in \mathbb{N}} \text{ with } e_i = (x, t) \in \Omega \times \mathbb{I} \quad (1)$$

The order of events in the stream obeys the constraint:  $\forall (e, e') \in S^2, e < e' \Leftrightarrow u(t(e)) <_{\mathbb{T}} u(t(e'))$ , which means that events are ordered by their end-time. In this paper, only on-time events are considered. This implies that events are received as soon as their upper bound is reached, at time  $u(t(e))$ . The set of streams is denoted by  $\mathcal{S}$ .

### 3.3 Aggregate Functions

In streaming systems, data load often makes it impossible to process data individually by an end-user application. A common solution is to use aggregates. Many aggregate functions exist, which can often be studied by categories rather than individually. We use two classifications, based on their properties about slices and spanning events.

Table 1 presents how common aggregate functions [7, 20] are classified.

One can distinguish several **algebraic properties** [3, 6, 8, 9, 17] such as: *distributive*: a stream can be split into sub-streams and some functions allow to compute an aggregate from sub-aggregates, e.g., a **sum** can be computed from a set of sub-sums; *algebraic*: an aggregate can be computed from a list of distributive aggregates, e.g., a **mean** can be computed from **sum**'s and corresponding **count**'s sub-aggregates; *holistic*: some functions do not belong to any of the above categories, e.g., **median**. No constant bound on storage applies for the last category of functions. This leads to using specifically tailored algorithms. For this reason, this last category is not studied in this paper.

One can also distinguish among **accumulative properties** [15]: *cumulative*: an aggregate is an accumulation of all the events, e.g., **count** adds one for each event; *selective*: an aggregate keeps only one event, in its original form, e.g., **max** keeps only the biggest value. Cumulative functions are sensitive to event duplicates. These do happen as a consequence of working with SES. Therefore, we shall study these categories separately.

### 3.4 Temporal Sliding Window with SES

Windows, in general, allow stream computations by dividing the stream into time intervals of interest where consecutive events can be aggregated. Sliding windows in particular are associated to a time interval. This time interval is defined in advance thanks to two parameters: the range  $\omega$ , and the step  $\beta$ . Window life-cycle goes through several steps: *window creation* is triggered when the window lower bound is reached, and *window release* is triggered as soon as the upper bound is reached. Between these two triggers, the window accumulates all the incoming events of interest. At window release, the system computes the aggregate related to this window.

■ **Table 1** Classification of most common aggregate functions.

	Aggregate function	Algebraic property	Accumulative property
<b>sum-like</b>	count, sum	distributive	cumulative
	mean, standard-deviation	algebraic	cumulative
<b>max-like</b>	max, min	distributive	selective
	argmax, argmin	algebraic	selective
	maxCount, minCount	algebraic	cumulative
<b>collect-like</b>	collect, concatenate (string)	holistic	cumulative
	ith-youngest	holistic	selective
<b>median-like</b>	median, percentile	holistic	cumulative
	ith-smallest	holistic	selective

We define such a set of windows in Equation 2.  $S_{w_i}$  is a finite sub-stream of  $S$  containing the events that occurred in window  $w_i$ , and  $t_i$  is the time interval, also denoted  $t(w_i)$ .

$$W_S = (w_i)_{i \in \mathbb{N}} \text{ with } w_i = (S_{w_i}, t_i) \in \mathcal{S} \times \mathbb{I} \quad (2)$$

However, SES obliges us to investigate modifications for temporal sliding windows. Firstly, window creation is not impacted by SES as the bounds of the window,  $t(w)$ , are independent from the stream contents. When a new event arrives, it is assigned to the current window. In contrast, and due to its duration, an incoming event may need to be assigned to *past* windows too. Thus, triggering a window release at its upper bound would lead to losing events for this window because they are still on-going and will be known only in the future.

To overcome this problem, we introduce a time-to-postpone (TTP). Its role is to delay the window release, with a trigger now happening at the window upper bound plus the TTP. Of course, this value needs to be chosen very carefully as long-standing events could still arrive after the TTP. Several ways exist to define this value, from a fixed user-defined value to an evolving value continuously learned by the system. Accurate definition of the TTP is outside of the scope of this paper, as our topic here is centered on slicing optimizations. Hence we only consider the simplest case of a pre-assigned TTP larger than the largest event expected. With this delay, an event can now be taken into account by any unreleased window. Event affiliation to a window is defined accordingly to the intersection predicate  $P_{\cap}$  from Section 3.1.

## 4 Slices

### 4.1 Point-event Slices

Slicing techniques divide windows into slices into which events are kept in the form of continuously updated sub-aggregates. These sub-aggregates are then combined in order to compute the window aggregate. Advantages of slices are numerous: (1) they limit memory usage by requiring only one aggregate per slice instead of buffering all the events, and (2) they reduce spikes in the system at window release since only partial aggregates need to be computed, and (3) they allow for computation sharing among windows [5, 11, 22].

We define a sequence of slices in Equation 3.  $\phi \in \Phi$  is an internal slice structure that stores the partial aggregate value, and  $t$  is the slice interval, also denoted by  $t(\gamma)$ . The internal slice structure  $\phi$  changes depending on the aggregate function used, e.g., **sum** would keep a sum for each slice, whereas **mean** would keep a sum and a count. A list of internal

## 10:6 Window-Slicing Techniques Extended to Spanning-Event Streams

structures for all common aggregate functions can be found in [20].

$$\Gamma_{W,S} = (\gamma_i)_{i \in \mathbb{N}} \text{ with } \gamma_i = (\phi, t) \in \Phi \times \mathbb{I} \quad (3)$$

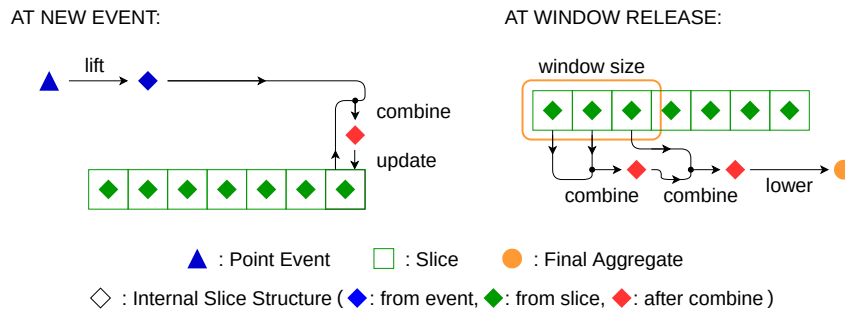
Slices obey two properties. They ensure that  $\Gamma_{W,S}$  is a time partitioning of the stream  $S$ , w.r.t. a family window  $W$ . These properties are:

**P1**  $\forall (i, j) \in \mathbb{N}^2, i \neq j \rightarrow \neg P_{\cap}(t(\gamma_i), t(\gamma_j))$ : two slices cannot overlap;

**P2**  $u(t(\gamma_i)) = \ell(t(\gamma_{i+1}))$ : two successive slices meet, in Allen's meaning.

To use our slices, we adopt the incremental aggregation method introduced by Tangwongsan et al. [20] and re-used in [21]. This approach is based on three functions. They are illustrated in Figure 3, and informally described as follows:

- **lift** :  $S \rightarrow \Phi$  transform events for a future insertion in slices. It is used when an event arrives in the system, and transforms it into the internal slice structure.
- **combine** :  $\Phi^2 \rightarrow \Phi$  gathers two slices internal structures into a new one. This operation is used both at event insertion and at window release, as shown in Figure 3.
- **lower** :  $\Phi \rightarrow \text{Agg}$ : computes the final aggregate from a slice internal structure, in order to actually release a window.



■ **Figure 3** Usage of functions **lift**, **lower** and **combine** to insert events and release aggregates in PES.

As an example, we want to know, every 5 minutes, the number of device disconnections and the maximum call duration for an antenna for the past 15 minutes. For such a query, the slice structure would consist of partial counts and max. An illustrated scenario is as follows:

- In the **initial structure**,  $n$  represents the partial counts, and max the partial maximums

time	0	5	10	15	20	25	30	35
$n$	8	19	15	18	14	12	16	
max	20	63	19	33	12	47	14	

- When a **new event** with a call duration of 18 minutes arrives at time 34, then:
  - The event is transformed by **lift**, giving ( $n = 1, \text{max} = 18$ );
  - This lifted event is **combine**'d to the last slice, as illustrated below.

time	0	5	10	15	20	25	30	35
$n$	8	19	15	18	14	12	17	
max	20	63	19	33	12	47	18	

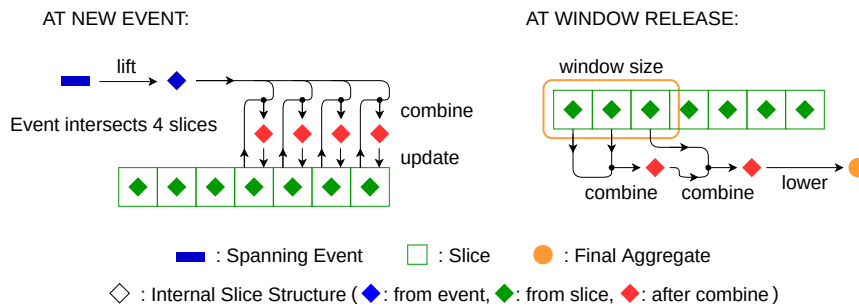
- Finally, **at window release**, we apply two final steps:
  - We use **combine** incrementally for the first three slices. A first **combine** is applied on the two first slices and gives ( $n = 27, \text{max} = 63$ ). Then a second **combine** on the previous result and third slice results in ( $n = 42, \text{max} = 63$ ).
  - We apply **lower** to output a count of 42 and a maximum of 63.

## 4.2 Spanning-event Slices

Now, we have to adapt slicing technique to SES. With spanning events, one event can find itself in several slices as shown in Figure 1, which is not the case for point events since slices are non-overlapping. Hence, the first adaptation is to give the possibility to update several slices during the insertion, at the combine level. However, this quickly leads to a duplication problem which we need to leverage. As their sensitivity to duplication varies, we shall study selective and cumulative aggregate functions separately.

### 4.2.1 Selective Aggregate Functions

The former is the simpler. With selective functions, e.g., `max`, duplication of events is not a problem, as only one event is kept. Hence, we can mostly compute selective aggregate functions with the same slice structure used with point-event streams. Figure 4 illustrates the slicing workflow for selective aggregate functions. We can see that only the combine step at event insert-time is modified. Now, *all the slices that have a non-empty intersection with the event* need to be updated, not only the last one.



**Figure 4** Usage of functions `lift`, `lower`, and `combine` to insert events and release aggregates in SES with selective aggregate functions.

Back to our example, we keep the selective function part and compute the maximum call duration with SES as follows:

- In the **initial structure**, the partial maximums are associated to each slice.

time	0	5	10	15	20	25	30	35
max	20	63	19	33	12	47	14	

- When a **new event** arrives, say a call duration of 18 minutes at time 34:
  - The event is `lift`'ed into (18);
  - This lifted event is `combine`'d to *each related slice*, the last four slices here.

time	0	5	10	15	20	25	30	35
max	20	63	19	33	18	47	18	

- Finally, at **window release**, we apply the two final steps:
  - We use `combine` for the three first slices. This gives (63) after the first `combine`, next (63) again after the second one.
  - We apply `lower` to output a maximum of 63.

### 4.2.2 Cumulative Aggregate Functions

Cumulative functions accumulate the data, hence they are sensitive to event duplication among slices. To compensate for this problem, we duplicate the internal structure, as shown in Equation 4. We propose the  $(\phi, \varphi)$ -structure to distinguish between events that end

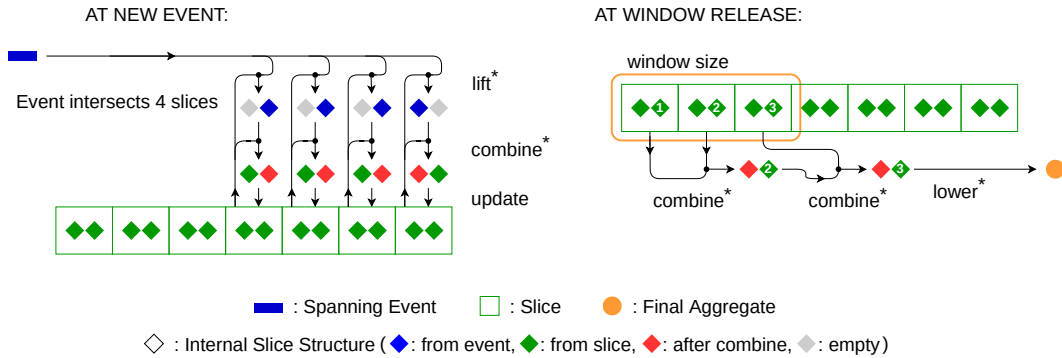
## 10:8 Window-Slicing Techniques Extended to Spanning-Event Streams

in the slice, defined with  $P_{\leftarrow}(t(e), t(\gamma))$ , from events that follow after the slice, defined by  $P_{\rightarrow}(t(e), t(\gamma))$ . To show that our extension performs the expected computations, it is worth to note that  $P_{\cap} = P_{\leftarrow} \vee P_{\rightarrow}$  and  $P_{\leftarrow} \wedge P_{\rightarrow} = \perp$ . In other words, an event that overlaps a given time interval, either finishes inside its time range or goes beyond.

$$\Gamma_{W,S} = (\gamma_i)_{i \in \mathbb{N}} \text{ with } \gamma_i = (\phi, \varphi, t) \in \Phi^2 \times \mathbb{I} \quad (4)$$

The aggregation process uses modified versions of the lift, combine, and lower operators as described in Section 4.1. This is illustrated in Figure 5. A formal definition of these modified versions is given in Table 2. The new functions operate in the following way:

- **lift\*** :  $S, \mathbb{I} \rightarrow \Gamma_{W,S}$ : classifies each event to the  $(\phi, \varphi)$  slice structure. To select which part will be initialized the  $P_{\leftarrow}$  and  $P_{\rightarrow}$  conditions are used, resp. for  $\phi$  and  $\varphi$ . Each event is eligible to only one of them, and the non-eligible part is left empty. Basically, as one can see in Figure 5, the last part of the event will contribute to the  $\phi$  part of the last slice, and to the  $\varphi$  part of all other intersecting slices. This implies that the lift\* operation depends on the interval of the slice, and should be computed for each slice;
- **combine\*** :  $\Gamma_{W,S}^2 \rightarrow \Gamma_{W,S}$ : behaves differently depending on the moment it is triggered. When combine\* is triggered *at event insertion*, it will rely on the raw combine operator from [20] to update as much  $\phi$  as  $\varphi$ . We can however note, as shown in Figure 5, that only one of them will be updated as the event cannot contribute to both at the same time during the lift\* phase. When combine\* is triggered *at window release* it will ignore the  $\varphi$  part of the oldest slice to prevent event duplication. We are sure that an event in  $\varphi$  will contribute to the next slice, either in  $\phi$  or  $\varphi$ . Hence keeping only the more recent  $\varphi$  ensures us neither to duplicate the event nor to forget it. This behavior can be seen in Figure 5 where, at each combine\*, only the  $\varphi$  of the most recent slice is kept.
- **lower\*** :  $\Gamma_{W,S} \rightarrow \text{Agg}$ : merges the distinct parts  $\phi$  and  $\varphi$  to provide the exact aggregate value.



■ **Figure 5** Usage of functions lift\*, lower\* and combine\* to insert events and release aggregates in SES with cumulative aggregate functions. The Internal structure is duplicated to keep track of: the events which end in the slice  $\phi$  on the left, and the events which end after the slice  $\varphi$  on the right.

As neither event duplication nor omission are possible with the  $(\phi, \varphi)$ -structure we claim that all popular cumulative aggregate functions can be used with this new structure.

We continue our example with the cumulative function part, and use these new functions to count the number of devices connected to an antenna with spanning events.

- In the **initial structure**,  $\phi$  and  $\varphi$  represent partial counts.

time	0	5	10	15	20	25	30	35
$\phi$	8	19	15	18	14	12	16	
$\varphi$	25	18	12	14	11	19	16	

- When a **new event** arrive, the phone call with a duration of 18 minutes at time 34:
  - The event is transformed with  $\text{lift}^*$  into  $(\phi = 1, \varphi = 0)$  for the most recent slice, whereas it gives  $(\phi = 0, \varphi = 1)$  for the three previous slices;
  - This lifted event is combined with each related slice, which are the last four slices.

time	0	5	10	15	20	25	30	35
$\phi$	8	19	15	18	14	12	17	
$\varphi$	25	18	12	15	12	20	16	

- Finally, at **window release**, we apply the two final steps:
  - We use  $\text{combine}^*$  for the three first slices. This gives  $(\phi = 27, \varphi = 18)$  after the first  $\text{combine}^*$ , then  $(\phi = 42, \varphi = 12)$  after the second.
  - We apply  $\text{lower}$  to output a (correct) count of 54.

- **Table 2** Extension (\*-form) of slice operators to the  $(\phi, \varphi)$ -structure for SES.

$$\begin{aligned} \text{lift}^*(e : S, t : \mathbb{I}) &\rightarrow (\phi, \varphi, t) : \Gamma_{W,S} & \text{lower}^*((\phi, \varphi, \_) : \Gamma_{W,S}) &\rightarrow y : \text{Agg} \\ \phi &= \text{lift}(e) \text{ if } P_{\leftarrow}(t(e), t) \text{ else } 0_{\text{Agg}} & y &= \text{lower}(\text{combine}(\phi, \varphi)) \\ \varphi &= \text{lift}(e) \text{ if } P_{\rightarrow}(t(e), t) \text{ else } 0_{\text{Agg}} \end{aligned}$$

$$\text{combine}^*((\phi_a, \varphi_a, a) : \Gamma_{W,S}, (\phi_b, \varphi_b, b) : \Gamma_{W,S}) \rightarrow (\phi, \varphi, a \cup b) : \Gamma_{W,S}$$

$$\text{assert } u(a) = \ell(b) \text{ or } u(b) = \ell(a) \text{ or } a = b$$

$$\phi = \text{combine}(\phi_a, \phi_b)$$

$$\varphi = \text{combine}(\varphi_a, \varphi_b) \text{ if } a = b \text{ else } \varphi_{\max\{a,b\}}$$

## 5 Stream Slicer

### 5.1 Algorithms applicable to SES

We are now able to insert events into slices in order to compute correctly window aggregates from the slices. Next, we need a system able to create such slices from the window parameters and the stream. We saw that for sliding windows several such systems already exist to address PES. However one of our requirements is to be able to update past windows, i.e., windows for which their upper bounds are located in the past. Therefore, in our stream slicer each window end must coincide with a slice end. For this purpose, the algorithms *Panes* [11] and *Pairs* [10] are good candidates, while *Cutty*[5] is unsuitable. However, *Panes* creates twice as much slices than *Pairs*. As the goal of a stream slicer is to produce as few slices as possible, in order to reduce insertion and release costs [22], *Pairs* is more appropriate. *Scotty* [22] produces slices for each new window start or end, which makes the method equivalent to *Pairs*. Hence, we shall use the *Pairs* technique in this paper.

### 5.2 Slicing Algorithms

For this section we consider cumulative aggregate functions, and each entering event is directly transformed into our aggregate  $(\phi, \varphi)$ -structure, as defined in Section 4.2.

We use the *Pairs* technique to separate our input stream into slices. It creates up to two slices per step where the first slice is of size  $|t(\gamma_1)| = \omega \bmod \beta$  and the second one of size  $|t(\gamma_2)| = \beta - |t(\gamma_1)|$ . This leads to  $n_\beta = 2$  slices per step if  $\omega \bmod \beta > 0$  else 1, and  $n_\omega = \lceil 2\omega/\beta \rceil$  slices per window if  $\omega \bmod \beta > 0$  else  $\lceil \omega/\beta \rceil$ .



## 10:10 Window-Slicing Techniques Extended to Spanning-Event Streams

Our slice-based SES aggregation process, as exposed in Algorithm 1, uses “an-event-at-a-time” execution model. In this algorithm, one considers  $\tau \in \mathbb{T}$  as the clock, i.e., an infinite time counter starting from  $0_{\mathbb{T}}$ . The  $n_{\omega}$  and  $n_{\beta}$  values are initialized (line 3 - *init\_nb\_slices*) with the above formulas. The tests for the window start and end times (resp. lines 5 and 7) are performed with a  $\mathbb{T}$ -mark incremented by  $\beta$  each time it is reached.  $\delta$  corresponds to the TTP and delays window release. *read\_stream*( $S, \tau$ ) (line 9) retrieves the event  $e$  at current time  $\tau$  if it exists, or nothing. *add\_slices* in Algorithm 2 creates the missing slices for a new window, the total size of which is  $\beta$ . *release\_window* in Algorithm 3 combines  $n_{\omega}$  slices, corresponding to all the slices in a window, and then lowers the results to release a final aggregate. It also deletes the first  $n_{\beta}$  slices (line 4), to advance the slices structure in time of a size  $\beta$ . *insert\_event* in Algorithm 4 insert the event in each applicable slice, starting with the more recent, and stopping as soon as it reaches a non intersecting slice.

### ■ Algorithm 1 SES Slice Aggregation.

---

```

input :  $S \in \mathcal{S}, \omega \in \mathbb{N}, \beta \in \mathbb{N}, \delta \in \mathbb{N}$ 
1  $\tau : \mathbb{T} \leftarrow 0_{\mathbb{T}}$ 
2  $\Gamma : \text{List} \langle (\Phi, \Phi, \mathbb{I}) \rangle$  as Slices  $\leftarrow ()$ 
3  $n_{\omega}, n_{\beta} : \mathbb{N}^2 \leftarrow \text{init\_nb\_slices}(\omega, \beta)$ 
4 while True do
5   if window_begins_at( $\tau$ ) then
6      $\Gamma \leftarrow \text{add\_slices}(\Gamma, \tau, \omega, \beta)$ 
7   if window_ends_at( $\tau - \delta$ ) then
8      $\Gamma \leftarrow \text{release\_window}(\Gamma, n_{\omega}, n_{\beta})$ 
9   if  $e \leftarrow \text{read\_stream}(S, \tau)$  then
10     $\Gamma \leftarrow \text{insert\_event}(\Gamma, e)$ 
11   $\tau \leftarrow \tau + 1$ 

```

---

### ■ Algorithm 2 add\_slices.

---

```

input :  $\Gamma \in \text{Slices}, \tau \in \mathbb{T}, \omega \in \mathbb{N}, \beta \in \mathbb{N}$ 
1 if  $\omega \bmod \beta > 0$  then
2    $\Gamma \leftarrow \text{add}(0, 0, [\tau, \tau + \omega \bmod \beta])$  to  $\Gamma$ 
3 add  $(0, 0, [\tau + \omega \bmod \beta, \tau + \beta])$  to  $\Gamma$ 

```

---

### ■ Algorithm 3 release\_window.

---

```

input :  $\Gamma \in \text{Slices}, n_{\omega} \in \mathbb{N}, n_{\beta} \in \mathbb{N}$ 
1  $\gamma : (\Phi, \Phi, \mathbb{I}) \leftarrow \Gamma[0]$ 
2 for  $i \in [0, n_{\omega}[$  do
3    $\gamma \leftarrow \text{combine}^*(\gamma, \Gamma[i])$ 
4 delete slice 0 to  $n_{\beta} - 1$  from  $\Gamma$ 
5 print lower*( $\gamma$ )

```

---

### ■ Algorithm 4 insert\_event.

---

```

input :  $\Gamma \in \text{Slices}, e \in S$ 
1  $i : \mathbb{N} \leftarrow |\Gamma| - 1$ 
2  $\gamma : (\Phi, \Phi, \mathbb{I}) \leftarrow \perp$ 
3 while  $P_{\cap}(t(e), t(\Gamma[i])) \wedge i \geq 0$  do
4    $\gamma \leftarrow \text{combine}^*(\gamma, \text{lift}^*(e, t(\Gamma[i])))$ 
5    $i \leftarrow i - 1$ 

```

---

## 5.3 Complexity Analysis

We now compare complexities between the Buckets method and the slicing one. Results are shown in Table 3, for each of the functions given in Algorithms 2, 3 and 4.

The Buckets method allocates one bucket per window. Then, it stores all the  $N$  events intersecting a window to its associated bucket. We here store the events in their original form, without any pre-aggregation as in the Tuple Buckets technique [21]. The events are hence duplicated for each bucket of every non-closed window they are in. The number of such windows depends on the TTP  $\delta$ , and is  $\lceil \delta / \beta \rceil$ . Then, the aggregate is computed from all the  $N$  events in the bucket, only at window release. No sharing can improve complexities in this case.

The slicing technique on the other hand create two slices per step. Initially the cost per window of adding slices is the number of slices per window,  $2 \lceil \omega / \beta \rceil$ . Nonetheless, because the slices are shared among windows, the cost of adding slices is shared too, one slice being used in  $\lceil \omega / \beta \rceil$  windows. This leads to a cost of 2 per window for slice addition.

Event insertion in SES has initially a worst case complexity in  $2 \cdot \mathcal{O}(\lceil \omega / \beta \rceil \cdot N)$  because all slices could receive all events. However, we assume that most of the time, event size is smaller than the window size. Hence the event needs not be inserted to all slices of a window,



and we prefer to consider the average event size  $\mu_e$  to analyze the number of impacted slices. The complexity becomes then  $2 \cdot \mathcal{O}(\lceil \mu_e / \beta \rceil \cdot N)$ . Again, slice sharing allows us to reduce the cost, which becomes  $2 \cdot \mathcal{O}(\lceil \mu_e / \omega \rceil \cdot N)$  with an upper bound in  $2 \cdot \mathcal{O}(N)$ . The best-case complexity is in  $\mathcal{O}(N / \lceil \omega / \beta \rceil)$  when each event is inserted into only one slice. Hence the behavior of event insertion varies depending on the size of the events.

The final aggregate computation represents the main improvement of our slicing technique, with a complexity depending only on window parameters. The complexity of window release is there reduced from the number  $N$  of events to the number  $2 \lceil \omega / \beta \rceil$  of slices per window, i.e., a constant value.

Finally, we can note that, when  $\omega \bmod \beta = 0$ , only one slice is created per step and all complexities are divided by two (see Table 3).

■ **Table 3** Complexity overview (time cost per window), w.r.t.  $N$ , the number of events in a window.

Algorithm	add_slices	release_window	insert_event
Buckets	$\emptyset$	$N$	$N$
SE-Slices ( $\omega \bmod \beta > 0$ )	2	$2 \lceil \omega / \beta \rceil$	$2 \lceil \mu_e / \omega \rceil \cdot N$
SE-Slices ( $\omega \bmod \beta = 0$ )	1	$\lceil \omega / \beta \rceil$	$\lceil \mu_e / \omega \rceil \cdot N$

The space complexity per window is also greatly improved by the slicing technique. Buckets keeps all then events and hence has a space complexity in  $\mathcal{O}(N)$ . In contrast, our slicing technique keeps only one pre-aggregate for each slice, with a complexity in  $\mathcal{O}(\lceil \omega / \beta \rceil)$ , and only in  $\mathcal{O}(1)$  when taking slice sharing into account. Notice that if the size is reduced, it also becomes bounded with the slicing technique.

## 6 Experiments

### 6.1 Experimental Setup

This series of experiment intends to demonstrate the performance improvements with slices compared to buckets. Throughput in these experiments is achieved by letting the program absorb as many events as it can.

**Data Set.** We used two data sets. Firstly, a *generated data set* where each event size is determined by a random number generated with a normal distribution ( $\mu$  is given as average event size parameter,  $\sigma = 10$ ). The system creates a non-delayed stream with one event per chronon, totalling 2M events. Next, the *SS7 data set* replays a real-world telephony network with one minute of anonymized data containing a stream of 3.2M events. Each event contains 119 fields from which we extract the start and stop times to generate event intervals.  $\delta$  represents the TTP.

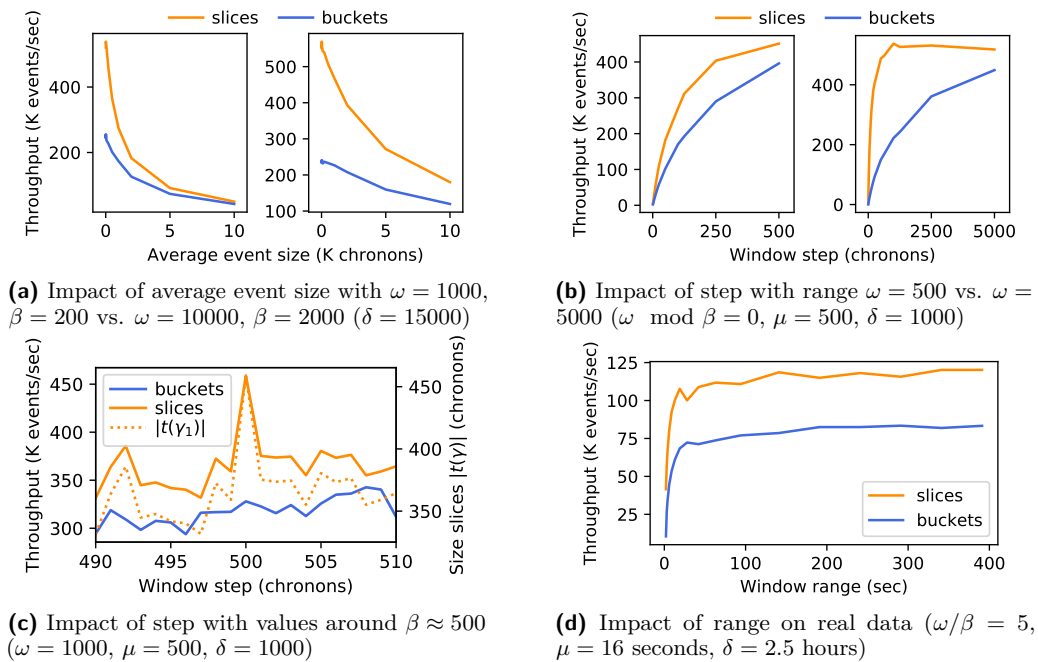
**Aggregates.** For each window we computed three aggregates: two cumulative functions, namely *count* and *sum*, and one selective function, *max*.

**Setup.** All experiments were executed on a single core Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz with 16 GB of RAM under Linux Debian 10.

**Implementation.** Implementation has been coded in modern C++. Algorithms for the slicing method SE-Slices are shown in Section 5.2 for *count* and *sum*. *max* uses a similar algorithm, with a non-duplicated slice structure. For Buckets, we only store event pointers.

## 6.2 Results

As expected from the complexity review, and as illustrated in Figure 6a, event size has an impact on throughput, for both methods, with SE-Slices performing better than Buckets in all cases. The smaller the event, the best SE-Slices performs. SE-Slices shows an increase in performance for all step sizes when  $\omega \bmod \beta = 0$  (see Figure 6b). In particular, a significant improvement appears for smaller steps (as long as they are not too close to one), which shows SE-Slices advantage with overlapping windows. When  $\omega \bmod \beta > 0$ , as in Figure 6c, performance improvement is smaller due to the increase in complexity, but slices still perform better than buckets. With real data, and for all window sizes, SE-Slices performs at least 40% better than Buckets (see Figure 6d). In summary, all the experiments show significant improvement in using the slicing technique compared to the bucket one.



■ **Figure 6** Throughput metrics comparing slices and bucket techniques.

## 7 Conclusion

This article extends the problem of aggregate sharing among overlapping windows to *spanning-event* streams (SES for short). Dealing with spanning events brings new constraints, since events intersect the on-going window as much as past windows. Concerning slicing techniques, dealing with SES implies that adjacent slices can both contain the very same event. Hence, operations sensitive to duplication would provide inaccurate results, and common slicing techniques cannot not be used directly.

Therefore, we extended slicing structures and algorithms depending on the properties of the aggregate functions. When functions are insensitive to event duplicates, we keep the structure and workflow previously used with point events, with a difference however. At event insertion, we update all the intersecting slices instead of only the last one. When functions do have this sensitivity, we duplicate the structure to separate events that ends in the slice from the ones that continue afterwards.

As expected from complexity analyses, slicing techniques with spanning events are computationally more costly than with point events, but they stay, in average, lower than with the buckets technique. Nevertheless, experiments show that the use of slices with spanning events results in significant improvements in throughput. However, when the range is not divisible by the step, performances are only slightly better than with simpler techniques, such as the bucket approach. Hence more advanced techniques need to be experimented in order to circumvent this limiting constraint. Our structure would be of great interest for all techniques which purpose is to partially aggregate spanning events. Eventually, the impact of out-of-order event streams should be studied in our framework.

---

## References

- 1 James F. Allen. Maintaining knowledge about temporal intervals. *Communications of ACM*, 26(11):832–843, 1983.
- 2 Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.
- 3 Arvind Arasu and Jennifer Widom. Resource Sharing in Continuous Sliding-Window Aggregates. *VLDB '04*, 30:336–347, 2004.
- 4 Michael H. Böhlen, Anton Dignös, Johann Gamper, and Christian S. Jensen. Temporal Data Management : An Overview. In *eBISS 2017*, volume 324, pages 51–83, 2017.
- 5 Paris Carbone, Jonas Traub, Asterios Katsifodimos, Seif Haridi, and Volker Markl. Cutty: Aggregate Sharing for User-Defined Windows. In *CIKM '16*, pages 1201–1210, 2016.
- 6 Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data Cube : A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
- 7 Martin Hirzel, Scott Schneider, and Kanat Tangwongsan. Tutorial: Sliding-Window Aggregation Algorithms. In *DEBS '17*, pages 11–14, 2017.
- 8 Hyeon Gyu Kim and Myoung Ho Kim. A review of window query processing for data streams. *Journal of Computing Science and Engineering*, 7(4):220–230, 2013.
- 9 Sailesh Krishnamurthy, Michael J. Franklin, Jeffrey Davis, Daniel Farina, Pasha Golovko, Alan Li, and Neil Thombre. Continuous analytics over discontinuous streams. In *SIGMOD '10*, pages 1081–1092, 2010.
- 10 Sailesh Krishnamurthy, Chung Wu, and Michael Franklin. On-the-fly sharing for streamed aggregation. In *SIGMOD '06*, pages 623–634, 2006.
- 11 Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. No Pane, No Gain: Efficient Evaluation of Sliding-Window Aggregates over Data Streams. *ACM SIGMOD Record*, 34(1):39–44, 2005.
- 12 Jin Li, Kristin Tufte, David Maier, and Vassilis Papadimos. AdaptWID: An Adaptive, Memory-Efficient Window Aggregation Implementation. *IEEE Internet Computing*, 12(6):22–29, 2008.
- 13 Jin Li, Kristin Tufte, Vladislav Shkapenyuk, Vassilis Papadimos, Theodore Johnson, and David Maier. Out-of-order processing: a new architecture for high-performance stream systems. *Proceedings of the VLDB Endowment*, 1(1):274–288, 2008.
- 14 Kostas Patroumpas and Timos Sellis. Window Specification over Data Streams. *EDBT '06*, pages 445–464, 2006.
- 15 Danila Piatov and Sven Helmer. Sweeping-based temporal aggregation. *SSTD 2017: Advances in Spatial and Temporal Databases*, LNCS 10411:125–144, 2017.
- 16 Anatoli U. Shein, Panos K. Chrysanthis, and Alexandros Labrinidis. FlatFIT: Accelerated incremental sliding-window aggregation for real-time analytics. *SSDBM '17*, pages 1–12, 2017.
- 17 Anatoli U. Shein, Panos K. Chrysanthis, and Alexandros Labrinidis. SlickDeque: High Throughput and Low Latency Incremental Sliding-Window Aggregation. *EDBT '18*, pages 397–408, 2018.

## 10:14 Window-Slicing Techniques Extended to Spanning-Event Streams

- 18 Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time. *DEBS '17*, pages 66–77, 2017.
- 19 Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. Optimal and general out-of-order sliding-window aggregation. *Proceedings of the VLDB Endowment*, 12(10):1167–1180, 2019.
- 20 Kanat Tangwongsan, Martin Hirzel, Scott Schneider, and Kun-Lung Wu. General incremental sliding-window aggregation. *Proceedings of the VLDB Endowment*, 8(7):702–713, 2015.
- 21 Jonas Traub, Philipp Grulich, Alejandro Rodríguez Cuéllar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. Efficient Window Aggregation with General Stream Slicing. In *EDBT '19*, pages 97–108, 2019.
- 22 Jonas Traub, Philipp Marian Grulich, Alejandro Rodriguez Cuellar, Sebastian Bress, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. Scotty: Efficient Window Aggregation for Out-of-Order Stream Processing. In *ICDE '18*, pages 1300–1303, 2018.

# Mining Significant Temporal Networks Is Polynomial

Guido Sciavicco 

Department of Mathematics and Computer Science, University of Ferrara, Italy  
guido.sciavicco@unife.it

Matteo Zavatteri 

Department of Computer Science, University of Verona, Italy  
matteo.zavatteri@univr.it

Tiziano Villa 

Department of Computer Science, University of Verona, Italy  
tiziano.villa@univr.it

---

## Abstract

A Conditional Simple Temporal Network with Uncertainty and Decisions (CSTNUD) is a formalism that tackles controllable and uncontrollable durations as well as controllable and uncontrollable choices simultaneously. In the classic top-down model-based engineering approach, a designer builds a CSTNUD to model, validate and execute some temporal plan of interest. Instead, in this paper, we investigate the bottom-up approach by providing a deterministic *polynomial time* algorithm to *mine* a CSTNUD from a set of execution traces (i.e., a log). This paper paves the way for the design of controllable temporal networks mined from traces that also contain information on uncontrollable events.

**2012 ACM Subject Classification** Computing methodologies → Temporal reasoning; Information systems → Data mining; Computing methodologies → Planning and scheduling; Mathematics of computing → Graph algorithms

**Keywords and phrases** Mining temporal constraints, cstnud, uncertainty, significant temporal network

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.11

**Funding** This work was partially supported by MIUR, Project *Italian Outstanding Departments, 2018-2022* and by INdAM, GNCS 2020, Project *Strategic Reasoning and Automated Synthesis of Multi-Agent Systems*.

## 1 Introduction

Temporal networks are a possible framework to model temporal plans and check the coherence of their temporal constraints imposing delays and deadlines between the occurrences of pairs of events in the plan [7]. The main components of a temporal network are *time points* and *constraints*. Time points are real variables modeling temporal events. Executing time points means to assign them real values to fix “when” the corresponding temporal events occurred. Constraints are linear inequalities imposing minimal and maximal temporal distances between pairs of time points.

Over the years the core formalism of *Simple Temporal Networks* [7] has been extended in several ways to cope with uncontrollable durations [17], uncontrollable and controllable choices [5, 13] and, more recently, with combinations of them (see, e.g., [3, 11, 12, 18, 19]). The most expressive formalisms of temporal networks are those that simultaneously handle all such features. Moreover, such formalisms give rise to several, different, taxonomies in which sub-formalisms belonging to them can be ordered by expressive power. As a result, solving any problem for a top-level formalism (e.g., checking consistency or controllability)



© Guido Sciavicco, Matteo Zavatteri, and Tiziano Villa;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 11; pp. 11:1–11:12

Leibniz International Proceedings in Informatics



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

## 11:2 Mining Significant Temporal Networks Is Polynomial

results in solving the same problem for every sub-formalism in the corresponding hierarchy. *Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNUDs, [19, 22])* is a recent formalism tackling controllable and uncontrollable durations as well as controllable and uncontrollable choices simultaneously. CSTNUDs define a hierarchy of temporal networks in which any combination of features can be considered by focusing on the corresponding sub-formalism.

Like any model-based engineering approach, creating a temporal network is a complex, time-consuming, and error-prone task, where typically discrepancies between the actual process and the obtained network might eventually emerge asking the designer for refinement or abstraction of the model being created. This is a top-down, trial-and-error approach. Instead, the opposite, bottom-up, approach is known in the literature under the name of *process mining* and it aims to *mine* (i.e., synthesize) process descriptions (or, more reasonably, model approximations) from execution traces (i.e., process logs).

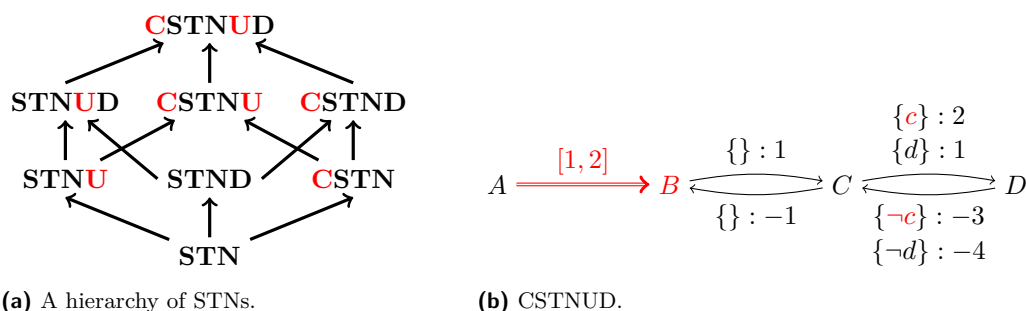
A *trace* formalizes a run of a process, and the set of all available traces can be thought of as a *log* of a process carried out many times by humans that base their actions on their experience only. As a result, since such actions may not follow any particular rule, we have no guarantee of consistency or controllability of the underlying process overall. One of the first contributions in process mining is that of Agrawal, Gunopulos, and Leymann [1], but, after this seminal work, many others have come by focusing on different process description languages [6, 8, 14, 15, 16]. However, to the best of our knowledge, the problem of mining temporal networks subject to uncontrollable parts has not received particular attention. Despite the current trend in process mining calls for machine learning techniques, we shall see that the well-founded mathematical structure of temporal networks allows us to solve this problem correctly and efficiently, because of a strong underlying monotonicity.

**Contribution.** We provide a deterministic polynomial time algorithm to mine a CSTNUD from a finite set of execution traces. By construction, every trace in the set will satisfy the constraints of the mined CSTNUD, therefore the CSTNUD is correct, complete and significant, meaning that every temporal event, (un)controllable duration and (un)controllable choice belonging to some processed trace occurs in it.

**Organization.** Section 2 discusses background and related work. Section 3 adapts CSTNUDs for the purpose of this paper. Section 4 defines the problem of mining significant CSTNUDs and provides a correct algorithm for it. In Section 5 we conclude by summing up and discussing future work.

## 2 Background and Related Work

*Simple Temporal Networks (STNs)* [7] model fully controllable and non-disjunctive temporal plans but they cannot deal with (un)controllable choices nor with uncontrollable durations. To bridge such gaps some extensions were put forth over the years. *Simple Temporal Networks with Uncertainty (STNUs)* [17] extend STNs with uncontrollable (but bounded) durations by means of contingent links. A contingent link consists of an activation (time) point, whose execution is under control, a contingent (time) point, whose execution is not, and a closed interval specifying the minimal and maximal duration of the link. *Conditional Simple Temporal Networks (CSTNs)* [10] (formerly, *Conditional Temporal Problem (CTP)* [13]) extend STNs with uncontrollable choices. Constraints are *labeled* by sets of consistent literals over a finite set of uncontrollable Boolean variables, or *booleans*, which describe when the



■ **Figure 1** Hierarchy (left) and an example of CSTNU D (right). STNDs, STNU Ds and CSTNDs are further frameworks implicitly arising from CSTNU Ds. Any acronym speaks for what the corresponding framework supports: “C” (uncontrollable choices), “D” (controllable choices), “U” (contingent links).

components labeled by them are relevant for an execution. The truth value assignments to these booleans are out of control and take place upon the execution of specific time points. Initially, labels were on both time points and constraints but later it was proved that having labels on constraints only does not limit the expressiveness of the network [2]. Temporal networks with labels on constraints only are called *streamlined Conditional Simple Temporal Networks with Uncertainty (CSTNUs)* [9] merge STNUs and CTPs/CSTNs, whereas *Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNU Ds)* [19, 22] extend CSTNUs with controllable choices (i.e., controllable booleans). Figure 1a shows the hierarchy of CSTNU Ds. Figure 1b gives an example of CSTNU D that we discuss in Section 3. Other formalisms were built on top of STNs (e.g., [3, 5, 11, 12, 18]) but are not employed in this work.

*Process mining* has been approached by several authors. Agrawal, Gunopulos and Leymann first introduced the problem of producing a process description from unstructured executions in a log [1]. Cook and Wolf investigated similar issues in the context of software engineering processes [6]. They described three methods for process discovery: one based on neural networks, one based on a purely algorithmic approach, and one based on a Markovian approach. In particular, in the second approach, they built a finite state machine where states are fused whenever their futures (in terms of possible behavior in the next  $k$  steps) are identical. In [8], Herbst addresses the issue of process mining in the context of workflow management using an inductive (machine learning based) approach. Finally, in [14], Van Der Aalst proposes an algorithm to extract a workflow network from logs of a hospital. The results of the experiments highlight that the proposed method can discover processes whose underlying models are acyclic and sound workflow nets, involving parallel, conditional and sequential workflow blocks.

### 3 Conditional STNUs with Decisions

When uncontrollable parts are supported, the corresponding planning and scheduling problem modeled by the underlying network can be seen as a two-player game between Controller (representing the executor) and Nature (representing the environment). Controller executes controllable time points and assigns truth values to controllable booleans. Nature does the same for uncontrollable time points and uncontrollable booleans. Controller aims to satisfy all constraints. Nature aims to have Controller violate at least one of them. In other words, we are the Controller and everything else is Nature.



► **Definition 1.** A Conditional Simple Temporal Network with Uncertainty and Decisions (CSTNUd) is a tuple  $\langle \mathcal{T}, \mathcal{B}, \mathcal{T}_B, \beta, \mathcal{L}, \mathcal{C} \rangle$ , where:

- $\mathcal{T} = \mathcal{T}_C \dot{\cup} \mathcal{T}_U = \{A, \dots, Z\}$  is a finite set of time points disjointly partitioned in controllable time points (those executed by Controller) and uncontrollable time points (those executed by Nature), respectively.
- $\mathcal{B} = \mathcal{B}_C \dot{\cup} \mathcal{B}_U = \{a, \dots, z\}$  is a finite set of booleans disjointly partitioned in controllable booleans (those assigned by Controller) and uncontrollable booleans (those assigned by Nature), respectively.
- $\mathcal{T}_B \subseteq \mathcal{T}_C$  is the set of controllable time points having booleans associated according to  $\beta$ .
- $\beta : \mathcal{T}_B \rightarrow \mathcal{B}$  is a bijection assigning to any  $A \in \mathcal{T}_B$  the boolean  $\beta(A)$ . Once a time point  $A \in \mathcal{T}_B$  is executed, the truth value of  $\beta(A)$  is set by Controller (if  $\beta(A) \in \mathcal{B}_C$ ) or by Nature (if  $\beta(A) \in \mathcal{B}_U$ ).
- $\mathcal{L}$  is a set of contingent links each having the form  $(A, \ell, u, B)$  where  $A \in \mathcal{T}_C$ ,  $B \in \mathcal{T}_U$ ,  $\ell, u \in \mathbb{R}$  with  $0 < \ell \leq u$ . Once  $A$  is executed by Controller,  $B$  is executed by Nature guaranteeing that the temporal distance between  $A$  and  $B$  falls in  $[\ell, u]$ . Contingent links do not share uncontrollable time points.
- $\mathcal{C}$  is a set of temporal constraints having the form  $\mathcal{S} : B - A \leq k$ , where  $\mathcal{S}$ , the label of the constraint, is a consistent set of literals over  $\mathcal{B}$ ,  $B, A \in \mathcal{T}$  and  $k \in \mathbb{R}$ . Many temporal constraints  $\mathcal{S}_i : B - A \leq k_i$  may be defined for the same pair of time points (w.r.t. the same direction<sup>1</sup>) provided  $\mathcal{S}_i$  is different. Any pair  $\mathcal{S}_1 : B - A \leq k_1$  and  $\mathcal{S}_2 : B - A \leq k_2$  with  $\mathcal{S}_1 = \mathcal{S}_2$ , implies  $\mathcal{S}_1 : B - A \leq \min\{k_1, k_2\}$  (tightening). Temporal constraints labeled by  $\mathcal{S} = \emptyset$  are unconditional (i.e., they must always hold). Those labeled by  $\mathcal{S} \neq \emptyset$  are conditional: they hold only if all literals in  $\mathcal{S}$  are satisfied by the truth value assignment to the booleans.

Definition 1 differs from that given in [19] as follows. First, our CSTNUdS are streamlined. Second, we allow the intervals of contingent links to be a single point; despite this resembles no uncertainty, it is an extension that does not break the current semantics (we just know what Nature will do in that case). Third, we no longer differentiate between observation and decision time points<sup>2</sup> but we just focus on controllable and uncontrollable booleans.

We graphically represent a CSTNUd as a directed graph whose set of nodes coincides with the set of time points and whose set of edges divides in double and single edges. A double edge  $A \Rightarrow B$  labeled by  $[\ell, u]$  models a contingent link  $(A, \ell, u, B)$ . A single edge  $A \rightarrow B$  labeled by  $\mathcal{S} : k$  models a temporal constraint  $\mathcal{S} : B - A \leq k$ . Figure 1b shows an example of CSTNUd where we highlight uncontrollable parts in red. This network contains three controllable time points  $A, C, D$ , one uncontrollable time point  $B$ , one contingent link  $(A, 1, 2, B)$ , a controllable boolean  $d$  associated to  $D$ , an uncontrollable boolean  $c$  associated to  $C$ , two unconditional temporal constraint  $\emptyset : C - B \leq 1$  and  $\emptyset : B - C \leq -1$  and four conditional ones  $\{-c\} : C - D \leq -3$ ,  $\{-d\} : C - D \leq -4$ ,  $\{c\} : D - C \leq 2$  and  $\{d\} : D - C \leq 1$ . A few problems are associated to CSTNUdS. For example, when  $\mathcal{B}_U$  and  $\mathcal{T}_U$  are both empty, the network does not have uncontrollable parts; in this case, we may ask whether the network is *consistent*. On the other hand, when at least one among  $\mathcal{B}_U$  and  $\mathcal{T}_U$  is nonempty, then the network has at least one uncontrollable part, and consistency is no longer a well-defined problem. In this case, we worry about *controllability*, that is,

<sup>1</sup> Regardless of  $\mathcal{S}$  and  $k$ , a constraint on a pair of time points  $A, B$  has two possible “directions”:  $\mathcal{S} : B - A \leq k$  and  $\mathcal{S} : A - B \leq k$ .

<sup>2</sup> Historically, time points associated to controllable (resp., uncontrollable) booleans were called decisions (resp., observations).



if we can find an execution strategy for the CSTNUD, according to different assumptions on Nature's behavior. Given a CSTNUD  $\mathcal{Z} = \langle \mathcal{T}, \mathcal{B}, \mathcal{T}_B, \beta, \mathcal{L}, \mathcal{C} \rangle$ , we say that a (partial) mapping  $t: \mathcal{T} \rightarrow \mathbb{R}$  assigning real values to the time points is a *schedule* if it enforces that for each  $(A, \ell, u, B) \in \mathcal{L}$ , if  $B \in \text{dom}(t)$ , then  $A \in \text{dom}(t)$  and  $t(B) \in [t(A) + \ell, t(A) + u]$ . Furthermore, let  $\mathcal{P}$  be a consistent set of literals over  $\mathcal{B}$ , that is, a (partial) instantiation of truth values to the booleans of a network. We call  $\langle \mathcal{P}, t \rangle$  a *model*.  $\langle \mathcal{P}, t \rangle$  is total iff  $\text{dom}(t) = \mathcal{T}$  and for each  $a \in \mathcal{B}$  either  $a$  or  $\neg a$  is in  $\mathcal{P}$ .

► **Definition 2.** Let  $\mathcal{Z} = \langle \mathcal{T}, \mathcal{B}, \mathcal{T}_B, \beta, \mathcal{L}, \mathcal{C} \rangle$  be a CSTNUD. We say that the model  $\langle \mathcal{P}, t \rangle$  satisfies a network  $\mathcal{Z}$  (in symbols,  $\langle \mathcal{P}, t \rangle \models \mathcal{Z}$ ) if and only if for each  $\mathcal{S} : B - A \leq k \in \mathcal{C}$ , whenever  $\mathcal{S} \subseteq \mathcal{P}$  and  $A, B \in \text{dom}(t)$ , then  $t(B) - t(A) \leq k$ .

► **Definition 3.** Let  $\mathcal{Z} = \langle \mathcal{T}, \mathcal{B}, \mathcal{T}_B, \beta, \mathcal{L}, \mathcal{C} \rangle$  be a CSTNUD. We say that:

- $\mathcal{Z}$  is weakly controllable if whenever Nature tells Controller (before starting the execution) what durations and truth values she is going to assign to contingent links and uncontrollable booleans, Controller can generate a schedule and a consistent set of literals that contain the information given by Nature and satisfy all constraints.
- $\mathcal{Z}$  is strongly controllable if Controller can find a unique schedule for controllable time points and a unique consistent set of literals over controllable booleans that will satisfy all constraints regardless of any possible extension that Nature can provide for uncontrollable time points and uncontrollable booleans.
- $\mathcal{Z}$  is dynamically controllable if Controller can dynamically generate a schedule over controllable time points and a consistent set of literals over controllable booleans depending on which extension Nature is providing for uncontrollable time points and uncontrollable booleans.

The CSTNUD shown in Figure 1b is weakly, dynamically but not strongly controllable. We provide an example of dynamic execution strategy. Controller executes  $A$  at 0. Then, Nature executes  $B$  at a time falling in  $[1, 2]$ . After that, Controller executes  $C$  exactly 1 after  $B$  and Nature chooses a truth value for  $c$ . If  $c$  is true, then Controller executes  $D$  within 1 after  $C$  and assigns true to  $d$ . If  $c$  is false, then Controller executes  $D$  after 4 since  $C$  and assigns false to  $d$ .

## 4 Mining Significant CSTNUDs

As we explained in Section 2, a CSTNUD models a temporal plan in a compact way. The problems that are associated with a network allow one to study intrinsic properties of the network itself. In real-world cases, often the network is not given, but, on the contrary, it must be hand-craftily designed. In this section, we approach this problem: given a finite set of execution traces (e.g., a *log*) of the underlying temporal plan, we solve the problem of automatically mine a “good” CSTNUD that describes it.

► **Definition 4.** A trace  $\tau$  is a sequence of these statements:

- $A = t_A$ , where  $t_A \in \mathbb{R}_{\geq 0}$ . This statement models the execution of a controllable time point  $A$  at time  $t_A$ .
- $B(A) = t_B$ , where  $t_B \in \mathbb{R}_{\geq 0}$ . This statement models the execution of an uncontrollable time point  $B$  at time  $t_B$  whose corresponding activation is  $A$  (contingent link).
- $a!$  (resp.,  $\neg a!$ ). This statement models that the controllable boolean  $a$  was assigned true (resp., false).
- $a?$  (resp.,  $\neg a?$ ). This statement models that the uncontrollable boolean  $a$  was assigned true (resp., false).

Controllable and uncontrollable booleans are identified by the suffixes  $!$  and  $?$ , respectively.

## 11:6 Mining Significant Temporal Networks Is Polynomial

The following are example of traces:

$\tau_1$ :  $Z = 0, A = 0, \neg a?$

$\tau_2$ :  $Z = 0, A = 0, \neg a?, E(A) = 5$

$\tau_3$ :  $Z = 0, A = 0, a?, B = 2, b!$

$\tau_4$ :  $Z = 0, A = 0, a?, C = 1, B = 2, \neg b!, D = 4$

$\tau_5$ :  $Z = 0, B = 0, b!, C = 2, E(A) = 2, A = 3, a?, D = 4$

$\tau_6$ :  $Z = 0, A = 0, a?, B = 1, b!, C = 4, D = 6, E(A) = 7$

$\tau_7$ :  $Z = 0, A = 0, a?, D = 5, B = 5, \neg b!$

► **Definition 5.** A trace  $\tau$  is well-defined if it is finite, starts with  $Z = 0$ , and:

- Any time point (resp., any boolean) appearing in  $\tau$  is assigned a real value (resp., truth value) exactly once.
- If  $B(A) = t_B$  appears in  $\tau$ , then  $A = t_A$  appears in  $\tau$  before  $B(A) = t_B$ .
- If any of  $a!, \neg a!, a?, \neg a?$  appears in  $\tau$ , then the statement appearing immediately before it is  $A = t_A$ , meaning that  $\beta(A) = a$ , whereas that appearing immediately after it (if any) is either  $A' = t_{A'}$  or  $A'(A'') = t_{A'}$ .
- If a statement  $B = t_B$  (or  $B(B') = t_B$ ) appears after a statement  $A = t_A$  (or  $A(A') = t_A$ ), then  $t_B \geq t_A$ .

► **Definition 6.** A pair of traces is coherent if and only if:

- Any time point appearing in both traces is of the same type, and, if such a time point is uncontrollable, then it also refers to the same activation.
- Any boolean appearing in both traces is of the same type, and it is associated to the same time point.

A set of traces is coherent if every pair of traces in it is coherent.

► **Definition 7.** A CSTNUD  $\mathcal{Z}$  is significant for a well-defined trace  $\tau$  if the following conditions hold.

- If  $A = t_A \in \tau$ , then  $A \in \mathcal{T}_C$
- If  $B(A) = t_B \in \tau$ , then  $B \in \mathcal{T}_U$  and  $(A, \ell, u, B) \in \mathcal{L}$  for some  $\ell, u \in \mathbb{R}$ ,  $0 \leq \ell \leq u$  such that  $t_B - t_A \in [\ell, u]$ .
- If  $a! = \top \in \tau$  or  $a! = \perp \in \tau$  (resp.,  $a? = \top \in \tau$  or  $a? = \perp \in \tau$ ), then  $a \in \mathcal{B}_C$  (resp.,  $a \in \mathcal{B}_U$ ) and  $\beta(A) = a$ ; moreover, if  $A = t_A$  is the statement before it, then  $\beta(A) = a$ .
- $\langle \mathcal{P}, t \rangle \models \mathcal{Z}$ , where  $\mathcal{P}$  and  $t$  are the consistent set of literals and schedule arising from  $\tau$ , respectively.

The set  $\mathcal{I} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7\}$  in the example above is coherent and contains well-defined traces.

**Problem.** Given a finite set of well-defined and coherent traces, mine a significant CSTNUD.

**CstnudMiner** (Algorithm 1) starts by creating a CSTNUD containing the zero-time point  $Z$  only. After processing a trace,  $\mathcal{Z}$  contains all time points, booleans, contingent links and temporal constraints specified by that trace. After processing a set of traces,  $\mathcal{Z}$  is such that all traces in that set satisfy it, and once a trace is processed, it can be forgotten. **CstnudMiner** internally uses the rules **WeakenTC** and **WeakenCL** on temporal constraints and contingent links, respectively. Table 1 shows these weakening rules. The aim of these two sub-procedures is to add temporal constraints and contingent links if they do not exist in  $\mathcal{Z}$  or to *weaken* them otherwise (to allow new traces to satisfy  $\mathcal{Z}$  as well). Before proceeding we introduce some useful notation. Given a pair of time point  $A, B$  and a set of literals  $\mathcal{S}$ , let  $L(B, A) = \{\mathcal{S} \mid \mathcal{S} : B - A \leq k \in \mathcal{C}\}$  be the set of labels of all temporal constraints going from  $A$  to  $B$  and let:

■ **Table 1** Weakening rules for Algorithm 1.

WeakenTC( $S : B - A \leq k$ )	
<b>Case T1:</b> $L_1(\mathcal{S}, B, A) = \{\mathcal{S}_1, \dots, \mathcal{S}_n\} \neq \emptyset$ .	
$\begin{array}{c} \dots \\ \mathcal{S}_1 : k_1 \dots \mathcal{S}_n : k_n \\ A \xrightarrow{\dots} B \end{array}$	$\begin{array}{c} \dots \\ \mathcal{S}_1 : \max\{k_1, k\} \dots \mathcal{S}_n : \max\{k_n, k\} \\ A \xrightarrow{\dots} B \end{array}$
<b>Example:</b> WeakenTC( $\{a, b\} : C - Z \leq 7$ )	
$\begin{array}{c} \{-c, \neg a\} : 6 \quad \{b\} : 8 \\ Z \xrightarrow{\{a\} : 3} C \end{array}$	$\begin{array}{c} \{-c, \neg a\} : 6 \quad \{b\} : 8 \\ Z \xrightarrow{\{a\} : 7} C \end{array}$
<b>Case T2:</b> $L_2(\mathcal{S}, B, A) = \{\mathcal{S}_1, \dots, \mathcal{S}_n\} \neq \emptyset$ .	
$\begin{array}{c} \dots \\ \mathcal{S}_1 : k_1 \dots \mathcal{S}_n : k_n \\ A \xrightarrow{\dots} B \end{array}$	$\begin{array}{c} \dots \\ S : \max\{K_2(\mathcal{S}, B, A), k\} \\ A \xrightarrow{\dots} B \end{array}$
<b>Example:</b> WeakenTC( $\{b\} : Z - C \leq -7$ )	
$\begin{array}{c} \{-c, b\} : -8 \quad \{-b, \neg a\} : -6 \\ Z \xleftarrow{\{a, b\} : -3} C \end{array}$	$\begin{array}{c} \{-b, \neg a\} : -6 \\ Z \xleftarrow{\{b\} : -3} C \end{array}$
<b>Case T3:</b> $L_1(\mathcal{S}, B, A) = L_2(\mathcal{S}, B, A) = \emptyset$ .	
$A \xrightarrow{\dots} B$	$A \xrightarrow{\dots S : k \dots} B$
<b>Example:</b> WeakenTC( $\{b\} : C - Z \leq 5$ )	
$\begin{array}{c} \{a, \neg b\} : 8 \\ Z \xrightarrow{\{-b\} : 3} C \end{array}$	$\begin{array}{c} \{a, \neg b\} : 8 \quad \{b\} : 5 \\ Z \xrightarrow{\{-b\} : 3} C \end{array}$
WeakenCL( $A, k, k, B$ )	
<b>Case L1:</b> contingent link between $A$ and $B$ exists	
$A \xrightarrow{[\ell, u]} B$	$A \xrightarrow{[\min\{\ell, k\}, \max\{u, k\}]} B$
<b>Case L2:</b> no contingent link between $A$ and $B$	
$A \quad B$	$A \xrightarrow{[k, k]} B$

- $L_1(\mathcal{S}, B, A) = \{\mathcal{S}_i \mid \mathcal{S}_i \in L(B, A), \mathcal{S}_i \subseteq \mathcal{S}\}$  be the set of labels in  $L(B, A)$  contained in  $\mathcal{S}$ .
- $L_2(\mathcal{S}, B, A) = \{\mathcal{S}_i \mid \mathcal{S}_i \in L(B, A), \mathcal{S} \subset \mathcal{S}_i\}$  be the set of labels in  $L(B, A)$  strictly containing  $\mathcal{S}$ .
- $L_3(\mathcal{S}, B, A) = L(B, A) \setminus (L_1 \cup L_2)$  be the set of all other labels in  $L(B, A)$  neither in  $L_1$  nor in  $L_2$ .
- $K_i(\mathcal{S}, B, A) = \{k_j \mid \mathcal{S}_j : B - A \leq k_j \in \mathcal{C}, \mathcal{S}_j \in L_i(\mathcal{S}, B, A)\}$  be the set of weights of all constraints from  $A$  to  $B$  labeled by  $\mathcal{S}_j \in L_i(\mathcal{S}, B, A)$  for  $i = 1, 2, 3$ .

---

**Algorithm 1** CstnudMiner.

---

**Input:** A set  $\mathcal{I}$  of well-defined and coherent traces.  
**Output:** A significant CSTNUD.

```

1  $\mathcal{Z} = \langle \{Z\}, \emptyset, \emptyset, \beta, \emptyset, \emptyset \rangle$  ▷ “Initial CSTNUD”
2 foreach  $\tau \in \mathcal{I}$  do
3    $\mathcal{S} \leftarrow \emptyset$ 
4   foreach statement in  $\tau$  do
5     if the statement is  $A = t_A$  then
6        $\mathcal{T}_C \leftarrow \mathcal{T}_C \cup \{A\}$ 
7       WeakenTC( $\mathcal{S} : A - Z \leq t_A$ )
8       WeakenTC( $\mathcal{S} : Z - A \leq -t_A$ )
9     if the statement is  $B(A) = t_B$  then
10       $\mathcal{T}_U \leftarrow \mathcal{T}_U \cup \{B\}$ 
11      Let  $t_A$  be the execution time of  $A$ .
12      WeakenCL( $A, t_B - t_A, t_B - t_A, B$ )
13    if the statement is  $a!$  or  $\neg a!$  or  $a?$  or  $\neg a?$  then
14      Let  $A = t_A$  be the previous statement.
15      if the suffix is  $!$  then  $\mathcal{B}_C \leftarrow \mathcal{B}_C \cup \{a\}$ ;
16      else  $\mathcal{B}_U \leftarrow \mathcal{B}_U \cup \{a\}$ ;
17       $\beta(A) = a$ 
18      if the prefix is  $\neg$  then  $\mathcal{S} \leftarrow \mathcal{S} \cup \{\neg a\}$ ;
19      else  $\mathcal{S} \leftarrow \mathcal{S} \cup \{a\}$ ;
20 return  $\mathcal{Z}$ 

```

---

**WeakenTC** works on a temporal constraint  $\mathcal{S} : B - A \leq k$ . When called, **WeakenTC** first computes  $L_1(\mathcal{S}, B, A)$ ,  $L_2(\mathcal{S}, B, A)$  and  $L_3(\mathcal{S}, B, A)$ . Then, it handles three cases as follows:

**Case T1.** This case applies whenever  $L_1(\mathcal{S}, B, A) \neq \emptyset$ . In such a case, each weight  $k_i$  associated to a constraint going from  $A$  to  $B$  labeled by any  $\mathcal{S}_i \in L_1(\mathcal{S}, B, A)$  is possibly weakened (meaning raised) to  $k$  if  $k > k_i$ .

**Case T2.** This case applies whenever  $L_2(\mathcal{S}, B, A) \neq \emptyset$ . In such a case, we add a single constraint labeled by  $\mathcal{S}$  whose numeric weight is the maximum value in the set  $K_2(\mathcal{S}, B, A) \cup \{k\}$ . Finally, we remove each constraint labeled by any  $\mathcal{S}_i \in L_2(\mathcal{S}, B, A)$ .

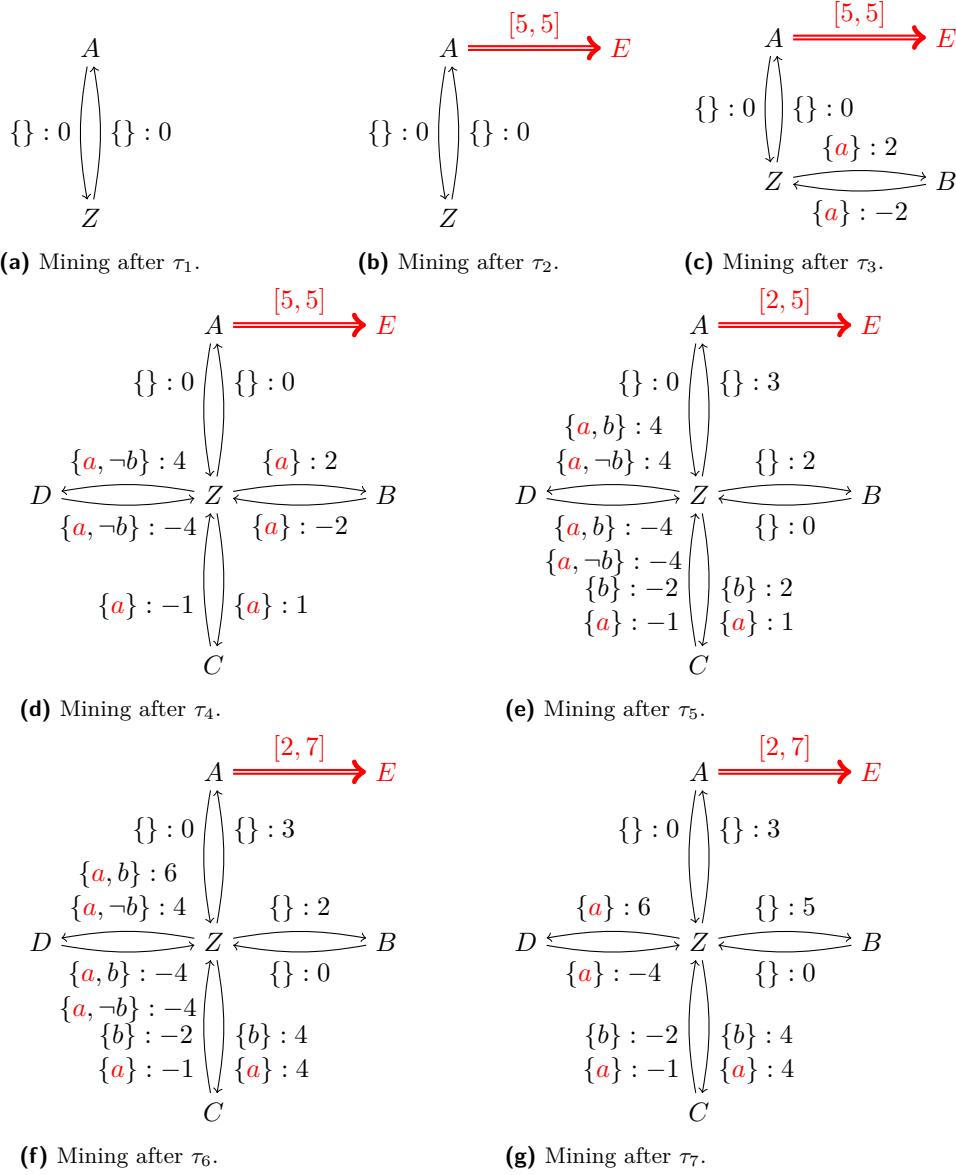
**Case T3.** This case applies whenever  $L_1(\mathcal{S}, B, A) = L_2(\mathcal{S}, B, A) = \emptyset$  (i.e., whenever neither Case T1 nor Case T2 does). In such a case we just add  $\mathcal{S} : A - B \leq k$ .

**WeakenCL**, on the other hand, works on a contingent link  $(A, k, k, B)$  by means of two mutually-exclusive cases:

**Case L1.** This case applies whenever  $(A, \ell, u, B)$  already exists. In such a case, we weaken (meaning lower)  $\ell$  to  $k$  if  $k < \ell$  or weaken (meaning raise)  $u$  to  $k$  if  $u < k$  (note that at most one among  $\ell$  and  $u$  is weakened).

**Case L2.** This case applies whenever  $(A, \ell, u, B)$  does not exist. In such a case, we add  $(A, k, k, B)$ .

We provide application examples for **WeakenTC** in Table 1. We omit those for **WeakenCL** due to their triviality. Figure 2 shows the result of applying **CstnudMiner** on the previous discussed set of traces  $\mathcal{I} = \{\tau_1, \dots, \tau_7\}$ .



■ **Figure 2** Mining a significant CSTNUD from execution traces.

We are left to discuss invariants, correctness and complexity of **CstnudMiner**. Let  $\mathcal{Z}$  be the CSTNUD being mined.

► **Invariant 1.** *Cases T1, T2 and T3 of **WeakenTC** are mutually-exclusive. So are cases L1 and L2 of **WeakenCL**.*

**Proof.** We only need to focus on **WeakenTC**, as cases L1 and L2 of **WeakenCL** are mutually-exclusive by definition (either a contingent link exists or it doesn't). When **CstnudMiner** starts, the invariant is trivially true as no constraints exist (so only case T3 is enable). Now assume that Invariant 1 holds and let  $\mathcal{S}$  be the set of literals collected in the current trace  $\tau$  being processed. Moreover, assume the current processed statement is  $A = t_A$ . If Case T1 applies, only the weights of the constraints labeled by some  $\mathcal{S}_i \in L_1(\mathcal{S}, B, A)$

are (possibly) modified. After processing the statement,  $L(B, A)$  remains the same, thus Invariant 1 still holds. If Case T2 applies, all constraints labeled by some  $\mathcal{S}_i \in L_2(\mathcal{S}, B, A)$  are thrown away and replaced by a single constraint labeled by  $\mathcal{S}$ . After processing the statement, (the new)  $L(B, A)$  contains  $\mathcal{S}$  but does not contain any  $\mathcal{S}_i$  in the previous  $L_2(\mathcal{S}, B, A)$ . Note that  $\mathcal{S}$  is not a superset of any other  $\mathcal{S}_i \in L(B, A)$ ,  $\mathcal{S}_i \neq \mathcal{S}$  as if it were, so would each set in  $L_2(\mathcal{S}, B, A)$  before processing the statement (invariant contradiction). If Case T3 applies, a constraint labeled by  $\mathcal{S}$  is merely added. After that, the new  $L(B, A)$  contains  $\mathcal{S}$ , and Invariant 1 still holds. ◀

► **Theorem 8.** *CstnudMiner mines a significant CSTNUD.*

**Proof.** We need to prove the following: first, that **WeakenTC** and **WeakenCL** are *sound* meaning that any constraint and uncontrollable duration for contingent links that held before applying the rules keeps holding after applying them, and, second, that **CstnudMiner** mines the same CSTNUD regardless of the ordering in which traces are processed.

**Soundness of the rules.** From Invariant 1 we know that all cases in Table 1 are mutually exclusive. Therefore, we analyze each case of each rule in isolation. Consider a call to **WeakenTC**( $\mathcal{S} : B - A \leq k$ ). In Case T1 only the numeric weights of all constraints  $\mathcal{S}_i : B - A \leq k_i$  where  $\mathcal{S}_i \in L_1(\mathcal{S}, B, A)$  are (possibly) modified. For each  $\mathcal{S}_i \in L_1(\mathcal{S}, B, A)$ , let  $k_i$  be the weight of the constraint  $\mathcal{S}_i : B - A \leq k_i$ . After the rule applies we have that the new weight is  $\max\{k_i, k\}$ . It is clear that the initial constraint still holds as  $\mathcal{S}_i \subseteq \mathcal{S} : B - A \leq k_i \leq \max\{k_i, k\}$ . All remaining constraints in the CSTNUD hold as well as they are left untouched. In Case T2 all constraints labeled by some some set in  $L_2(\mathcal{S}, B, A)$  are thrown away and replaced by a single constraint labeled by  $\mathcal{S}$ . After the rule applies we have that  $\mathcal{S} : B - A \leq \max(K_2(\mathcal{S}, B, A) \cup \{k\})$  is added. Each removed constraint  $\mathcal{S}_i : B - A \leq k_i$  still holds as  $\mathcal{S} \subseteq \mathcal{S}_i : B - A \leq k_i \leq \max(K_2(\mathcal{S}, B, A) \cup \{k\})$ . Once again, all remaining constraints in the CSTNUD hold as well as they are left untouched. In Case T3 no existing constraint is modified. Now, consider a call to **WeakenCL**( $A, k, k, B$ ). We only need to focus on the part of the durations related to this link as all other parts of such durations are left untouched. In Case L1 a contingent link  $(A, \ell, u, B)$  already exists.

After applying the rule, either  $\ell$  is lowered to  $k$  (if  $k < \ell$ ) or  $u$  is raised to  $k$  (if  $u < k$ ) or both are left untouched (if  $\ell \leq k \leq u$ ). Let  $\ell'$  and  $u'$  be the new minimal and maximal durations after the rule is applied. It is clear that  $\ell' \leq \ell \leq u \leq u'$ , therefore all previous durations are still possible. In Case L2 the contingent link does not exist yet, therefore we have nothing to verify.

**Processing order of traces.** **CstnudMiner** processes each trace once. Processing a trace means to process each statement in it. Whenever a time point does not exist, the algorithm adds it to the CSTNUD associating it to the right activation if the time point is uncontrollable. Likewise, whenever a boolean does not exist, the algorithm adds it to the CSTNUD and sets  $\beta$  accordingly. Also, the significance of the resulting CSTNUD (constraints and contingent links) follows from the soundness of the rules. ◀

We are left to discuss the complexity of **CstnudMiner**. Let  $\mathcal{I}$  be the set of well-defined and coherent traces in input.

► **Theorem 9.** *CstnudMiner runs in polynomial time.*

**Proof.** We focus on **WeakenTC** since the operations carried out by **WeakenCL** are negligible (no partitioning of contingent links is done). Instead, **WeakenTC** hides internally inner cycles to compute  $L_1(\mathcal{S}, B, A)$ ,  $L_2(\mathcal{S}, B, A)$  and  $L_3(\mathcal{S}, B, A)$  every time that it is called. The worst case happens when **CstnudMiner** applies **WeakenTC** as much as possible as

$L_3(\mathcal{S}, B, A)$  keeps growing. We show how to build a set of traces  $\mathcal{I}$  that leads to this situation. Let  $\mathcal{B}$  be a finite set of booleans and  $\mathcal{T} = \{Z, X\} \cup \{Y_b \mid b \in \mathcal{B}\}$  a finite set of time points. In this way every boolean in  $b$  can be associated to a time point in  $\mathcal{T} \setminus \{Z, X\}$ . We shall see that it is enough to have  $|\mathcal{I}| \leq 2^{|\mathcal{B}|}$ . In our construction, if  $\mathcal{B} = \{a_1, \dots, a_{|\mathcal{B}|}\}$ , then any trace has the form  $Z = 0, Y_{a_1} = 0, a = \diamond, \dots, Y_{a_{|\mathcal{B}|}} = 0, a_{|\mathcal{B}|} = \diamond, X = 0$  where  $\diamond \in \{\top, \perp\}$  with trace specifying a different set of literals  $\mathcal{S}$ .

In this way, each trace  $\tau \in \mathcal{I}$  has length  $|\tau| = 2 + 2 \times |\mathcal{T} \setminus \{Z, X\}|$ . Each trace specifies a different truth value assignment for the booleans in  $\mathcal{B}$ . When the last statement of each trace is processed, **CstnudMiner** can only add a new constraint between  $Z$  and  $X$  (for each direction). As a result, the maximum number of constraints that appear between  $Z$  and  $X$  (in any direction) is  $|\mathcal{I}|$ . However, when the “ $X = 0$ ” of the  $|\mathcal{I}|^{\text{th}}$  trace is processed, we know (from the proof of Theorem 8) that **WeakenTC**( $\mathcal{S} : X - Z \leq 0$ ) (resp., **WeakenTC**( $\mathcal{S} : Z - X \leq 0$ )) partitions the set of constraints between  $Z$  and  $X$  (resp., between  $Z$  and  $X$ ) in  $L_1(\mathcal{S}, X, Z)$ ,  $L_2(\mathcal{S}, X, Z)$ ,  $L_3(\mathcal{S}, X, Z)$  (resp.,  $L_1(\mathcal{S}, Z, X)$ ,  $L_2(\mathcal{S}, Z, X)$ ,  $L_3(\mathcal{S}, Z, X)$ ). The cost of this operation is  $2 \times (|\mathcal{I}| - 1)$  (the number of constraints that are currently between  $Z$  and  $X$  in both directions). Eventually, when all traces have been processed, the overall cost of this operation is  $2 \times \sum_{i=0}^{|\mathcal{I}|-1} n = 2 \times ((|\mathcal{I}| - 1) \times |\mathcal{I}|) / 2 = \mathcal{O}(|\mathcal{I}|^2)$ . Since this term is greater of any other number of analyzed constraints between  $Z$  and any  $Y_b$  in the trace, an upper bound for the algorithm is given by  $\mathcal{O}(|\mathcal{I}|^2 \times |\tau|)$ . ◀

## 5 Conclusions and Future Work

Like any model-based engineering approach, creating a temporal network is a complex, time-consuming, and error-prone task. Along the lines of the bottom-up approach in the field of process mining, we proposed **CstnudMiner**, an algorithm for mining significant CSTNUDs from execution traces that also contain information on uncontrollable events. A CSTNUD is significant if it contains all time points, booleans and uncontrollable durations in the processed traces and each partial instantiation of a schedule and consistent set of literals arising from any processed trace satisfies all constraints involving those components. We proved that **CstnudMiner** runs in polynomial time with respect to the size of the set of input traces, and the length of each trace. Since in our approach once a trace is processed it can be forgotten, this paves the way for future “streaming” versions of the algorithm. As future work, we plan to carry out a thorough analysis of the properties of the mined CSTNUDs as well as adapting the algorithm for other classes of constraint networks involving resources either in isolation [20, 21, 23] or in conjunction with time [4].

---

## References


- 1 Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *EDBT '98*, pages 469–483. Springer, 1998.
- 2 Massimo Cairo, Luke Hunsberger, Roberto Posenato, and Romeo Rizzi. A streamlined model of conditional simple temporal networks - semantics and equivalence results. In *TIME 2017*, volume 90 of *LIPICs*, pages 10:1–10:19. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.TIME.2017.10.
- 3 Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. *Acta Informatica*, 53(6):681–722, 2016.
- 4 Carlo Combi, Roberto Posenato, Luca Viganò, and Matteo Zaverri. Conditional simple temporal networks with uncertainty and resources. *Journal of Artificial Intelligence Research*, 64:931–985, 2019. doi:10.1613/jair.1.11453.



- 5 Patrick R. Conrad and Brian C. Williams. Drake: An efficient executive for temporal plans with choice. *JAIR*, 42(1):607–659, 2011.
- 6 Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *TOSEM*, 7(3):215–249, 1998.
- 7 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991.
- 8 Joachim Herbst. A machine learning approach to workflow management. In *ECML '00*, pages 183–194. Springer, 2000.
- 9 Luke Hunsberger, Roberto Posenato, and Carlo Combi. The Dynamic Controllability of Conditional STNs with Uncertainty. In *PlanEx at ICAPS-2012*, pages 1–8, 2012.
- 10 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 *TIME 2015*, pages 4–18. IEEE CPS, 2015. doi:10.1109/TIME.2015.26.
- 11 Erez Karpas, Steven James Levine, Peng Yu, and Brian Charles Williams. Robust execution of plans for human-robot teams. In *ICAPS '15*, pages 342–346. AAAI Press, 2015.
- 12 Steven J. Levine and Brian C. Williams. Concurrent plan recognition and execution for human-robot teams. In *ICAPS '14*, pages 490–498. AAAI, 2014.
- 13 Ioannis Tsamardinou, Thierry Vidal, and Martha E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003. doi:10.1023/A:1025894003623.
- 14 Wil M. P. van der Aalst. Daelemans. automated discovery of workflow models from hospital data. In *BNAIC '01*, pages 25–26, 2001.
- 15 Wil M. P. van der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.
- 16 Wil M. P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow mining: Which processes can be rediscovered? In *Eindhoven University of Technology*, pages 1–25, 2002.
- 17 Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Jour. of Exp. & Theor. Artif. Intell.*, 11(1):23–45, 1999. doi:10.1080/095281399146607.
- 18 Peng Yu, Cheng Fang, and Brian Charles Williams. Resolving uncontrollable conditional temporal problems using continuous relaxations. In *ICAPS '14*, pages 341–349. AAAI, 2014.
- 19 Matteo Zavatteri. Conditional simple temporal networks with uncertainty and decisions. In *TIME 2017*, volume 90 of *LIPICs*, pages 23:1–23:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.TIME.2017.23.
- 20 Matteo Zavatteri, Romeo Rizzi, and Tiziano Villa. Dynamic Controllability and (J,K)-Resiliency in Generalized Constraint Networks with Uncertainty. In *ICAPS 2020*, pages 314–322. AAAI Press, 2020.
- 21 Matteo Zavatteri and Luca Viganò. Constraint networks under conditional uncertainty. In *ICAART 2018*, pages 41–52. SciTePress, 2018. doi:10.5220/0006553400410052.
- 22 Matteo Zavatteri and Luca Viganò. Conditional simple temporal networks with uncertainty and decisions. *Theoretical Computer Science*, 797:77–101, 2019. doi:10.1016/j.tcs.2018.09.023.
- 23 Matteo Zavatteri and Luca Viganò. Conditional uncertainty in constraint networks. In *Agents and Artificial Intelligence*, pages 130–160. Springer, 2019. doi:10.1007/978-3-030-05453-3\_7.



# Dynamic Branching in Qualitative Constraint Networks via Counting Local Models

Michael Sioutis<sup>1</sup> 

Faculty of Information Systems and Applied Computer Sciences, University of Bamberg, Germany  
<https://msioutis.gitlab.io/>  
michail.sioutis@uni-bamberg.de

Diedrich Wolter

Faculty of Information Systems and Applied Computer Sciences, University of Bamberg, Germany  
<https://www.uni-bamberg.de/en/sme/team/diedrich-wolter/>  
diedrich.wolter@uni-bamberg.de

---

## Abstract

We introduce and evaluate *dynamic branching* strategies for solving Qualitative Constraint Networks (QCNs), which are networks that are mostly used to represent and reason about spatial and temporal information via the use of simple qualitative relations, e.g., a constraint can be “Task *A* is scheduled *after or during* Task *C*”. In qualitative constraint-based reasoning, the state-of-the-art approach to tackle a given QCN consists in employing a backtracking algorithm, where the branching decisions during search are governed by the restrictiveness of the possible relations for a given constraint (e.g., *after* can be more restrictive than *during*). In the literature, that restrictiveness is defined a priori by means of static weights that are precomputed and associated with the relations of a given calculus, without any regard to the particulars of a given network instance of that calculus, such as its structure. In this paper, we address this limitation by proposing heuristics that dynamically associate a weight with a relation, based on the *count of local models* (or *local scenarios*) that the relation is involved with in a given QCN; these models are local in that they focus on triples of variables instead of the entire QCN. Therefore, our approach is adaptive and seeks to make branching decisions that preserve most of the solutions by determining what proportion of local solutions agree with that decision. Experimental results with a *random* and a *structured* dataset of QCNs of Interval Algebra show that it is possible to achieve up to 5 times better performance for structured instances, whilst maintaining non-negligible gains of around 20% for random ones.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Computing methodologies → Temporal reasoning; Computing methodologies → Spatial and physical reasoning

**Keywords and phrases** Qualitative constraints, spatial and temporal reasoning, counting local models, dynamic branching, adaptive algorithm

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.12

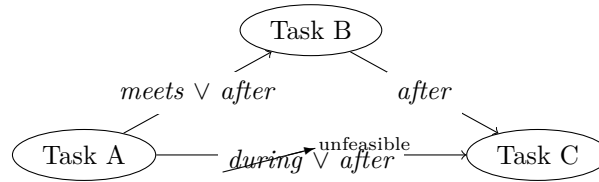
## 1 Introduction

Qualitative Spatial and Temporal Reasoning (QSTR) is a major field of study in AI that deals with the fundamental cognitive concepts of space and time in a human-like manner, via simple qualitative constraint languages [18, 8]. Such languages consist of abstract, qualitative, expressions like *inside*, *before*, or *north of* to spatially or temporally relate two or more objects to one another, without involving any quantitative information. Thus, QSTR offers tools for efficiently automating common-sense spatio-temporal reasoning and, hence, further boosts research to a plethora of application areas and domains that deal with spatio-temporal

---

<sup>1</sup> Corresponding author.



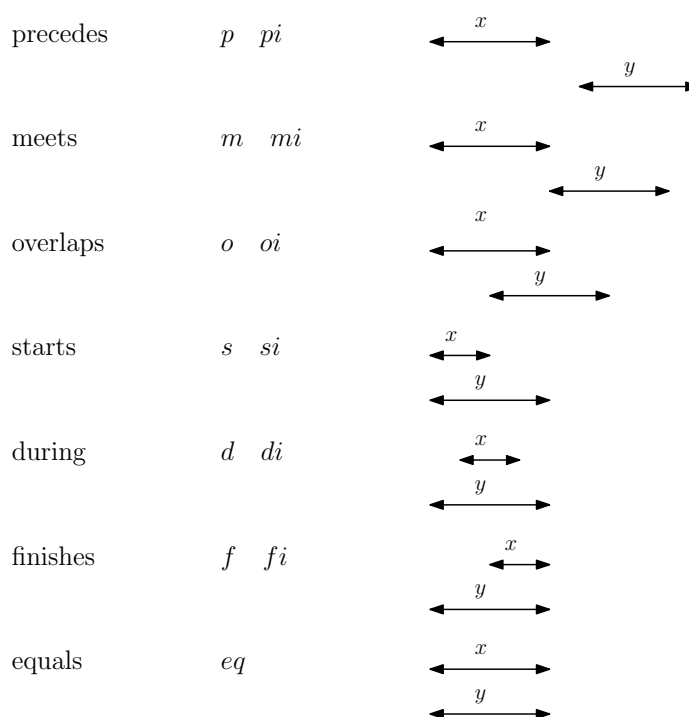


■ **Figure 1** The static weighting scheme in the literature dictates that relation *during* is less restrictive than relation *after* in general for the IA calculus and, hence, *during* should be preferred over *after* in branching decisions [39, Figure 9], but in the above simplified QCN *during* cannot appear in any solution; such schemes are defined for other calculi as well [13].

information, such as cognitive robotics [10], deep learning [17], visual explanation [37] and sensemaking [36], semantic question-answering [35], qualitative simulation [5], modal logic [21, 3, 20, 16, 11], temporal diagnosis [12], and stream reasoning [6, 14].

Qualitative spatial or temporal information may be modeled as a *Qualitative Constraint Network* (QCN), which is a network where the vertices correspond to spatial or temporal entities, and the arcs are labeled with qualitative spatial or temporal relations respectively. For instance  $x \leq y$  can be a temporal QCN over  $\mathbb{Z}$ . Given a QCN  $\mathcal{N}$ , the literature is particularly interested in its *satisfiability problem*, which is the problem of deciding if there exists a spatial or temporal interpretation of the variables of  $\mathcal{N}$  that satisfies its constraints, viz, a *solution* of  $\mathcal{N}$ . For instance,  $x = 0 \wedge y = 1$  is one of the infinitely many solutions of the aforementioned QCN, and  $x < y$  is the corresponding *scenario* that concisely represents all the cases where  $x$  is assigned a lesser value than  $y$ . In general, for most widely-adopted qualitative calculi the satisfiability problem is NP-complete [9]. In the sequel, we will be using Interval Algebra (IA) [1] as an illustrative example of a qualitative calculus.

**Motivation & Contribution.** The state-of-the-art constraint-based approach for tackling a given QCN consists in employing a backtracking algorithm, where each branching decision during search is guided by the restrictiveness of the possible relations for a given constraint. Currently, that restrictiveness is defined a priori by means of entirely precomputed static weights that are associated with the relations of a given calculus. That static strategy has two major problems: it assumes a uniform use of relations in QCNs (as weights are computed by equally considering all the relations of a calculus); and it does not exploit any structure that may exist in QCNs (a relation that is used to form more than one constraints in a given QCN, which is typically the case, may exhibit different levels of restrictiveness among those constraints). A simple example of how this scheme can be problematic is detailed in Figure 1. In this paper, we address this limitation by proposing a *dynamic branching* mechanism via heuristics that dynamically associate a weight with a relation during search, based on the *count of local models*, i.e., scenarios pertaining to triples of variables, that the relation is involved with in a given QCN. This makes our approach similar to a counting-based one for CSPs [24], as it too is adaptive and it too seeks to make branching decisions that preserve most of the solutions by determining what proportion of local solutions agree with that decision. Further inspiration was drawn from a recent work in [32], where it was observed that a scenario of a QCN may often be constructed collectively by relations that appear in many scenarios individually, i.e., a scenario of a QCN may often be constructed by selecting the most popular relation for each constraint. Finally, through an evaluation with a random and a structured dataset of QCNs of IA, we show that we may achieve up to 5 times better performance for structured instances, and gains of about 20% for random ones.



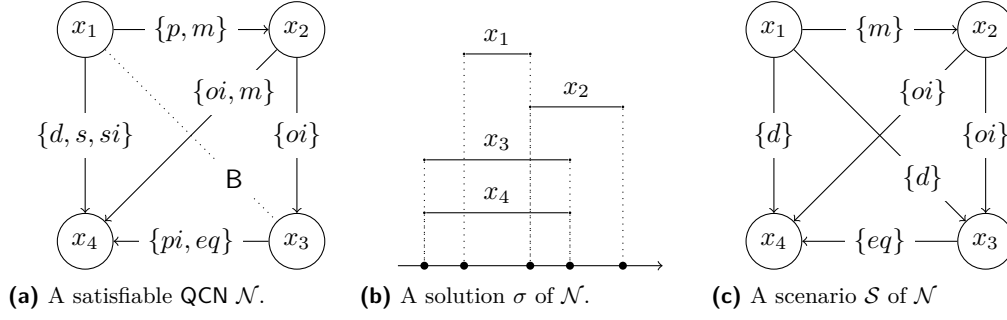
■ **Figure 2** The base relations of IA;  $\cdot i$  denotes the converse of  $\cdot$ .

The rest of the paper is organized as follows. In Section 2 we give some preliminaries on QSTR. Next, in Section 3 we propose our dynamic approach, discuss some dynamic heuristics that are used internally, and present the related algorithms. Then, in Section 4 we evaluate our approach with random and structured QCNs of IA and comment on the outcome. Finally, in Section 5 we draw some conclusive remarks and give directions for future work.

## 2 Preliminaries

A binary qualitative spatial or temporal constraint language, is based on a finite set  $\mathbf{B}$  of *jointly exhaustive and pairwise disjoint* relations, called the set of *base relations* [19], that is defined over an infinite domain  $\mathbf{D}$ . The base relations of a particular qualitative constraint language can be used to represent the definite knowledge between any two of its entities with respect to the level of granularity provided by the domain  $\mathbf{D}$ . The set  $\mathbf{B}$  contains the identity relation  $\text{Id}$ , and is closed under the *converse* operation ( $^{-1}$ ). Indefinite knowledge can be specified by a union of possible base relations, and is represented by the set containing them. Hence,  $2^{\mathbf{B}}$  represents the total set of relations. The set  $2^{\mathbf{B}}$  is equipped with the usual set-theoretic operations of union and intersection, the converse operation, and the *weak composition* operation denoted by the symbol  $\diamond$  [19]. For all  $r \in 2^{\mathbf{B}}$ , we have that  $r^{-1} = \bigcup \{b^{-1} \mid b \in r\}$ . The weak composition ( $\diamond$ ) of two base relations  $b, b' \in \mathbf{B}$  is defined as the smallest (i.e., strongest) relation  $r \in 2^{\mathbf{B}}$  that includes  $b \circ b'$ , or, formally,  $b \diamond b' = \{b'' \in \mathbf{B} \mid b'' \cap (b \circ b') \neq \emptyset\}$ , where  $b \circ b' = \{(x, y) \in \mathbf{D} \times \mathbf{D} \mid \exists z \in \mathbf{D} \text{ such that } (x, z) \in b \wedge (z, y) \in b'\}$  is the (true) composition of  $b$  and  $b'$ . For all  $r, r' \in 2^{\mathbf{B}}$ , we have that  $r \diamond r' = \bigcup \{b \diamond b' \mid b \in r, b' \in r'\}$ .

As an illustration, consider the well-known qualitative temporal constraint language of Interval Algebra (IA), introduced by Allen [1]. IA considers time intervals (as temporal entities) and the set of base relations  $\mathbf{B} = \{eq, p, pi, m, mi, o, oi, s, si, d, di, f, fi\}$  to



■ **Figure 3** Figurative examples of QCN terminology using IA.

encode knowledge about the temporal relations between intervals on the timeline, as depicted in Figure 2. Specifically, each base relation represents a particular ordering of the four endpoints of two intervals on the timeline, and  $eq$  is the identity relation  $Id$ .

Notably, most of the well-known and well-studied qualitative constraint languages, such as Interval Algebra [1] and RCC8 [25], are in fact *relation algebras* [9].

The problem of representing and reasoning about qualitative spatial or temporal information can be modeled as a *qualitative constraint network*, defined as follows:

- **Definition 1.** A *qualitative constraint network* (QCN) is a tuple  $(V, C)$  where:
- $V = \{v_1, \dots, v_n\}$  is a non-empty finite set of variables, each representing an entity of an infinite domain  $D$ ;
  - and  $C$  is a mapping  $C : V \times V \rightarrow 2^{\mathbb{B}}$  such that  $C(v, v) = \{Id\}$  for all  $v \in V$  and  $C(v, v') = (C(v', v))^{-1}$  for all  $v, v' \in V$ , where  $\bigcup \mathbb{B} = D \times D$ .

An example of a QCN of IA is shown in Figure 3a; for clarity, converse relations as well as  $Id$  loops are not mentioned or shown in the figure.

- **Definition 2.** Let  $\mathcal{N} = (V, C)$  be a QCN, then:
- a *solution* of  $\mathcal{N}$  is a mapping  $\sigma : V \rightarrow D$  such that  $\forall (u, v) \in V \times V, \exists b \in C(u, v)$  such that  $(\sigma(u), \sigma(v)) \in b$  (see Figure 3b);
  - $\mathcal{N}$  is *satisfiable* iff it admits a solution;
  - a *sub-QCN*  $\mathcal{N}'$  of  $\mathcal{N}$ , denoted by  $\mathcal{N}' \subseteq \mathcal{N}$ , is a QCN  $(V, C')$  such that  $C'(u, v) \subseteq C(u, v) \forall u, v \in V$ ; if in addition  $\exists u, v \in V$  such that  $C'(u, v) \subset C(u, v)$ , then  $\mathcal{N}' \subset \mathcal{N}$ ;
  - $\mathcal{N}$  is *atomic* iff  $\forall v, v' \in V, C(v, v')$  is a *singleton relation*, i.e., a relation  $\{b\}$  with  $b \in \mathbb{B}$ ;
  - a *scenario*  $\mathcal{S}$  of  $\mathcal{N}$  is an atomic satisfiable sub-QCN of  $\mathcal{N}$  (see Figure 3c);
  - the *constraint graph* of  $\mathcal{N}$  is the graph  $(V, E)$  where  $\{u, v\} \in E$  iff  $C(u, v) \neq \mathbb{B}$  and  $u \neq v$ ;
  - $\mathcal{N}$  is *trivially inconsistent*, denoted by  $\emptyset \in \mathcal{N}$ , iff  $\exists v, v' \in V$  such that  $C(v, v') = \emptyset$ ;
  - $\mathcal{N}$  is the *empty QCN* on  $V$ , denoted by  $\perp^V$ , iff  $C(u, v) = \emptyset$  for all  $u, v \in V$ .

Given a QCN  $\mathcal{N} = (V, C)$  and  $v, v' \in V$ , we introduce the following operation that substitutes  $C(v, v')$  with a relation  $r \in 2^{\mathbb{B}}$  to produce a new, modified, QCN:  $\mathcal{N}_{[v, v']/r}$  with  $r \in 2^{\mathbb{B}}$  yields the QCN  $\mathcal{N}' = (V, C')$ , where  $C'(v, v') = r$ ,  $C'(v', v) = r^{-1}$  and  $C'(u, u') = C(u, u') \forall (u, u') \in (V \times V) \setminus \{(v, v'), (v', v)\}$ .

We recall the definition of  $\mathcal{G}$ -consistency [4] (cf [27]), which entails consistency for all triples of variables in a QCN that form triangles in an accompanying graph  $G$ , and is a basic and widely-used local consistency for reasoning with QCNs.

- **Definition 1.** Given a QCN  $\mathcal{N} = (V, C)$  and a graph  $G = (V, E)$ ,  $\mathcal{N}$  is said to be  $\mathcal{G}$ -consistent iff  $\forall \{v_i, v_j\}, \{v_i, v_k\}, \{v_k, v_j\} \in E$  we have that  $C(v_i, v_j) \subseteq C(v_i, v_k) \diamond C(v_k, v_j)$ .

We note here that if  $G$  is complete,  $\overset{\circ}{G}$ -consistency becomes identical to  $\diamond$ -consistency [27], and, hence,  $\diamond$ -consistency is a special case of  $\overset{\circ}{G}$ -consistency. In the sequel, given a QCN  $\mathcal{N} = (V, C)$  of some calculus and a graph  $G = (V, E)$ , we assume that  $\overset{\circ}{G}(\mathcal{N})$  is computable. This assumption holds for most widely-adopted qualitative calculi [9].

### 3 Approach

In qualitative constraint-based reasoning, the state-of-the-art approach to check the satisfiability of a given QCN  $\mathcal{N}$ , consists in splitting every relation  $r$  that forms a constraint between two variables in  $\mathcal{N}$  into a subrelation  $r' \subseteq r$  that belongs to a set of relations  $\mathcal{A}$  over which the QCN becomes tractable [29]. In particular, for most widely-adopted qualitative calculi [9], such *split* sets are either known or readily available [26], and tractability is then achieved via the use of some local consistency in backtracking fashion; after every refinement of a relation  $r$  into a subrelation  $r'$ , the local consistency is enforced to know whether the refinement is valid or backtracking should occur and another subrelation should be chosen at an earlier point [29, Section 2]. One of the most essential and widely-used such local consistencies is  $\overset{\circ}{G}$ -consistency, where  $G$  is either the complete graph on the variables of  $\mathcal{N}$  [27], or a *triangulation* (*chordal completion*) of the constraint graph of  $\mathcal{N}$  [4].<sup>2</sup>

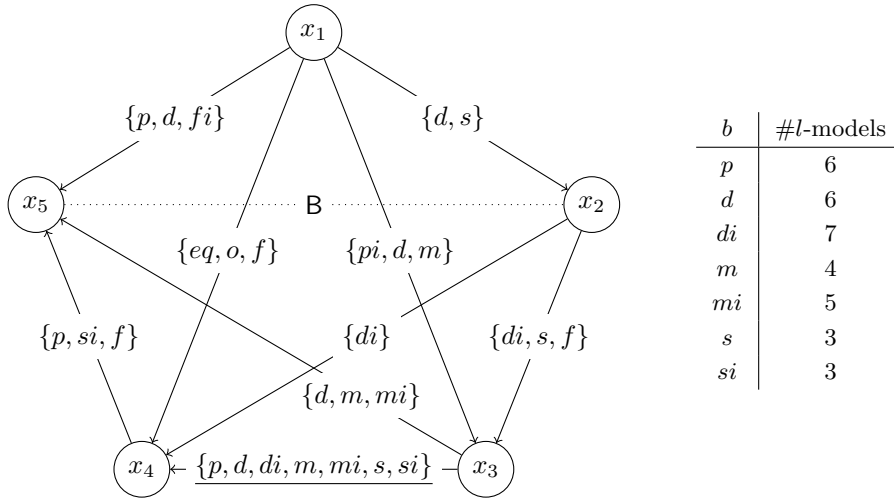
As an illustration, the subset  $\mathcal{H}_{IA}$  of the set of relations of Interval Algebra [23] is tractable for  $\overset{\circ}{G}$ -consistency, i.e.,  $\overset{\circ}{G}$ -consistency is complete for deciding the satisfiability of any QCN defined over  $\mathcal{H}_{IA}$  with respect to a triangulation  $G$  of its constraint graph [4]. That subset contains exactly those relations that are transformed to propositional Horn formulas when using the propositional encoding of Interval Algebra [23]. To further facilitate the reader, let us consider the constraint  $C(x_3, x_4)$  in the QCN of Interval Algebra in Figure 4. The relation  $\{mi, di, si, p, m, d, s\}$  that is associated with that constraint does not appear in the subset  $\mathcal{H}_{IA}$  and hence tractability is not guaranteed in general, but it can be split into subrelations  $\{mi\}, \{di, si\}, \{p, m\}, \{d, s\}$  with respect to  $\mathcal{H}_{IA}$ ; each of those subrelations belongs to  $\mathcal{H}_{IA}$ .

### Dynamic Selection of Subrelations via Counting Local Models

It is standard practice in the qualitative constraint-based reasoning community, and the constraint programming community in general, that, given a constraint of some QCN, a subrelation that is most likely to lead to a solution should be prioritized [39, 28]; in the context of finite-domain CSPs, this strategy is known as the *least-constraining value* heuristic [7].

Currently, the state of the art in qualitative constraint-based reasoning implements that selection strategy in a completely static manner. In particular, base relations of a calculus are assigned static weights a priori, and the overall weight that is associated with a subrelation corresponds to the sum of the weights of its base relations [39, 28]. In detail, a weight for a base relation is obtained by successively composing it with every possible relation and calculating the sum of the cardinalities of the results, which is then suitably scaled. Thus, the bigger the weight for a base relation is, the less restrictive that base relation is. For example, the weights of base relations  $d$  and  $s$  in Interval Algebra are 4 and 2 respectively and, consequently, the weight of relation  $\{d, s\}$  is  $4 + 2 = 6$  [39, Figure 9].

<sup>2</sup> Please refer to [15] for the properties that are needed to exploit triangulations of QCNs in terms of tractability preservation.



■ **Figure 4** Given the above QCN  $\mathcal{N} = (V, C)$  of IA, a partition of  $C(x_3, x_4)$  with respect to the subset  $\mathcal{H}_{IA}$  [23] is  $\{\{mi\}, \{di, si\}, \{p, m\}, \{d, s\}\}$ ; for this QCN, heuristics *dynamic\_avg* and *dynamic\_sum* would prioritize relation  $\{p, m\}$ , and heuristics *static* [39], *dynamic\_max*, and *dynamic\_min* would prioritize relations  $\{d, s\}$ ,  $\{di, si\}$ , and  $\{mi\}$  respectively.

Two major problems with the aforementioned *static* strategy is that it assumes a uniform use of relations in QCNs (since weights are computed by equally considering all the relations of a calculus), and it does not exploit any structure that may be present in QCNs (a relation that is used to form more than one constraints in a given QCN, which is typically the case, may exhibit different levels of restrictiveness among those constraints).

In this paper, we propose the selection of subrelations to be *dynamic* and, in particular, based on the count of *local models* that the individual base relations of a subrelation are part of. Let  $\mathcal{N} \downarrow_{V'}$ , with  $V' \subseteq V$ , denote the QCN  $\mathcal{N} = (V, C)$  restricted to  $V'$ , we formally define the notion of local models as follows:

► **Definition 3** (local models). Given a QCN  $\mathcal{N} = (V, C)$ , a graph  $G = (V, E)$ , and a constraint  $C(v, v')$  with  $\{v, v'\} \in E$ , the *local models* of a base relation  $b \in C(v, v')$  are the scenarios  $\mathcal{S} = (V', C')$  of  $\mathcal{N} \downarrow_{V'}$ , with  $V' = \{v, v', u\}$ , such that  $\{v, u\}, \{u, v'\} \in E$  and  $C(v, v') = \{b\}$ .

Simply put, given a QCN  $(V, C)$ , a graph  $G = (V, E)$ , and a constraint  $C(v, v')$  with  $\{v, v'\} \in E$ , we count how many times a given base relation  $b \in C(v, v')$  participates in the scenarios of each triangle in  $G$  that involves variables  $v$  and  $v'$ , i.e., the local models from our perspective. In that sense, our approach can be seen as being similar to a counting-based one for CSPs [24], which, as our own method, formalizes a framework that is adaptive and seeks to make branching decisions that preserve most of the solutions by determining what proportion of local solutions agree with that decision. We devise the following strategies for choosing a subrelation from a given set of subrelations:

- **dynamic\_f**: for each subrelation  $r'$  find the  $f$  count of local models among each base relation  $b \in r'$ , where  $f \in \{\max, \min, \text{avg}, \text{sum}\}$ , then choose the subrelation for which the highest such count was obtained.

In the context of counting local models, *dynamic\_max*, *dynamic\_min*, *dynamic\_avg*, and *dynamic\_sum* prioritize the subrelation with the best *most*, *least*, *on average*, and *in aggregate* supportive base relation respectively. At this point, let us revisit the QCN of

■ **Algorithm 1**  $\text{Refinement}(\mathcal{N}, G, \mathcal{A}, f)$ .

---

```

in      : A QCN  $\mathcal{N} = (V, C)$ , a graph  $G = (V, E)$ , a subset  $\mathcal{A} \subseteq 2^{\mathbb{B}}$ , and a function
            $f \in \{\text{max}, \text{min}, \text{avg}, \text{sum}\}$ .
out     : A refinement of  $\mathcal{N}$  with respect to  $G$  over  $\mathcal{A}$ , or  $\perp^V$ .
1 begin
2    $\mathcal{N} \leftarrow \circ_G(\mathcal{N})$ ;
3   if  $\emptyset \in \circ_G(\mathcal{N})$  then
4     return  $\perp^V$ ;
5   if  $\forall \{v, v'\} \in E, C(v, v') \in \mathcal{A}$  then
6     return  $\mathcal{N}$ ;
7    $(v, v') \leftarrow \{v, v'\} \in E$  such that  $C(v, v') \notin \mathcal{A}$ ;
8   foreach  $r \in \text{dynamicSelection}(\mathcal{N}, G, \mathcal{A}, (v, v'), f)$  do
9      $\text{result} \leftarrow \text{Refinement}(\mathcal{N}_{[v, v']/r}, G, \mathcal{A}, f)$ ;
10    if  $\text{result} \neq \perp^V$  then
11      return  $\text{result}$ ;
12  return  $\perp^V$ ;

```

---

■ **Algorithm 2**  $\text{dynamicSelection}(\mathcal{N}, G, \mathcal{A}, (v, v'), f)$ .

---

```

in      : A QCN  $\mathcal{N} = (V, C)$ , a graph  $G = (V, E)$ , a subset  $\mathcal{A} \subseteq 2^{\mathbb{B}}$ , a pair of variables
            $(v, v')$ , and a function  $f \in \{\text{max}, \text{min}, \text{avg}, \text{sum}\}$ .
out     : A relation  $r \in \mathcal{A}$ .
1 begin
2    $\text{counter} \leftarrow \text{hashTable}()$ ;
3   foreach  $r \in \{r_1, r_2, \dots, r_n \in \mathcal{A} \mid \{r_1, r_2, \dots, r_n\} \text{ is a partition of } C(v, v')\}$  do
4      $\text{counter}[r] \leftarrow f\{\text{localModels}(b, \mathcal{N}, G, (v, v')) \mid b \in r\}$ ;
5   while  $\text{counter}$  is not empty do
6      $r \leftarrow \text{counter}$  key paired with the maximum value;
7     remove  $r$  from  $\text{counter}$ ;
8     yield  $r$ ;

```

---

Interval Algebra in Figure 4, where the relation  $\{mi, di, si, p, m, d, s\}$  that is associated with the constraint  $C(x_3, x_4)$  is split into subrelations  $\{mi\}, \{di, si\}, \{p, m\}, \{d, s\}$  with respect to subset  $\mathcal{H}_{IA}$ . By viewing the table that lists the count of local models for each base relation in  $C(x_3, x_4)$  on the right-hand side of the figure, the reader can verify that each strategy correctly prioritizes its subrelation of choice according to its objective; as a reminder, the weights associated with the static strategy detailed earlier are provided in [39, Figure 9].

## Tackling QCNs via Incorporating Dynamic Branching

For reference, a variation of the state-of-the-art backtracking algorithm for solving a QCN is provided in Algorithm 1, the main difference to the one appearing in the literature [29, Section 2] being the use of dynamic selection of subrelations, in line 8, instead of selection based on static weights. Another difference is the use of a graph as a parameter, but, over the past few years, this has become a standard way of generalizing the original algorithm to exploit certain properties of a calculus that relate to graphs, see [34] and references therein.

The dynamic strategies that we described earlier are formally presented in Algorithms 2 and 3. In particular, in lines 2–4 of Algorithm 2 we calculate the count of local models for each base relation of each subrelation that pertains to a given constraint. This calculation



■ **Algorithm 3** `localModels( $b, \mathcal{N}, G, (v, v')$ )`.

---

```

in      : A base relation  $b$ , a QCN  $\mathcal{N} = (V, C)$ , a graph  $G = (V, E)$ , and a pair of variables
           ( $v, v'$ ).
out     : An integer.
1 begin
2    $count \leftarrow 0$ ;
3   foreach  $u \in N_G(v) \cap N_G(v')$  do
4     foreach  $(b', b'') \in C(v, u) \times C(u, v')$  do
5       if  $b \in b' \diamond b''$  then
6          $count \leftarrow count + 1$ ;
7   return  $count$ ;

```

---

is performed via a call to Algorithm 3. After obtaining the count of models for each such base relation, we implement the chosen strategy by applying the respective function among  $\{\max, \min, \text{avg}, \text{sum}\}$  on the results. Now, each subrelation is associated with a number, a dynamic weight, and in lines 5–8 the subrelation with the highest dynamic weight is prioritized each time there is a need for a new subrelation to be tried out in an assignment.

**Complexity Analysis.** Given the fact that for a calculus the number of its base relations, i.e.,  $|\mathbf{B}|$ , can be viewed as a constant, Algorithm 2 calculates the count of local models for a base relation in a given constraint in linear time in the maximum degree of the graph  $G$  that is used as a parameter; each subsequent prioritization of a subrelation based on those calculated counts (lines 5–8) takes constant time. In particular, given a QCN  $\mathcal{N} = (V, C)$  and a graph  $G = (V, E)$ , Algorithm 2 runs in  $\Theta(\Delta(G))$  time. In practice, there was no noticeable slowdown for the dataset that we consider in this paper (see Section 4), which is not surprising, as the search space for solving a QCN is  $O(|\mathbf{B}|^{|E|})$  in general.

## 4 Evaluation

In this section we evaluate the proposed dynamic branching heuristics, as well as the state-of-the-art static branching strategy that appears in the literature, with respect to the fundamental reasoning problem of *satisfiability checking* of QCNs. Specifically, we explore the *efficiency* of the involved heuristics in determining the satisfiability of a given network instance when used in the standard backtracking algorithm (see Algorithm 1), and investigate their *fitness score* too, which is the difference “% of times a heuristic  $f$  is the best choice” – “% of times a heuristic  $f$  is the worst choice”; clearly,  $\text{fitness score} \in [-100\%, 100\%]$ . Finally, we also present results for two virtual portfolios of reasoners that always make the *best* and *worst* choice of a heuristic respectively for a given network instance.

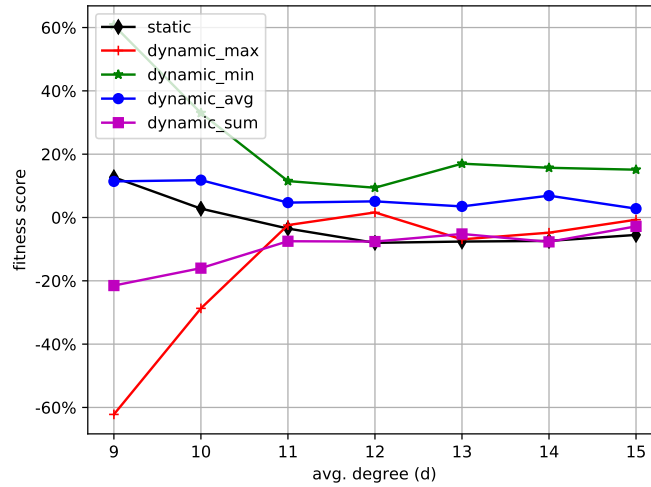
**Technical specifications.** The evaluation was carried out on a computer with an Intel Core i7-8565U processor, 16 GB of RAM, and the Ubuntu 18.04.4 LTS x86\_64 OS. All algorithms were coded in Python and run using the PyPy interpreter under version 5.10.0, which implements Python 2.7.13. Only one CPU core was used per run.

**Dataset.** We generated 7 000 *random* instances of Interval Algebra using model  $H(n = 40, d)$  [22], 1 000 for each constraint graph degree value  $d \in \{9, 10, 11, 12, 13, 14, 15\}$  specifically, and 4 000 *structured* instances of Interval Algebra using model  $BA(n = 80, m, 3\text{CNF})$  [31],



■ **Table 1** Evaluation with random IA networks that were generated using model  $H(n = 40, d) [22]$ ;  $\frac{\text{median|average|maximum \# of visited nodes}}{\text{median|average|maximum CPU time}}$ .

d	best	static	dynamic_max	dynamic_min	dynamic_avg	dynamic_sum	worst
9	62.0 61.8 95.0 0.0s 0.0s 10.8s	78.0 79.4 413.0 0.0s  <b>0.0s</b>  11.0s	107.0 112.0 258.0 0.0s 0.0s 10.8s	66.0  <b>65.9</b>  144.0 0.0s 0.0s 10.9s	78.0 79.5 350.0 0.0s 0.0s 10.8s	94.0 95.6 520.0 0.0s 0.0s 10.9s	114.0 120.4 520.0 0.0s 0.0s 11.0s
10	52.0 52.6 168.0 0.0s 0.0s 10.9s	71.0 125.8 8.9k 0.0s 0.1s 10.9s	85.0 101.3 2.0k 0.0s  <b>0.0s</b>  10.9s	60.0  <b>89.9</b>  2.9k 0.0s 0.0s 11.0s	67.0 104.4 8.7k 0.0s 0.1s 10.9s	81.0 129.6 6.6k 0.0s 0.1s 10.9s	108.0 227.7 8.9k 0.0s 0.1s 11.0s
11	59.0 679.3 388.0k 0.0s 0.5s 4.4m	174.0 2.5k 500.3k 0.1s 1.7s 5.3m	143.5 2.1k 388.0k 0.1s 1.4s 4.4m	141.0  <b>1.7k</b>  546.8k 0.1s  <b>1.2s</b>  6.2m	133.0 2.1k 619.7k 0.1s 1.4s 6.9m	181.0 2.6k 474.2k 0.1s 1.9s 5.4m	682.0 5.0k 619.7k 0.4s 3.4s 6.9m
12	931.5 10.3k 953.5k 0.7s 7.4s 10.7m	4.4k 24.6k 958.6k 3.1s 16.8s 10.7m	3.5k 21.0k 1.0m 2.6s 15.1s 11.6m	3.1k 18.7k 1.0m 2.3s 13.5s 12.0m	2.9k  <b>18.6k</b>  996.1k 2.2s  <b>13.3s</b>  11.4m	4.7k 24.6k 953.5k 3.5s 17.4s 10.8m	10.0k 34.9k 1.0m 7.4s 24.5s 12.0m
13	2.0k 7.6k 214.9k 1.6s 5.9s 2.7m	3.4k 11.9k 277.3k 2.6s 8.7s 3.1m	3.2k 11.2k 302.8k 3.0s 9.8s 4.1m	2.7k  <b>10.8k</b>  245.6k 2.1s  <b>8.2s</b>  3.0m	2.9k 10.9k 272.8k 2.3s 8.4s 3.2m	3.4k 11.7k 283.8k 2.7s 9.1s 3.5m	4.6k 15.7k 302.8k 3.7s 12.3s 4.1m
14	460.0 1.3k 58.1k 0.8s 1.1s 43.9s	720.0 1.9k 64.9k 0.5s  <b>1.4s</b>  47.5s	700.5 1.9k 77.5k 0.5s 1.5s 58.0s	584.5  <b>1.6k</b>  64.9k 0.6s 1.6s 1.0m	619.0 1.7k 63.8k 0.6s 1.6s 55.8s	708.5 2.0k 106.0k 0.6s 1.6s 1.3m	983.0 2.5k 106.0k 0.8s 2.1s 1.3m
15	113.0 220.7 2.6k 0.4s 0.2s 11.2s	179.0 365.2 5.4k 0.4s 0.3s 11.2s	168.5 350.2 4.4k 0.1s 0.3s 11.3s	157.0  <b>285.7</b>  2.8k 0.1s  <b>0.2s</b>  11.4s	167.0 317.9 4.8k 0.4s 0.3s 11.3s	176.0 360.2 5.9k 0.4s 0.3s 11.3s	251.5 463.7 5.9k 0.2s 0.4s 11.4s

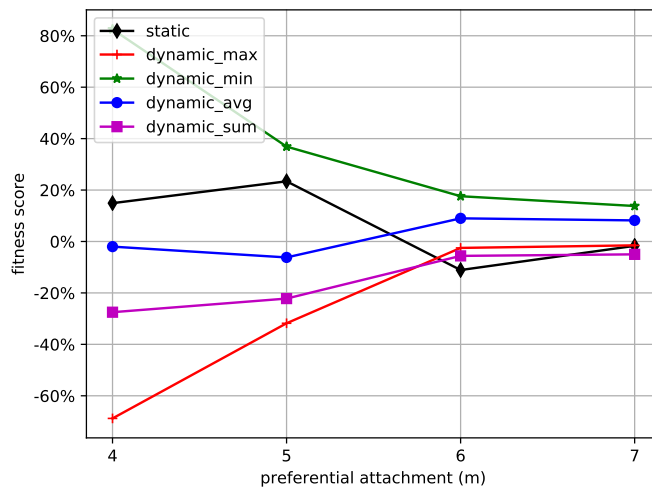


■ **Figure 5** Fitness score of each strategy for the instances of Table 1; “% of times a heuristic  $f$  is the best choice” – “% of times a heuristic  $f$  is the worst choice”.

1 000 for each constraint graph *preferential attachment* [2] value  $m \in \{4, 5, 6, 7\}$  specifically. In both of the aforementioned generation models, constraints were picked from the set of relations expressible in 3CNF when transformed into first-order formulae [22], in order to increase the branching factor in the search tree as much as possible. Finally, regarding the graphs that were given as input to our algorithms, the *maximum cardinality search* algorithm [38] was used to obtain triangulations of the constraint graphs of our QCNs.

**Results.** Regarding the generation model  $H(n = 40, d)$ , the main results are presented in Table 1. The dynamic strategies of *dynamic\_min* and *dynamic\_avg* are up to 20% faster on average than the *static* one in the phase transition of the tested instances.<sup>3</sup> Specifically, the phase transition covers mostly the case where  $d = 12$ , and a little less the case where  $d = 13$ . With respect to the rest of the dynamic heuristics, viz., *dynamic\_max* and *dynamic\_sum*, the results suggest that *dynamic\_max* outperforms *static* by a small margin on average in the phase transition, whilst *dynamic\_sum* almost mimics the performance of *static*, if not arguably being a little worse than *static* overall. This last finding informs us that relying too much on the number of base relations of a relation (viz., the cardinality of a relation) is a bad choice in general, i.e., it is better to focus on few base relations individually, where each one appears in many local models (quality), than on many base relations aggregately, where each one appears in few local models (quantity). The aforementioned results are depicted from a different perspective and complemented in Figure 5, where the fitness score for each heuristic is detailed. The superiority of heuristics *dynamic\_min* and *dynamic\_avg* among all strategies becomes even more so clear, and the marginal performance gains of *dynamic\_max*, and *dynamic\_sum* at times with respect to *static* in the phase transition are well-captured by their fitness scores too. Finally, at this point, it is interesting to observe the performance of the virtual portfolios of reasoners *best* and *worst*; as a reminder these always make the *best*

<sup>3</sup> Even though the improvement for this particular dataset may not seem that drastic, bear in mind that the instances of *this* dataset have little to no structure as their constraint graphs are regular graphs.



■ **Figure 6** Fitness factor of each strategy for the instances of Table 2; “% of times a heuristic  $f$  is the best choice” – “% of times a heuristic  $f$  is the worst choice”.

and *worst* choice of a heuristic respectively for a given network instance. The performance of portfolio *best* in particular allows us to be optimistic about future research in dynamic strategies, since it shows that there is still a lot of room for improvement. Specifically, research could be carried out both in terms of defining new dynamic strategies and in terms of devising selection protocols that choose among already existing strategies.

Regarding the generation model  $BA(n = 80, m, 3CNF)$ , the results are presented in Table 2 and Figure 6. Here, *dynamic\_min* and *dynamic\_avg* are up to 3 and 5 times faster on average respectively than the *static* one in the phase transition of the tested instances, which appears for  $m = 6$ . The rest of the results are qualitative similar to the previous dataset.

Since the runtime distribution is heavy-tailed for both datasets, the interested reader may want to look into the 0.5<sup>th</sup> percentile of most difficult instances pertaining to Tables 1 and 2 for each strategy, depicted in Figures 7 and 8 respectively in Appendix A.

## 5 Conclusion and Future Work

We introduced and evaluated *dynamic branching* strategies for solving QCNs via backtracking search, based on the *count* of *local models* (or *local scenarios*) that a possible relation for a given constraint is involved with in a considered QCN. Thus, we addressed a limitation in the state of the art in qualitative constraint-based reasoning, where the selection of a possible relation for a given constraint is dictated a priori by precomputed static weights, without any regard to the particulars of a given network instance of that calculus, such as its structure. Our approach is adaptive and seeks to make branching decisions that preserve most of the solutions by determining what proportion of local solutions agree with that decision. An evaluation with a *random* and a *structured* dataset of QCNs of Interval Algebra showed that up to 5 times better performance may be achieved for structured instances, whilst non-negligible gains of around 20% are maintained for random ones.

As this is a new approach, there are many directions for future work. In particular, we aim to devise selection protocols that choose among different strategies, as the performance of the virtual portfolio *best* in our evaluation, which always makes the *best* choice of a heuristic for

■ **Table 2** Evaluation with structured IA networks that were generated using model BA( $n = 80, m, 3CNF$ ) [31];  $\frac{\text{median|average|maximum \# of visited nodes}}{\text{median|average|maximum CPU time}}$ .

$m$	<i>best</i>	<i>static</i>	<i>dynamic_max</i>	<i>dynamic_min</i>	<i>dynamic_avg</i>	<i>dynamic_sum</i>	<i>worst</i>
4	$\frac{144.0 144.1 177.0}{0.0s 0.0s 10.7s}$	$\frac{166.0 167.1 257.0}{0.0s 0.0s 10.8s}$	$\frac{213.0 216.0 332.0}{0.0s 0.0s 10.8s}$	$\frac{146.0 146.3 320.0}{0.0s 0.0s 10.8s}$	$\frac{178.0 177.0 231.0}{0.0s 0.0s 10.8s}$	$\frac{198.0 198.9 321.0}{0.0s 0.0s 10.7s}$	$\frac{219.0 222.3 332.0}{0.0s 0.0s 10.8s}$
5	$\frac{113.0 114.5 550.0}{0.0s 0.0s 10.8s}$	$\frac{128.0 154.4 1.0k}{0.0s 0.1s 10.8s}$	$\frac{155.0 173.3 1.6k}{0.0s 0.1s 11.0s}$	$\frac{124.0 139.9 2.9k}{0.0s 0.1s 10.9s}$	$\frac{141.0 159.8 1.2k}{0.0s 0.1s 10.8s}$	$\frac{151.0 177.8 1.2k}{0.0s 0.1s 10.9s}$	$\frac{178.0 226.3 2.9k}{0.1s 0.1s 11.0s}$
6	$\frac{121.0 452.3 72.5k}{0.1s 0.6s 1.7m}$	$\frac{235.0 4.7k 1.3m}{0.3s 5.4s 23.7m}$	$\frac{201.0 7.4k 3.8m}{0.2s 8.1s 1.1h}$	$\frac{172.0 1.4k 460.4k}{0.2s 1.8s 8.8m}$	$\frac{182.5 933.4 139.6k}{0.2s 1.2s 3.1m}$	$\frac{223.0 4.0k 2.0m}{0.3s 4.6s 34.5m}$	$\frac{337.5 10.6k 3.8m}{0.4s 11.8s 1.1h}$
7	$\frac{34.0 82.4 3.4k}{0.0s 0.1s 11.1s}$	$\frac{51.0 178.9 10.4k}{0.1s 0.2s 15.2s}$	$\frac{50.0 168.7 17.9k}{0.1s 0.2s 27.7s}$	$\frac{47.0 110.8 3.4k}{0.1s 0.2s 11.3s}$	$\frac{49.0 129.4 5.0k}{0.1s 0.2s 11.2s}$	$\frac{51.0 206.6 22.6k}{0.1s 0.3s 35.2s}$	$\frac{74.5 252.2 22.6k}{0.1s 0.4s 35.2s}$

a given network instance, revealed that there is still a lot of room for improvement. Further, more sophisticated dynamic heuristics could be developed by going beyond triples of variables, which currently form the local models, and engaging larger parts of an instance. Finally, we would like to pair this approach with ongoing research on singleton consistencies [33, 30], and implement adaptive reasoners where the level of consistency checking during search would be adjusted according to the count of local models pertaining to a given constraint.

---

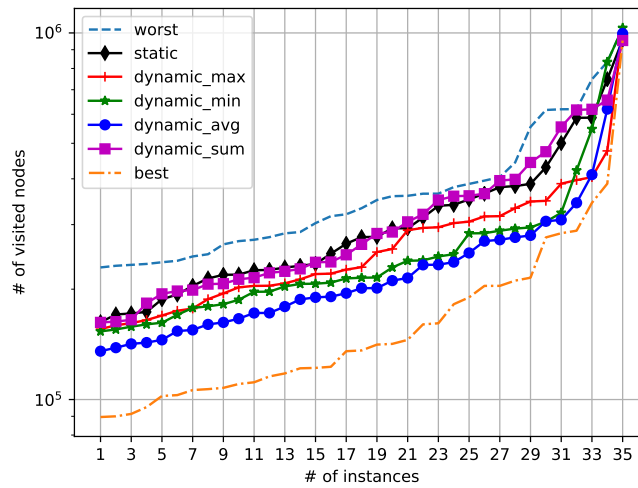
## References

- 1 James F. Allen. Maintaining Knowledge about Temporal Intervals. *Commun. ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.
- 2 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science (New York, N.Y.)*, 286(5439):509–512, 1999. doi:10.1126/science.286.5439.509.
- 3 Davide Bresolin, Dario Della Monica, Angelo Montanari, Pietro Sala, and Guido Sciavicco. Decidability and complexity of the fragments of the modal logic of Allen’s relations over the rationals. *Inf. Comput.*, 266:97–125, 2019. doi:10.1016/j.ic.2019.02.002.
- 4 Assef Chmeiss and Jean-Francois Condotta. Consistency of Triangulated Temporal Qualitative Constraint Networks. In *ICTAI*, pages 799–802, 2011. doi:10.1109/ICTAI.2011.125.
- 5 Zhan Cui, Anthony G. Cohn, and David A. Randell. Qualitative Simulation Based on a Logical Formalism of Space and Time. In *AAAI*, pages 679–684, 1992. URL: <http://www.aaai.org/Library/AAAI/1992/aaai92-105.php>.
- 6 Daniel de Leng and Fredrik Heintz. Qualitative Spatio-Temporal Stream Reasoning with Unobservable Intertemporal Spatial Relations Using Landmarks. In *AAAI*, pages 957–963, 2016. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12077>.
- 7 Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.
- 8 Frank Dylla, Jae Hee Lee, Till Mossakowski, Thomas Schneider, André van Delden, Jasper van de Ven, and Diedrich Wolter. A Survey of Qualitative Spatial and Temporal Calculi: Algebraic and Computational Properties. *ACM Comput. Surv.*, 50(1):7:1–7:39, 2017. doi:10.1145/3038927.
- 9 Frank Dylla, Till Mossakowski, Thomas Schneider, and Diedrich Wolter. Algebraic Properties of Qualitative Spatio-temporal Calculi. In *COSIT*, pages 516–536, 2013. doi:10.1007/978-3-319-01790-7\_28.
- 10 Frank Dylla and Jan Oliver Wallgrün. Qualitative Spatial Reasoning with Conceptual Neighborhoods for Agent Control. *J. Intell. Robot. Syst.*, 48(1):55–78, 2007. doi:10.1007/s10846-006-9099-4.
- 11 David Gabelaia, Roman Kontchakov, Ágnes Kurucz, Frank Wolter, and Michael Zakharyashev. Combining Spatial and Temporal Logics: Expressiveness vs. Complexity. *J. Artif. Intell. Res.*, 23:167–243, 2005. doi:10.1613/jair.1537.
- 12 Johann Gamper and Wolfgang Nejdl. Abstract temporal diagnosis in medical domains. *Artificial Intelligence in Medicine*, 10(3):209–234, 1997. doi:10.1016/S0933-3657(97)00393-X.
- 13 Zeno Gantner, Matthias Westphal, and Stefan Wöfl. GQR-A Fast Reasoner for Binary Qualitative Constraint Calculi. In *AAAI Workshop on Spatial and Temporal Reasoning*, 2008.
- 14 Fredrik Heintz and Daniel de Leng. Spatio-Temporal Stream Reasoning with Incomplete Spatial Information. In *ECAI*, pages 429–434, 2014. doi:10.3233/978-1-61499-419-0-429.
- 15 Jinbo Huang. Compactness and its implications for qualitative spatial and temporal reasoning. In *KR*, 2012.
- 16 Roman Kontchakov, Ágnes Kurucz, Frank Wolter, and Michael Zakharyashev. Spatial Logic + Temporal Logic = ? In *Handbook of Spatial Logics*, pages 497–564. Springer-Verlag, 2007. doi:10.1007/978-1-4020-5587-4\_9.
- 17 Nikhil Krishnaswamy, Scott Friedman, and James Pustejovsky. Combining Deep Learning and Qualitative Spatial Reasoning to Learn Complex Structures from Sparse Examples with Noise. In *AAAI*, pages 411–415, 2019.

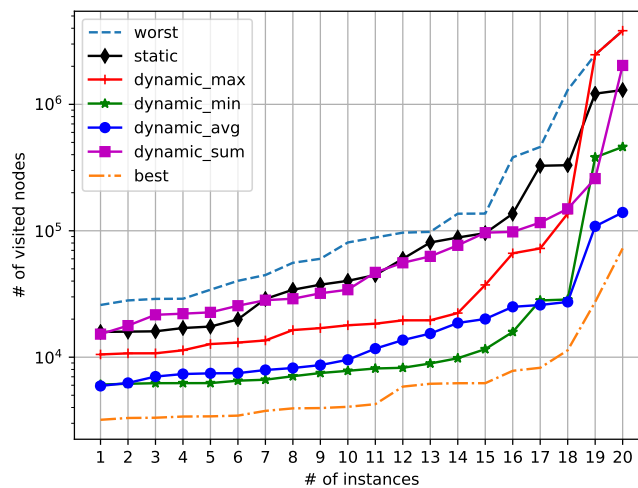
- 18 Gérard Ligozat. *Qualitative Spatial and Temporal Reasoning*. ISTE. Wiley, 2013.
- 19 Gérard Ligozat and Jochen Renz. What Is a Qualitative Calculus? A General Framework. In *PRICAI*, pages 53–64, 2004. doi:10.1007/978-3-540-28633-2\_8.
- 20 Antonio Morales, Isabel Navarrete, and Guido Sciavicco. A new modal logic for reasoning about space: spatial propositional neighborhood logic. *Ann. Math. Artif. Intell.*, 51(1):1–25, 2007. doi:10.1007/s10472-007-9083-0.
- 21 Emilio Muñoz-Velasco, Mercedes Pelegrín-García, Pietro Sala, Guido Sciavicco, and Ionel Eduard Stan. On coarser interval temporal logics. *Artif. Intell.*, 266:1–26, 2019. doi:10.1016/j.artint.2018.09.001.
- 22 Bernhard Nebel. Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class. *Constraints*, 1(3):175–190, 1997. doi:10.1007/BF00137869.
- 23 Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen’s Interval Algebra. *J. ACM*, 42(1):43–66, 1995. doi:10.1145/200836.200848.
- 24 Gilles Pesant, Claude-Guy Quimper, and Alessandro Zanarini. Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems. *J. Artif. Intell. Res.*, 43:173–210, 2012. doi:10.1613/jair.3463.
- 25 David A. Randell, Zhan Cui, and Anthony Cohn. A Spatial Logic Based on Regions and Connection. In *KR*, pages 165–176, 1992.
- 26 Jochen Renz. Qualitative Spatial and Temporal Reasoning: Efficient Algorithms for Everyone. In *IJCAI*, pages 526–531, 2007.
- 27 Jochen Renz and Gérard Ligozat. Weak Composition for Qualitative Spatial and Temporal Reasoning. In *CP*, pages 534–548, 2005. doi:10.1007/11564751\_40.
- 28 Jochen Renz and Bernhard Nebel. Efficient Methods for Qualitative Spatial Reasoning. *J. Artif. Intell. Res.*, 15:289–318, 2001. doi:10.1613/jair.872.
- 29 Jochen Renz and Bernhard Nebel. Qualitative Spatial Reasoning Using Constraint Calculi. In *Handbook of Spatial Logics*, pages 161–215. Springer-Verlag, 2007. doi:10.1007/978-1-4020-5587-4\_4.
- 30 Michael Sioutis. Just-In-Time Constraint-Based Inference for Qualitative Spatial and Temporal Reasoning. *KI*, 34(2):259–270, 2020. doi:10.1007/s13218-020-00652-z.
- 31 Michael Sioutis, Jean-François Condotta, and Manolis Koubarakis. An Efficient Approach for Tackling Large Real World Qualitative Spatial Networks. *Int. J. Artif. Intell. Tools*, 25:1–33, 2016. doi:10.1142/S0218213015500311.
- 32 Michael Sioutis, Zhiguo Long, and Tomi Janhunen. On Robustness in Qualitative Constraint Networks. In *IJCAI*, pages 448–455, 2020. To appear.
- 33 Michael Sioutis, Anastasia Paparrizou, and Jean-François Condotta. Collective singleton-based consistency for qualitative constraint networks: Theory and practice. *Theor. Comput. Sci.*, 797:17–41, 2019. doi:10.1016/j.tcs.2019.02.028.
- 34 Michael Sioutis, Yakoub Salhi, and Jean-François Condotta. Studying the use and effect of graph decomposition in qualitative spatial and temporal reasoning. *Knowl. Eng. Rev.*, 32:e4, 2017. doi:10.1017/S026988891600014X.
- 35 Jakob Suchan and Mehul Bhatt. Semantic Question-Answering with Video and Eye-Tracking Data: AI Foundations for Human Visual Perception Driven Cognitive Film Studies. In *IJCAI*, pages 2633–2639, 2016. URL: <http://www.ijcai.org/Abstract/16/374>.
- 36 Jakob Suchan, Mehul Bhatt, and Srikrishna Varadarajan. Out of Sight But Not Out of Mind: An Answer Set Programming Based Online Abduction Framework for Visual Sensemaking in Autonomous Driving. In *IJCAI*, pages 1879–1885, 2019. doi:10.24963/ijcai.2019/260.
- 37 Jakob Suchan, Mehul Bhatt, Przemyslaw Andrzej Walega, and Carl P. L. Schultz. Visual Explanation by High-Level Abduction: On Answer-Set Programming Driven Reasoning About Moving Objects. In *AAAI*, pages 1965–1972, 2018. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17303>.

- 38 Robert E. Tarjan and Mihalis Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984. doi:10.1137/0213035.
- 39 Peter van Beek and Dennis W. Manchak. The design and experimental analysis of algorithms for temporal reasoning. *J. Artif. Intell. Res.*, 4(1):1–18, 1996.

**A** Evaluation Figures



■ **Figure 7** Insight into the 0.5<sup>th</sup> percentile of most difficult instances of Table 1 for each strategy.




■ **Figure 8** Insight into the 0.5<sup>th</sup> percentile of most difficult instances of Table 2 for each strategy.





# Non-Simultaneity as a Design Constraint

**Jean Guyomarc'h** 

Krono-Safe, Massy, France

Université Paris-Saclay, CNRS,

Systèmes et Applications des Technologies de l'Information et de l'Energie, Orsay, France

jean.guyomarch@krono-safe.com

**François Guerret**

Krono-Safe, Massy, France

francois.guerret@krono-safe.com

**Bilal El Mejjati**

Krono-Safe, Massy, France

bilal.elmejjati@krono-safe.com

**Emmanuel Ohayon**

Krono-Safe, Massy, France

emmanuel.ohayon@krono-safe.com

**Bastien Vincke**

Université Paris-Saclay, CNRS,

Systèmes et Applications des Technologies de l'Information et de l'Energie, Orsay, France

bastien.vincke@universite-paris-saclay.fr

**Alain Mérigot**

Université Paris-Saclay, CNRS,

Systèmes et Applications des Technologies de l'Information et de l'Energie, Orsay, France

alain.merigot@universite-paris-saclay.fr

---

## Abstract

Whether one or multiple hardware execution units are activated (*i.e.* CPU cores), invalid resource sharing, notably due to simultaneous accesses, proves to be problematic as it can yield to unexpected runtime behaviors with negative implications such as security or safety issues. The growing interest for off-the-shelf multi-core architectures in sensitive applications motivates the need for safe resources sharing. If critical sections are a well-known solution from imperative and non-temporized programming models, they fail to provide safety guarantees. By leveraging the time-triggered programming model, this paper aims at enforcing that identified critical windows of computations can never be simultaneously executed. We achieve this result by determining, before an application is compiled, the exact dates during which a task accesses a shared resource, which enables the off-line validation of non-simultaneity constraints.

**2012 ACM Subject Classification** Theory of computation → Models of computation

**Keywords and phrases** Temporal reasoning, Temporal constraints, Specification and verification of systems

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.13

**Supplementary Material** The implementation of algorithms is available at <https://github.com/krono-safe/mcti-detect/>

**Funding** This research is supported by the company Krono-Safe.

**Acknowledgements** We would like to thank Fabien Siron, Matthieu Texier for their interesting and constructive discussions and other engineers from Krono-Safe who helped contributing to this paper. We would also like to thank the anonymous reviewers for their feedback and suggestions.



© Jean Guyomarc'h, François Guerret, Bilal El Mejjati, Emmanuel Ohayon, Bastien Vincke, and Alain Mérigot;

licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 13; pp. 13:1–13:15



Leibniz International Proceedings in Informatics

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

## 1 Introduction

Resources sharing is a topic of particular interest, notably in safety-critical real-time research, which is challenging for multi-core architectures. These systems are usually bound to stringent timing constraints: failure to perform a computation within a well-specified time interval contributes to the system failure [16]. As failure is not an option, industrials usually rely on a strategy of time provisioning, where predefined slices of time are dedicated to computations, with an additional safety margin. For example, this concept is described as a system of time frames in the ARINC-653P1 specification, which is used in the avionic industry [11]. Determining a strict upper bound of the computation time is known as the Worst-Case Execution Time (WCET) problem: the execution times of a sequence of computations may vary between multiple runs. This variability of execution times is caused by multiple factors, such as the hardware implementation [30, 13], the physical environment in which the hardware operates or the implementation of the software and the interactions software-hardware [32].

To reduce the development and production costs of their systems, as well as the time-to-market, industrials usually rely on commercially available Components Off-The-Shelf (COTS) instead of designing and manufacturing their own hardware [6]. Hardware COTS are produced by a different industry that targets a wider audience. As a result, most architectures are designed in order to minimize average execution times, rather than worst-case execution times. In addition to time-interferences induced by a single core, simultaneous accesses to a same hardware resource (*e.g.* the shared memory or a peripheral) made by multiple cores causes the hardware to arbitrate these concurrent accesses and to serialize them, effectively introducing additional time-interferences [31]. It is estimated that the current WCET analysis techniques would yield the WCET to be multiplied by a value close to the number of cores activated [24, 22, 8]. Such pessimistic estimates lead to over-constrained systems, wasting computing resources, causing higher development and production costs with an unnecessarily increased power consumption.

**Contributions.** This paper proposes a technique for safe multi-core systems design that is based on an offline temporal partitioning. It allows a system designer to specify windows of computations that shall never be executed simultaneously. Such property would be of great importance for safety-critical avionics systems [1, 29]. After reviewing related work in Section 2, we detail the model of computation our work is based on in Section 3. We then improve this model in Section 4 to express *simultaneity*, and in Section 5 we devise state-of-the-art algorithms to verify that non-simultaneity constraints always hold. An illustrative proof-of-concept is then provided in Section 6 before we conclude in Section 7.

## 2 Related Work

As summarized in [28], resources sharing can either be limited or avoided by design to ensure the absence of interferences, or controlled during the execution of the system through dedicated services. We advocate for the first proposition, however other interesting research has been conducted in different directions and are worth mentioning.

### 2.1 Hardware Design

In this paper, we focus only on off-the-shelf processors because they are intensively used by industrials. However, it should be noted that hardware solutions have been devised, notably with PRET machines [13] or the MERASA project [30], with the goal to design specific

hardware that are better suited towards time-sensitive applications. For example, Reineke et al. [26] have designed a DRAM controller that aims at eliminating contention for shared resources.

## 2.2 Runtime Mitigations

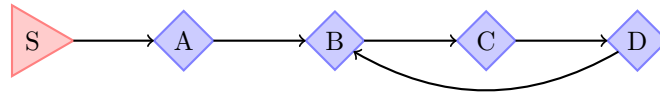
Mancuso et al. [20] have proposed the Single Core Equivalence framework, that can be applied on COTS platforms to partition shared resources and, as a result dynamically provide isolation between the different cores. To achieve this goal, the authors rely mainly on three techniques: colored cache lockdown [19], MemGuard [34] and PALLOCC [33]. These have been implemented on a Linux kernel and are well suited for dynamic systems by assigning portions of cache to tasks, regulating memory bandwidth and allocating memory pages based on the affinity of DRAM banks with tasks. Bak et al. [5] build on the PREM model of execution [23] by taking advantage of predictable intervals that distinguish memory and execution phases. Memory phases are dedicated to access shared memories, while execution phases shall not (by contract) access these. This allows to dynamically schedule tasks so two memory phases do not execute simultaneously, effectively removing sources of inter-core interferences. If these approaches effectively contribute to improving resources sharing, they do not provide strict design guarantees, because the resolution of resources sharing is determined at runtime.

## 2.3 Time-Division Multiplexing

Time-Division Multiplexing (TDM) has been extensively studied because of its inherent predictability and improved composability [16, 4]. Because immutable time slices are statically reserved in TDM, this time-division scheme presents the downside to cause underutilization of resources [14]. This is however a useful safety guarantee for safety-critical systems, because it offers greater failure detection capabilities [15].

TDM are enforced at run-time by an *execution model*, which usually consists in a tasks scheduler based on a source of time. Because they are difficult to build by hand, multiple solutions have been devised to generate them. Boniol et al. [9] propose an approach in which they instantiate a scheduling plan in which time slices are dedicated either to access the shared memory or to execute code that does not use shared resources. Their system is generated from a model of the hardware and a static analysis of WCET. Similar works have been conducted by Becker et al. [7].

David et al. [12], Chabrol et al. [10] and Lemerre et al. [18] rely on a model of computation that can be instantiated to express temporal constraints. From instances of this model of computation, data configuring an execution model are produced. This execution model ensures that the specified temporal constraints are enforced at run-time. This model has been formalized as a *time-constrained automata* [17]. It has also been explicitly used by Jan et al. [15] to automatically generate a TDM scheme allowing the control of a real-time network bus from communication specifications that were expressed in the model of computation. Our contribution follows the same path, by improving their model of computation with *non-simultaneity* semantics; effectively enabling to design critical sections driven by the time-triggered paradigm. It differs from critical sections used in imperative and non-temporized programming models [25] in that the dates at which each critical section start and end are precisely known at compile-time, offering additional safety properties, such as the guaranteed absence of deadlocks.



■ **Figure 1** Example of a trivial time-constrained automaton that describes a periodic behavior. After starting at node  $S$ , the node  $A$  is accepted. Then the sequence of nodes  $B$ ,  $C$  and  $D$  is periodically repeated.

### 3 Time-Constrained Automata

The model proposed in this paper is based on the model of computation formalized by Lemerre et al.: *time-constrained automata* [17]. We extend it later in Section 4.2, but we start by explaining briefly its foundations. This formalism defines a block as a sequence of computations that are time-bounded by at least one of the following constraints:

- *after* that indicates that a block may only start from a given date; and
- *before* that indicates that a block must end before a given date.

They respectively define the *earliest start date* and *deadline* of a batch of blocks, with homogeneous time units. Such automata are formalized as directed graphs, where arcs represent the blocks and nodes represent the temporal constraints that are applied to the arcs joining them (hence constraining the blocks). A node may carry both constraints, but only one constraint for each type. Therefore, three types of time-constrained node exist. They can either be a representation of:

- a single *after* constraint, denoted by  $\blacktriangleright$ , which can be seen in Figure 1 as the node  $S$ .
- a single *before* constraint, usually denoted by  $\blacktriangleleft$ , but not represented in this paper as it is never used as the sole constraint of a node;
- both a *before* and an *after* constraints, denoted by  $\blacklozenge$ , which can be seen in Figure 1 as the nodes  $A$ ,  $B$ ,  $C$  and  $D$ . This particular node is named *synchronization*.

► **Definition 1** (Trivial time-constrained automaton). *A time-constrained automaton is trivial if and only if every node of the automaton has exactly one output arc. Otherwise, the automaton is said non-trivial.*

There exist several graph simplification techniques that allow to detect impossible graphs or to remove redundant constraints. They are formally defined in the original paper, and we only assume their existence and that graphs can possibly be re-written to a simpler form or proved impossible. In the following of this paper, we assume that all time-constrained automata are valid and reduced to their most simplified form.

An interesting application of time-constrained automata is the ability to derive execution models (*i.e.* scheduling schemes) that preserves the temporal constraints that bound computation blocks. The ability to transform a mathematical model to a concrete result that can be embedded on a hardware target asserted our choice to build on top of this model. The authors of the original paper designed and implemented a variation of the EDF (Earliest Deadline First) algorithm, called *EDF-dyn*, which has been proved optimal for time-constrained sequences of blocks on single processors. However, our approach is not limited to one specific scheduling algorithm, since verification algorithms are applied on the model of computation, and not on the model of execution.

## 4 System Model

The model of computation we propose is based on time-constrained automata described in Section 3. We insist on the separation of *model of computation* that embodies the design space and the *model of execution* that embodies the run-time of the designed application on a specific execution platform (*e.g.* an embedded COTS system).

### 4.1 Non-Simultaneity as a Design Constraint

In this paper, we define the *simultaneity* as applied to windows of computations that execute within a known and bounded time span. Simultaneity between two windows of computations describes that their execution may overlap in time.

In Section 2.3, it has been shown that scheduling plans implementing critical sections driven by the time-triggered paradigm can be generated from constraints deduced from characteristics of the system. In approaches that do not rely on a model of computation, there is no guarantee that a feasible schedule exists, because simultaneity is yet another parameter involved in scheduling algorithms. In such cases, it is necessary to tweak multiple parameters of the scheduling algorithm to hope for a viable solution to be found. This process is not guaranteed to converge towards a solution.

Considering a *model of computation* during the design phase that is implemented by a *model of execution* allows to divide the global scheduling problem into independent ones. As the model of computation deals with temporal constraints, simultaneity can be verified regardless of the actual execution times of the tasks. If the application does not respect these new design constraints, then only the original design has to be modified. On the contrary, if such errors were detected later, fixing them would jeopardize the whole application: both its design and implementation.

To the best of our knowledge, there exist no methodology in time-triggered resource sharing that allows to model *simultaneity* as an explicit design constraint integrated to a model of computation. We think that addressing this early in the design phase contributes to safer and more robust multi-core applications.

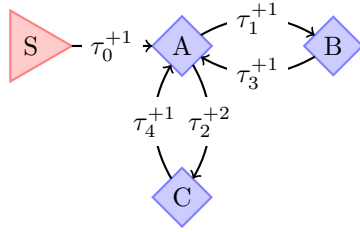
### 4.2 Augmenting Time-Constrained Automata

This paper claims to add a new semantic to time-constrained automata, which is detailed in this section.

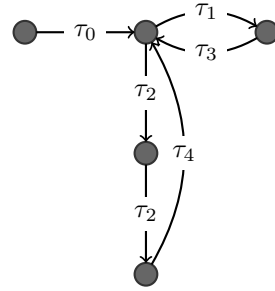
**Temporal transitions.** Let a *clock* be a structure that causes the global time to advance; a time-constrained automaton is bound to exactly one clock. We define a *temporal transition* as the ordered set of blocks encompassed within exactly one *after* and one *before* constraints. It is associated with the time span of the computations, which corresponds to the time difference between the deadline (carried by the before constraint) and the earliest start date (carried by the after constraint). This time span, denoted by  $t$  may only be strictly positive and is expressed as a finite number of clock ticks. As such, a temporal transition is formally written as the time interval  $\tau^{+t}$ . The time span can be omitted for brevity; in this case a temporal transition is only denoted by its name (*e.g.*  $\tau$ ).

**Isochronous Time-Constrained Automata.** Let us consider time-constrained automata where every sequence of blocks is bounded by exactly one *after* and one *before* constraints. They are composed of an *entry* node and a connected graph of *synchronization nodes*, in

## 13:6 Non-Simultaneity as a Design Constraint



(a) Non-trivial time-constrained automata with each temporal transition is bounded by a before and an after node.



(b) Isochronous equivalent of the automaton in Figure 2a.

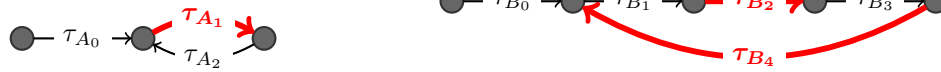
■ **Figure 2** Representations of a non-trivial time-constrained automaton (Figure 2a) and its isochronous equivalent (Figure 2b). From the start node  $S$ , only one temporal transition is allowed:  $\tau_0$ , which is performed in one clock tick. After  $A$  is reached, either  $B$  or  $C$  is reachable, respectively through  $\tau_1$  in one tick and  $\tau_2$  in two ticks.  $A$  is then activated from either  $B$  or  $C$  in one tick through either  $\tau_3$  or  $\tau_4$ , depending on the previous transition. This behavior is then infinitely repeated.

which each node has at least one output arc. As a result, there exists at least one cycle in this graph. The entry node is an *after* node, which represents the unique entry point of the automaton. It is connected to the graph of synchronization nodes by at least one output arc, and it accepts no input arc. Such automata can be made *isochronous* by splitting each temporal transition into a sequence of successive transitions of unitary length, such that the sum of lengths of the resulting transitions equals the time span of the original transition. In the underlying graphical representations, these additional nodes are denoted by  $\bullet$ . We define such automata as *isochronous time-constrained automata*. Figure 2 illustrates how the non-trivial time-constrained automaton with labeled temporal transitions shown in Figure 2a can be represented as an isochronous time-constrained automaton in Figure 2b.

**Time-Constrained Applications.** A *time-constrained application* is defined as a fixed set of isochronous time-constrained automata that share a same unique base clock. More specifically, at each clock tick a new temporal transition is simultaneously completed by all the automata that compose the application: because they share the same clock, they are *synchronous*. A software implementation of time-constrained applications is required to implement *bound multi-processing*: each task described by an isochronous time-constrained automaton must be statically assigned to one execution unit (*i.e.* a CPU core).

An application is associated with a set of *exclusion groups*, an exclusion group being a fixed set of temporal transitions that shall not overlap in time. These are specified by the designer of the application after a preliminary analysis. The property that temporal transitions of a given exclusion group do not overlap in time is a *safety property* (“bad things do not happen during execution of a program” [2]). For a given exclusion group, this property must be verified on the result of the composition of every automata that has at least one temporal transition belonging to this exclusion group.

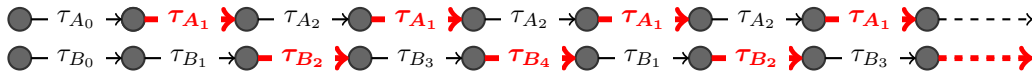
Exclusion groups model the *non-simultaneity* within a system. When part of a set, they translate the requirement that the simultaneous execution of their associated windows of computation is *forbidden*.



(a) Automaton  $A$ , allocated to core  $c_A$ , which describes a periodic behavior: after  $\tau_{A_0}$  has been taken, the sequence  $\tau_{A_1}$  and  $\tau_{A_2}$  is repeated.

(b) Automaton  $B$ , allocated to core  $c_B$ , which describes a periodic behavior: after  $\tau_{B_0}$  has been taken, the sequence  $\tau_{B_1}$ ,  $\tau_{B_2}$ ,  $\tau_{B_3}$  and  $\tau_{B_4}$  is repeated.

■ **Figure 3** Example of time-constrained application composed of two trivial isochronous time-constrained automata  $A$  and  $B$  respectively allocated to cores  $c_A$  and  $c_B$  such that  $c_A \neq c_B$ . The temporal transitions  $\tau_{A_1}$ ,  $\tau_{B_2}$  and  $\tau_{B_4}$  shall not overlap in time.



■ **Figure 4** Infinite “unfolding” of automata  $A$  (above) and  $B$  (below). It hints towards a periodic pattern where temporal transition in the exclusion group  $G = \{\tau_{A_1}, \tau_{B_2}, \tau_{B_4}\}$  cannot overlap in time, because of the temporal specification of  $A$  and  $B$ .

### 4.3 Example

Figure 3 shows an example of a simple time-constrained application that consists in two trivial time-constrained automata  $A$  and  $B$  that are allocated to two different CPU cores. Each automaton defines its own set of temporal transitions:  $\tau_{A_0}$ ,  $\tau_{A_1}$  and  $\tau_{A_2}$  for  $A$  and  $\tau_{B_0}$ ,  $\tau_{B_1}$ ,  $\tau_{B_2}$ ,  $\tau_{B_3}$  and  $\tau_{B_4}$  for  $B$ . One exclusion group is arbitrarily defined here:  $G = \{\tau_{A_1}, \tau_{B_2}, \tau_{B_4}\}$ : these temporal transitions shall not overlap in time.

In this example, the temporal design of automata  $A$  and  $B$  allows for the exclusion group  $G$  to hold: since isochronous time-constrained automata within a time-constrained application are synchronous and since temporal transitions are isochronous, one can observe that when  $A$  runs  $\tau_{A_1}$ ,  $B$  simultaneously runs either  $\tau_{B_1}$  or  $\tau_{B_3}$ , but never  $\tau_{B_2}$  nor  $\tau_{B_4}$ . This is illustrated by Figure 4, which shows that “unfolding”  $A$  and  $B$  hints towards thinking that temporal transitions listed in the exclusion group  $G$  cannot overlap in time. In the next section, we show how this problem can be automatically verified.

## 5 Validating the simultaneity constraints

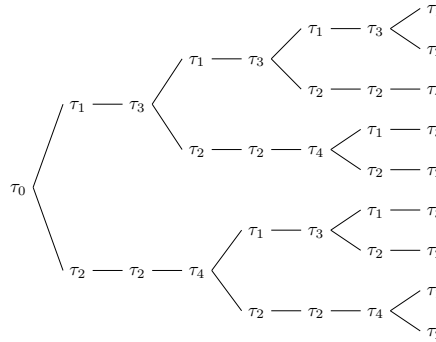
We have introduced in Section 4 the notions of *time-constrained applications* and of *exclusion groups*, that specify the property that the temporal transitions they contain must not execute simultaneously. In this section, we propose algorithms that verify this property.

### 5.1 Formalization of the problem

Time-constrained automata may exhibit an infinite possibility of temporal behaviors, because a task embodying the software implementation of an automaton virtually does not have an upper bound of running time. The dates at which a transition can be activated may result from all the infinite possible sequences of these cycles. As an illustration of this complexity, Figure 5 shows all the possible temporal behaviors of the time-constrained automaton shown in Figure 2b between clock ticks zero and seven.

Because a time-constrained application is composed of isochronous time-constrained automata and because they all share the same clock, they are also *synchronous*. As a result, each clock tick causes a temporal transition to be activated in each automata. This implies





■ **Figure 5** Tree that results from the “unfolding” of temporal behaviors of the non-trivial time-constrained automaton shown in Figure 2b after seven clock ticks. The transition  $\tau_0$  is activated at date zero and each arc represents the occurrence of a clock tick.

that a temporal transition can be activated for a possibly infinite set of dates, where a date is represented by a natural number. For example, in Figure 3a,  $\tau_{A_0}$  can only be activated at date zero, whereas  $\tau_{A_1}$  can be activated for all dates that are odd. An isochronous time-constrained automaton can therefore be understood as a finite automaton, where:

- each state but the initial one can be marked as accepting;
- the increment of time, associated to all the temporal transitions can be seen as the symbol of a unary alphabet (isochronous property);
- the set of dates at which a state can be reached is given by set of the lengths of the words that lead to this state. Note that this set may be infinite, if the state is included in a cycle.

The set of dates at which a state can be reached can therefore be expressed as the regular language over a unary alphabet accepted by the automaton where only this state is marked as accepting. It is known that each regular unary language can be represented as the union of a finite number of arithmetic progressions of the form  $\{c + dk | k \in \mathbb{N}\}$  where  $c$  and  $d$  are positive constants specifying their offset and period [27]. They can also be written as the pair  $(c, d)$ .

Temporal transitions that originate from a state are reachable at this set of dates. Therefore, the set of dates at which a temporal transition can be activated is the union of set of dates at which their respective states are reachable.

## 5.2 Determination of dates of reachability for every transitions

**Notations.** Let a unary, non-deterministic finite automaton (UNFA)  $\mathcal{A}$  with  $n \geq 2$  states and  $m$  transitions, such that  $\mathcal{A} = (Q, \delta, I, F)$  where  $Q$  is the finite set of states ( $|Q| = n$ ),  $\delta \subseteq Q \times Q$  is a transition relation,  $I \subseteq Q$  is the set of initial states of the automaton and  $F \subseteq Q$  is the set of *accepting states*. Using the notations defined in [27],  $q \xrightarrow{x} q'$  denotes that there exist a path of length  $x$  from  $q \in Q$  to  $q' \in Q$ . On a UNFA, a path of length  $x$  can be seen as a word  $x$ ; as such, a word of length  $x$  is accepted by  $\mathcal{A}$  if there exists a path of length  $x$  from  $q_i \in I$  to  $q_f \in F$ , and the language  $\mathcal{L}(\mathcal{A})$  accepted by  $\mathcal{A}$  is the set of all the words accepted by  $\mathcal{A}$ .

**Expressing  $\mathcal{L}(\mathcal{A})$ .** Sawa proposes in [27] the algorithm *UNFA-Arith-Progressions* that processes a UNFA  $\mathcal{A}$  to construct a finite set of arithmetic progressions  $\mathcal{R}$  describing the language  $\mathcal{L}(\mathcal{A})$ , with a space complexity in  $O(n + m)$  and a time complexity in  $O(n^2(n + m))$ .



Applied to isochronous time-constrained automata, the result of this algorithm consists in the exhaustive set of dates at which a given state can be reached. The essence of the algorithm relies on expressing a path  $\alpha$  from  $q_i \in I$  to  $q_f \in F$  via  $q \in Q$  so that  $q_i \xrightarrow{c_1} q \xrightarrow{c_2} q_f$ . If  $q$  belongs to a cycle of length  $d$ , then the length of  $\alpha$  can be expressed as the pair  $(c_1 + c_2, d)$ ; otherwise it is simply  $(c_1 + c_2, 0)$ . As such,  $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$  with  $|\mathcal{R}_1| \leq n^2$  and  $|\mathcal{R}_2| \leq n$ .  $\mathcal{R}_1$  contains every word of length  $x < n^2$  written  $(x, 0)$  whereas  $\mathcal{R}_2$  contains all the other words of  $\mathcal{L}(\mathcal{A})$  (with  $x \geq n^2$ ) expressed as arithmetic progressions (at most  $n$ ).

**Tailoring the algorithm.** Running the algorithm unmodified for each of the  $n - 1$  states that can be marked as accepting<sup>1</sup> would yield a total time complexity of  $O(n^3(n + m))$ . We propose a modified version of this algorithm to specifically determine the set of reachability dates of temporal transitions without degrading the time complexity:

- For  $q \in Q$ , the value  $sl(q)$  is defined as the length of the shortest loop that can be done in  $q$ . If  $q$  is not part of a loop, then  $sl(q)$  is undefined.
- A state  $q$  is called *important* if  $q$  belongs to a nontrivial strongly connected component  $C$  (implying that  $sl(q)$  is defined) and the value  $sl(q)$  is minimal for all states in  $C$ .
- The sets  $S_i$  are computed so each set contains all states reachable from the initial state  $s$  by  $i$  steps:  $S_i = \{q \in Q : s \xrightarrow{i} q\}$  for  $i \in [0, n^2)$ .
- Let  $Imp$  the set of important states of  $\mathcal{A}$ .
- Let  $Q_{imp} = S_{n-1} \cap Imp$  the important states that can be reached after exactly  $n - 1$  steps from the initial state  $s$ .
- Let  $D = \{sl(q) | q \in Q_{imp}\}$  the set of the shortest loop lengths among the important states in  $Q_{imp}$ .
- Since there is only one initial state  $s$  to isochronous time-constrained automata,  $I$  can be written as  $I = \{s\}$ .
- We re-define  $F$  as the set of states that can be marked as accepting. By definition,  $F = Q \setminus \{s\}$ .
- We define  $\mathbb{T}_q$  the set of temporal transitions that can be activated at state  $q$ , that is the outgoing vertices.

From the definition of isochronous time-constrained automata, we can propose a new formulation of the set  $\mathcal{R}_1$ , such that  $\mathcal{R}_1 = \{(i, 0) | i \in [1, n^2)\}$ . This allows to build a first set of dates at which states are reachable. In this case, we can re-use this formula to determine an initial set of dates for each temporal transition  $\mathcal{D}_{1,\tau}$  as shown in Algorithm 1. Because the original formula excludes the initial state, we add that the transitions reachable from the initial state are all reachable at date zero (by definition). We just associate the temporal transitions activated at a state  $q$  with the date at which  $q$  is reached. This is possible because each state is associated with a date.

The second set of dates  $\mathcal{R}_2$  is built around the sets  $T_i$  that contain all states from which some final state can be reached by  $i$  steps. They are defined as in Equation (1). Then the pair  $(c' + n - 1, d)$  is added to  $\mathcal{R}_2$  for  $c' \in [n^2 - 2n, n^2 - n - 1]$  and each  $d \in D$  such that  $c' \geq n^2 - n - d$ , if there exists some  $q \in Q_{imp}$  with  $sl(q) = d$  such that  $q \in T_{c'}$ .

$$T_i = \{q \in Q | \exists q_f \in F : q \xrightarrow{i} q_f\} \text{ for } i \in [0, n^2 - n - 1] \quad (1)$$

A consequence of this formulation in the original algorithm is that the different temporal transitions leading to  $q_f \in F$  are entangled in the construction of the sets  $T_i$  in Equation (1).

<sup>1</sup> the initial state cannot be reached from another state

## 13:10 Non-Simultaneity as a Design Constraint

■ **Algorithm 1** Construction of the first set of dates  $\mathcal{D}_{1,\tau}$  that contains dates at which each temporal transition  $\tau$  is activated.

---

```

for  $\tau \in \mathbb{T}_s$  do
   $\mathcal{D}_{1,\tau} = \{(0, 0)\}$ 
for  $i \in [1, n^2)$  do
  for  $q \in S_i$  do
    for  $\tau \in \mathbb{T}_q$  do
       $\mathcal{D}_{1,\tau} = \mathcal{D}_{1,\tau} \cup \{(i, 0)\}$ 

```

---

To preserve dates specific to temporal transitions, we can instead propose the creation of the sets that discriminate temporal transitions, as written in Equation (2).

$$T_{i,q_f} = \{q \in Q : q \xrightarrow{i} q_f\} \text{ for } i \in [0, n^2 - n - 1] \text{ and } q_f \in F \quad (2)$$

Considering each  $q_f \in F$ , and each  $\tau \in \mathbb{T}_{q_f}$ , the same algorithm can be re-used to compute  $\mathcal{D}_{2,\tau}$  as in Algorithm 2 by substituting  $T_i$  with  $T_{i,q_f}$ . Finally the set of dates  $\mathcal{D}_\tau$  for which each temporal transition  $\tau$  is activated can be computed as  $\mathcal{D}_\tau = \mathcal{D}_{1,\tau} \cup \mathcal{D}_{2,\tau}$ .

Instead of running the original algorithm for each of the  $n - 1$  accepting states, we build the sets  $T_{i,q_f}$  once. Furthermore, constructing the sets  $T_{i,q_f}$  requires the same operations than the sets  $T_i$  as only data organization changes. Thus, we can preserve the overall time complexity of the original algorithm ( $O(n^2(n + m))$ ) since the application of Algorithm 2 does not increase it.

■ **Algorithm 2** Construction of the second set of dates  $\mathcal{D}_{2,\tau}$  that contains dates at which each temporal transition  $\tau$  is activated.

---

```

for  $q \in Q_{imp}$  do
  for  $c' \in [n^2 - 2n, n^2 - n - 1]$  do
    for  $q_f \in F$  do
      if  $q \in T_{c',q_f}$  and  $c' \geq n^2 - n - sl(q)$  then
        for  $\tau \in \mathbb{T}_{q_f}$  do
           $\mathcal{D}_{2,\tau} = \mathcal{D}_{2,\tau} \cup \{(c' + n - 1, sl(q))\}$ 

```

---

**Special case of trivial time-constrained automata.** The structure of trivial time-constrained automata allows major simplifications of this algorithm. The dates at which a state can be reached can be written as a single arithmetic progression. If we consider the graph representation of the automaton, nodes that are not part of a cycle can be written as  $(i, 0)$  where  $i$  can be trivially found by exploring the automaton until the node is reached. Nodes that are part of a cycle can be written as  $(c, d)$  where  $c$  is the first date at which the node is reached and  $d$  is the length of the loop. For example, in Figure 3a, dates at which the temporal transition  $\tau_{A_1}$  is activated are  $\{1 + 2k | k \in \mathbb{N}\}$ . Similarly, they are  $\{2 + 2k | k \in \mathbb{N}\}$  for  $\tau_{A_2}$  and  $\{0\}$  for  $\tau_{A_0}$ .

### 5.3 Intersection of dates

We have shown earlier how to compute the set of dates  $\mathcal{D}_\tau$  at which each temporal transition  $\tau$  is activated. This set can be defined as a union of:

- singletons; and
- arithmetic progressions expressed as  $\{c + dk | k \in \mathbb{N}\}$ .

For a given exclusion group  $G$ , verifying that the intersection of the dates that characterize temporal transitions belonging to different automata is empty is equivalent to verify the safety property that temporal transitions within  $G$  cannot overlap in time, and as a result cannot be executed simultaneously. We now show the conditions that apply on two dates  $D_a$  and  $D_b$  for them to intersect. Different techniques may be used depending on whether  $D_a$  and  $D_b$  represent constants or arithmetic progressions.

**Intersection of dates for two constants:** let  $D_a = h_a$  and  $D_b = h_b$  two constant dates. In this trivial case, they share a date in common if and only if  $h_a = h_b$ .

**Intersection of dates for arithmetic progressions:** let  $D_a = \{c_a + d_a k | k \in \mathbb{N}\}$  and  $D_b = \{c_b + d_b k | k \in \mathbb{N}\}$  two arithmetic progressions. Their intersection is not empty if and only if the following linear diophantine equation has a solution:  $\alpha x + \beta y = \gamma$  for  $x \in \mathbb{Z}$  and  $y \in \mathbb{Z}$ , with  $\alpha = d_a$ ,  $\beta = -d_b$  and  $\gamma = c_b - c_a$ . Linear diophantine equations are well-known structures that have been extensively studied; the problem of testing the existence of solutions as well as finding them has long been solved [3]. This linear diophantine equation admits a solution in  $\mathbb{Z}^2$  if and only if the greatest common denominator of  $\alpha$  and  $\beta$  divides  $\gamma$ . If this equation has no solution then the intersection of  $D_a$  and  $D_b$  is empty. Otherwise, if there exists a solution in  $\mathbb{Z}^2$  then  $D_a$  and  $D_b$  have in common an infinite set of dates since for any solution  $(x_0, y_0)$  of the equation, the set of solutions  $\{(x_0 + d_b k, y_0 + d_a k) | k \in \mathbb{Z}\}$  can always be built (this set of solutions in  $\mathbb{Z}^2$  contains an infinite number of pairs where both members are natural integers).

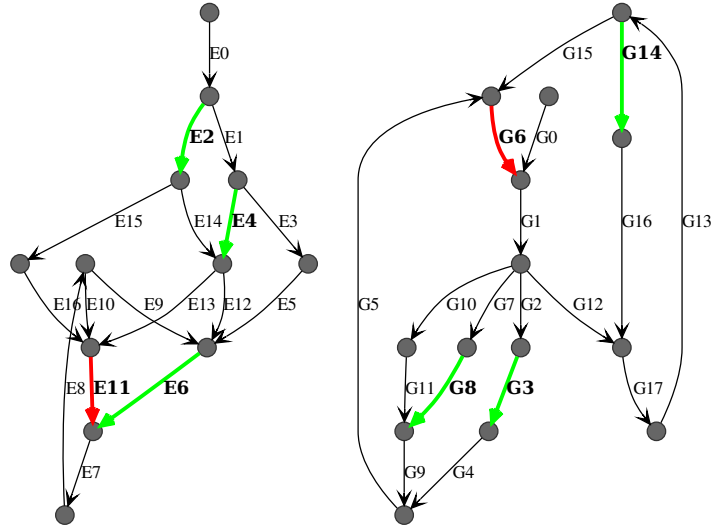
**Intersection of dates for a constant and an arithmetic progression:** if  $D_a = \{h_a\}$  is a singleton and  $D_b = \{c_b + d_b k | k \in \mathbb{N}\}$  is an infinite set representing an arithmetic progression, they may intersect at most once, if  $D_a \subset D_b$ , that is when they exist  $x \in \mathbb{N}$  such that  $h_a = c_b + d_b x$ . We find that this is true when  $h_a \geq c_b$  and  $d_b$  divides  $h_a - c_b$ .

## 6 Proof of Concept

**Implementation and Reproducibility.** The model defined in Section 4 has been integrated to the ASTERIOS suite, developed by the Krono-Safe company. It relies on a programming model detailed by Methni et al. in [21] to instantiate a model of computation, in which support for simultaneity has been added. The algorithms presented in Section 5 and the method to validate the intersection of dates have been implemented in a standalone executable that has been open-sourced<sup>2</sup> under the Apache-2.0 license. It takes as inputs a specification of the different tasks that compose an application with the list of exclusion groups to be checked, and generates a report containing the dates at which each temporal transition can be activated, as well as a graphical representation of the time-constrained application and either the validation of exclusion groups or a counter-example. The proof-of-concept in this section is based on the open-source version.

<sup>2</sup> <https://github.com/krono-safe/mcti-detect/>

## 13:12 Non-Simultaneity as a Design Constraint



■ **Figure 6** Design in which  $E_2, E_4, E_6, E_{11}, G_3, G_6, G_8$  and  $G_{14}$  are used to access the shared resource. However it is found that  $E_{11}$  and  $G_6$  can be simultaneously reached at the same date. This is therefore an example of a design that does not guarantee safe resources sharing.

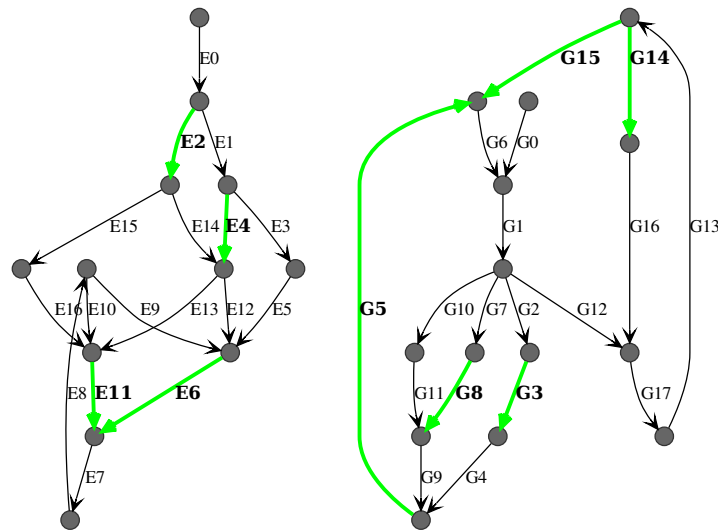
■ **Table 1** Counter-example showing traces leading for  $E_{11}$  and  $G_6$  to overlap in time at tick 12.

Tick	0	1	2	3	4	5	6	7	8	9	10	11	12
Task $E$	$E_0$	$E_1$	$E_4$	$E_{12}$	$E_6$	$E_7$	$E_8$	$E_{10}$	$E_{11}$	$E_7$	$E_8$	$E_{10}$	$E_{11}$
Task $G$	$G_0$	$G_1$	$G_{10}$	$G_{11}$	$G_9$	$G_5$	$G_6$	$G_1$	$G_{12}$	$G_{17}$	$G_{13}$	$G_{15}$	$G_6$

**Sharing resources between two parallel tasks.** For this illustrative proof-of-concept, let's consider a simple application that uses two non-trivial tasks  $E$  and  $G$ , each implanted on a different CPU core. The requirements of this application impose they exchange data through a shared resource (*e.g.* shared memory). In this specific use case, we assume the temporal constraints are fixed: nodes cannot be added nor removed. When considering an incremental design, this may not be the case. The end goal is to guarantee that accesses to the shared resources are performed during temporal transitions that never overlap in time. The occurrence of unwanted simultaneous accesses may result in data corruption (*e.g.* the two tasks write at the same memory address) or in increased execution times caused by additional contention.

**Exposing an invalid design.** A first design can be seen in Figure 6, which represents a time-constrained application composed of two non-trivial tasks where accesses to the shared resource are performed during the temporal transitions  $E_2, E_4, E_6, E_{11}, G_3, G_6, G_8$  and  $G_{14}$ . However, we find that temporal transitions  $E_{11}$  and  $G_6$  may overlap in time, as shown in Table 1. This small example showcases that checking for absence of simultaneity is not a trivial process and highlights the importance of automated validation.

**Towards a safe design.** If the design in Figure 6 does not guarantee safe resources sharing, it is possible to try other design candidates. If the functional requirements of the application allow it, the shared resource could be accessed from  $G_5$  and  $G_{15}$  and the retrieved data made available to  $G_6$ . This modified design is checked as in Figure 7 and the new set of temporal



■ **Figure 7** Design in which  $E_2$ ,  $E_4$ ,  $E_6$ ,  $E_{11}$ ,  $G_3$ ,  $G_5$ ,  $G_8$ ,  $G_{14}$  and  $G_{15}$  are used to access the shared resource. It is found that they never overlap in time. This is therefore an example of a design that guarantees safe resources sharing, given that accesses to the shared resource happen only during these temporal transitions.

transitions that access the shared resource ( $E_2$ ,  $E_4$ ,  $E_6$ ,  $E_{11}$ ,  $G_3$ ,  $G_5$ ,  $G_8$ ,  $G_{14}$  and  $G_{15}$ ) have been found to never overlap in time. Implementing such design removes entire classes of problems that could comprise data integrity or negatively impact execution times, while allowing for a better use of overall computing resources.

## 7 Conclusion and Perspectives

We have presented a model of computation based on time-constrained automata, that can be used to express *non-simultaneity* as a design constraint in a model of computation. This allows to express a safety property over parallel systems, which, if verified, ensures that litigious sequences of computations can never run simultaneously. Designing such systems with non-simultaneity as a constraint from the ground-up is believed to bring significant safety benefits, notably for safety-critical real-time systems. We have then shown that this safety property could be automatically verified, with reasonable complexity, by standalone and open-sourced algorithms that extend the state of the art. As for future work, it would be interesting to propose more advanced techniques to help the designer to interactively explore the traces leading to a violation of its design constraints, for a more efficient convergence towards a safe design. It seems also important to explore techniques to determine the sources of time-interferences when they occur.

### References

- 1 Irune Agirre, Jaume Abella, Mikel Azkarate-Askasua, and Francisco J Cazorla. On the tailoring of cast-32a certification guidance to real cots multicore architectures. In *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8. IEEE, 2017. doi:10.1109/SIES.2017.7993376.

## 13:14 Non-Simultaneity as a Design Constraint

- 2 Bowen Alpern and Fred B Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987. doi:10.1007/BF01782772.
- 3 George E Andrews. *Number theory*. Courier Corporation, 1994.
- 4 Christophe Aussagues, Damien Chabrol, and Vincent David. Method for the deterministic execution and synchronisation of an information processing system comprising a plurality of processing cores executing system tasks, April 2010. Patent WO 2010/043706 A2.
- 5 Stanley Bak, Gang Yao, Rodolfo Pellizzoni, and Marco Caccamo. Memory-aware scheduling of multicore task sets for real-time systems. In *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 300–309. IEEE, 2012. doi:10.1109/RTCSA.2012.48.
- 6 Thomas G Baker. Lessons learned integrating cots into systems. In *International Conference on COTS-Based Software Systems*, pages 21–30. Springer, 2002. doi:10.1007/3-540-45588-4\_3.
- 7 Matthias Becker, Dakshina Dasari, Borislav Nolic, Benny Akesson, Vincent Nélis, and Thomas Nolte. Contention-free execution of automotive applications on a clustered many-core platform. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 14–24. IEEE, 2016. doi:10.1109/ECRTS.2016.14.
- 8 Jingyi Bin, Sylvain Girbal, Daniel Gracia Pérez, Arnaud Grasset, and Alain Mérigot. Studying co-running avionic real-time applications on multi-core COTS architectures. In *Embedded Real Time Software and Systems (ERTS2014)*, Toulouse, France, February 2014.
- 9 Frédéric Boniol, Hugues Cassé, Eric Noulard, and Claire Pagetti. Deterministic execution model on cots hardware. In *International Conference on Architecture of Computing Systems*, pages 98–110. Springer, 2012. doi:10.1007/978-3-642-28293-5\_9.
- 10 Damien Chabrol, Vincent David, Patrice Oudin, Gilles Zeppa, and Mathieu Jan. Freedom from interference among time-triggered and angle-triggered tasks: a powertrain case study. In *Embedded Real Time Software and Systems (ERTS2014)*, Toulouse, France, February 2014.
- 11 Airlines Electronic Committee. Avionics application software standard interface - part 1: Required services. Arinc 653p1, Airlines Electronic Committee, August 2015.
- 12 Vincent David, Christophe Aussaguès, Stéphane Louise, Philippe Hilsenkopf, Bertrand Ortolò, and Christophe Hessler. The oasis based qualified display system. In *Fourth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls and Human-Machine Interface Technologies (NPIC&HMIT 2004)*, Columbus, Ohio, USA, page 11, 2004.
- 13 Stephen A Edwards and Edward A Lee. The case for the precision timed (pret) machine. In *Proceedings of the 44th annual Design Automation Conference*, pages 264–265. ACM, 2007. doi:10.1145/1278480.1278545.
- 14 Farouk Hebbache, Mathieu Jan, Florian Brandner, and Laurent Pautet. Shedding the shackles of time-division multiplexing. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 456–468. IEEE, 2018. doi:10.1109/RTSS.2018.00059.
- 15 Mathieu Jan, Jean-Sylvain Camier, and Vincent David. Scheduling safety-critical real-time bus accesses using time-constrained automata. In *RTNS*, pages 87–96. Citeseer, 2011.
- 16 Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Real-Time Systems Series. Springer, 2011. doi:10.1007/978-1-4419-8237-7.
- 17 Matthieu Lemerre, Vincent David, Christophe Aussagues, and Guy Vidal-Naquet. An introduction to time-constrained automata. In *Proceedings of the 3rd Interaction and Concurrency Experience Workshop (ICE'10)*, volume 38, pages 83–98, June 2010. doi:10.4204/EPTCS.38.9.
- 18 Matthieu Lemerre and Emmanuel Ohayon. A model of parallel deterministic real-time computation. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 273–282. IEEE, 2012. doi:10.1109/RTSS.2012.78.
- 19 Renato Mancuso, Roman Dudko, Emiliano Betti, Marco Cesati, Marco Caccamo, and Rodolfo Pellizzoni. Real-time cache management framework for multi-core architectures. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 45–54. IEEE, 2013. doi:10.1109/RTAS.2013.6531078.

- 20 Renato Mancuso, Rodolfo Pellizzoni, Marco Caccamo, Lui Sha, and Heechul Yun. Wcet (m) estimation in multi-core systems using single core equivalence. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 174–183. IEEE, 2015. doi:10.1109/ECRTS.2015.23.
- 21 Amira Methni, Emmanuel Ohayon, and François Thurieau. ASTERIOS Checker : A Verification Tool for Certifying Airborne Software. In *10th European Congress on Embedded Real Time Systems (ERTS 2020)*, Toulouse, France, January 2020. URL: <https://hal.archives-ouvertes.fr/hal-02508852>.
- 22 Jan Nowotsch and Michael Paulitsch. Leveraging multi-core computing architectures in avionics. In *2012 Ninth European Dependable Computing Conference*, pages 132–143. IEEE, 2012. doi:10.1109/EDCC.2012.27.
- 23 Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo, and Russell Kegley. A predictable execution model for cots-based embedded systems. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 269–279. IEEE, 2011. doi:10.1109/RTAS.2011.33.
- 24 Rodolfo Pellizzoni, Andreas Schranzhofer, Jian-Jia Chen, Marco Caccamo, and Lothar Thiele. Worst case delay analysis for memory interference in multicore systems. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 741–746. IEEE, 2010. doi:10.1109/DATE.2010.5456952.
- 25 Michel Raynal. *Concurrent programming: algorithms, principles, and foundations*. Springer Science, 2013. doi:10.1007/978-3-642-32027-9.
- 26 Jan Reineke, Isaac Liu, Hiren D Patel, Sungjun Kim, and Edward A Lee. Pret dram controller: Bank privatization for predictability and temporal isolation. In *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 99–108. IEEE, 2011. doi:10.1145/2039370.2039388.
- 27 Zdeněk Sawa. Efficient construction of semilinear representations of languages accepted by unary nondeterministic finite automata. *Fundamenta Informaticae*, 123(1):97–106, 2013. doi:10.3233/FI-2013-802.
- 28 Nathanaël Sensfelder, Julien Brunel, and Claire Pagetti. Modeling cache coherence to expose interference. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ECRTS.2019.18.
- 29 Certification Authorities Software Team. Multi-core processors - position paper. Cast-32a, Certification Authorities Software Team, November 2016.
- 30 Theo Ungerer, Francisco Cazorla, Pascal Sainrat, Guillem Bernat, Zlatko Petrov, Christine Rochange, Eduardo Quinones, Mike Gerdes, Marco Paolieri, Julian Wolf, et al. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 30(5):66–75, 2010. doi:10.1109/MM.2010.78.
- 31 Stephen C Vestal, Pamela Binns, Aaron Larson, Murali Rangarajan, and Ryan Roffelsen. Safe partition scheduling on multi-core processors, 2012. US Patent 8,316,368.
- 32 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008. doi:10.1145/1347375.1347389.
- 33 Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 155–166. IEEE, 2014. doi:10.1109/RTAS.2014.6925999.
- 34 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64. IEEE, 2013. doi:10.1109/RTAS.2013.6531079.





# One-Pass Context-Based Tableaux Systems for CTL and ECTL

Alex Abuin 

Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA),  
Arrasate-Mondragón Gipuzkoa, Spain  
aabuin@ikerlan.es

Alexander Bolotov 

University of Westminster, London, UK  
<https://www.westminster.ac.uk/about-us/our-people/directory/bolotov-alexander>  
A.Bolotov@westminster.ac.uk

Montserrat Hermo 

University of the Basque Country, San Sebastián, Spain  
montserrat.hermo@ehu.es

Paqui Lucio 

University of the Basque Country, San Sebastián, Spain  
<http://www.sc.ehu.es/paqui>  
paqui.lucio@ehu.eus

---

## Abstract

When building tableau for temporal logic formulae, applying a two-pass construction, we first check the validity of the given tableau input by creating a tableau graph, and then, in the second “pass”, we check if all the eventualities are satisfied. In one-pass tableaux checking the validity of the input does not require these auxiliary constructions. This paper continues the development of one-pass tableau method for temporal logics introducing tree-style one-pass tableau systems for Computation Tree Logic (CTL) and shows how this can be extended to capture Extended CTL (ECTL). A distinctive feature here is the utilisation, for the core tableau construction, of the concept of a context of an eventuality which forces its earliest fulfilment. Relevant algorithms for obtaining a systematic tableau for these branching-time logics are also defined. We prove the soundness and completeness of the method. With these developments of a tree-shaped one-pass tableau for CTL and ECTL, we have formalisms which are well suited for the automation and are amenable for the implementation, and for the formulation of dual sequent calculi. This brings us one step closer to the application of one-pass context-based tableaux in certified model checking for a variety of CTL-type branching-time logics.

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

**Keywords and phrases** Temporal logic, fairness, expressiveness, branching-time

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.14

**Funding** *Alexander Bolotov*: This author has been supported by the European Union (FEDER funds) under grant TIN2017-86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU18-182.

*Montserrat Hermo*: This author has been supported by the European Union (FEDER funds) under grant TIN2017-86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU18-182.

*Paqui Lucio*: This author has been supported by the European Union (FEDER funds) under grant TIN2017-86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU18-182.



© Alex Abuin, Alexander Bolotov, Montserrat Hermo, and Paqui Lucio;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 14; pp. 14:1–14:20

Leibniz International Proceedings in Informatics



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

## 1 Introduction

In this paper we continue our investigation of tableaux-based deductive techniques for temporal logic having in mind their potential application in model checking, more specifically, in certified model checking [16], which aims to generate proofs as certificates of the properties that are verified, as well as counterexamples for those properties that are invalidated. There are two known ways to build tableau constructions for temporal logic formulae (for the survey of tableau method for temporal logic we refer an interested reader to [13]). Two-pass constructions check the validity of the given tableaux input in two passes - in the first pass a tableau graph is obtained and the second “pass” checks the satisfiability of all eventualities. In one-pass tableaux checking the validity of the input does not require these auxiliary constructions. This paper continues the development of one-pass tableau method for temporal logics [11, 4], this time for Computation Tree Logic (CTL) and Extended Computation Tree Logic (ECTL) introduced, respectively, in [6] and [10]. The core tableau construction is based on the concept of a *context of an eventuality*, which is a set of formulae that “accompanies” the eventuality in the label of the node of a tableaux graph. Our specific tableau rules that involve context force the earliest fulfilment of eventualities. In previous works such a context-based one-pass tableaux approach has been developed for propositional linear-time temporal logic, PLTL [11], and for the branching-time logic ECTL<sup>#</sup> [4], which introduces a new class of fairness constraints utilising the “until” temporal operator. It has also been shown how, in the linear-time case, the method, being mingled with a SAT solver, can be invoked as part of the certified model checking for PLTL [2]. Aiming at similar developments for branching-time cases, in particular for CTL, we make two observations.

Firstly, the satisfiability of a property  $\varphi$  in PLTL can be reduced to checking if a complete transition system satisfies  $\neg\varphi$  (since any counter-model of  $\neg\varphi$  is a model of  $\varphi$ ) and both the satisfiability and model checking are PSPACE-complete [18]. However, the CTL satisfiability problem cannot be reduced to the CTL model checking. In particular, a model checking algorithm for CTL properties (for example [5] implemented in NuSMV) cannot be adapted for testing CTL satisfiability: the model checking problem for CTL is known to be P-complete [7], while the satisfiability problem for CTL is EXPTIME-complete [9]. However, any decision procedure of CTL satisfiability can be used to perform model checking tasks.

Secondly, note that in our previous work one-pass tableaux method was developed for a richer logic - ECTL<sup>#</sup> [4]. However, the application of such model checking procedure for CTL simply based on the existing one-pass tableaux for ECTL<sup>#</sup> would become too “non-intuitive” due to the complexity of its rules that are needed for this richer logic. We also note that the distinguished (and unavoidable) feature of one-pass technique for ECTL<sup>#</sup> is the utilisation of two types of context, unlike in the case of PLTL. Here the so-called “outer” (similar to PLTL) context is a collection of state formulae, and is complemented by so called “inner” context, a collection of path formulae. Subsequently, the development of a simpler one-pass method for CTL is an important task. In our tableau method for CTL, similarly to PLTL, we only need the “outer” context, yet, similar to ECTL<sup>#</sup> the generated tableaux are AND-OR trees. Our results provide an intuitive tableau method that serves as a decision procedure of CTL satisfiability and can also be used in certified model checking of CTL properties hence the method presented in the paper would enable a subsequent study and implementation of a certified model checker. With the development of tree-shaped one-pass tableaux for CTL and ECTL, this paper has proved the effectiveness of the approach which now covers both linear-time and a range of branching-time logics. Moreover, the results of this paper give us formalisms which are well suited for the automation and are amenable for the

implementation, and for the formulation of a dual sequent calculi - all these bring us one step closer to the application of these developments in certified model checking for a variety of branching-time logics. Additionally, aiming at the extension of the certified model checking to the branching-time framework, a proof system, e.g. sequent calculus, is required to check the proof certificates in the branching-time setting.

Our extensive search for tableau methods for CTL has not shown a great variety of systems. For example, [3] presents a two-pass tableau, where in the first pass the tableau rules are applied creating a cyclic graph. In the second pass, “bad loops” are pruned (where a “bad loop” is a loop containing some eventuality that is not fulfilled along it). In [1, 12] the authors introduce a single-pass tableaux decision procedure for CTL. It is based on Schwendimann’s one-pass procedure for PLTL [17]. This tableau method uses an additional mechanism for collecting information on the set of formulae in the nodes, and passing it, to subsequent nodes along branches. The information on previously generated nodes helps detecting “bad loops” without constructing the whole graph. Finally, we note that we have not found an explicit formulation of a tableaux (one or two pass) method for ECTL.

To ensure that the presentation of quite technical details in the paper is clear and self-contained, we supply all major technical details in the text. This determines the following structure of the paper. In §2 we give CTL and ECTL syntax and semantics as sublogics of CTL\*. The formulation of the tableau method is presented in §3, where we first give some preliminaries and then overview the tableau construction as an AND-OR tree and provide examples. A *systematic* tableau construction is introduced in §4. In §5 we show further extension of the method for ECTL. In §6 we draw the conclusions and prospects of future work that the presented results open. Finally, in Appendix A we briefly recall the cyclic models characterization of satisfiability in branching temporal logics. The soundness and completeness of our tableau methods are proved in Appendix B. Finally, in Appendix C we depict the complete tableau for the running example in the paper.

## 2 Syntax and Semantics of CTL and ECTL

The language of branching-time logic extends the language of classical propositional logic by future time temporal operators  $\circ$  - “at the next moment of time”,  $\diamond$  - “eventually”,  $\square$  - “always” and  $\mathcal{U}$  - “until”, together with paths quantifiers  $A$  - “for all paths” quantifier, and  $E$  - “there exists a path” quantifier.

The hierarchy of CTL-type family of Branching-time logics (BTL) is defined by releasing restrictions on the concatenations of temporal operators and paths quantifiers which define classes of admissible state formulae distinguished for these logics. As in CTL every temporal operator must be preceded by a path quantifier, this logic cannot express fairness which requires at least the concatenation of  $\square$  and  $\diamond$ . These are tackled by ECTL [9] which enables simple fairness constraints but not their Boolean combinations. ECTL<sup>+</sup> [10] further extends the expressiveness of ECTL allowing Boolean combinations of temporal operators and ECTL fairness constraints (but not permitting their nesting). The logic ECTL<sup>#</sup> [4] extends ECTL<sup>+</sup> by allowing the combinations  $\square(A\mathcal{U}B)$  or  $A\mathcal{U}\square B$ , referred to as modalities  $\square\mathcal{U}$  and  $\mathcal{U}\square$ . The logic CTL\*, often considered as the “full branching-time logic” overcomes all these restrictions on syntax allowing any arbitrary combinations of temporal operators and path quantifiers. For the sake of generality, as all logics we are interested in are subsumed by CTL\*, we first recall CTL\* syntax and then, by restricting it, derive the syntax for each of ECTL<sup>#</sup>, ECTL<sup>+</sup>, ECTL and CTL.

► **Definition 1** (Syntax of CTL<sup>\*</sup>). *Given Prop is a fixed set of propositions, and  $p \in \text{Prop}$ , we define sets of state ( $\sigma$ ) and path ( $\pi$ ) CTL<sup>\*</sup> formulae over Prop as follows:  $\sigma ::= \mathbf{T} \mid p \mid \sigma_1 \wedge \sigma_2 \mid \neg\sigma \mid E\pi$  and  $\pi_{\text{CTL}^*} ::= \sigma \mid \pi_1 \wedge \pi_2 \mid \neg\pi \mid \circ\pi \mid \pi\mathcal{U}\pi$ .*

Observe that in Definition 1 for the set of path formulae we deliberately used an index  $\text{CTL}^*$  and did not use any index for the set of state formulae: the syntax of CTL<sup>\*</sup> sublogics we will define later, will be distinguished exactly by specific to these logics path formulae constructions, while the set of state formulae is preserved from the definition of CTL<sup>\*</sup> syntax. Other usual Boolean operators can be derived from those introduced in the standard way while the “release” ( $\mathcal{R}$ ),  $\diamond$  and  $\square$  operators can be defined as follows:  $\varphi_1 \mathcal{R} \varphi_2 \equiv \neg(\neg\varphi_1\mathcal{U}\neg\varphi_2)$ ,  $\diamond\varphi \equiv \mathbf{T}\mathcal{U}\varphi$ , and  $\square\varphi \equiv \neg\diamond\neg\varphi$ .

We consider a Kripke-style semantics of CTL<sup>\*</sup>: a Kripke structure,  $\mathcal{K}$ , is a triple  $(S, R, L)$  where  $S \neq \emptyset$  is a set of *states*,  $R \subseteq S \times S$  is a total binary relation, called *the transition relation*, and  $L : S \rightarrow 2^{\text{Prop}}$  is a *labelling function*. Our Kripke structures are labelled directed graphs that correspond to Emerson’s R-generable structures, i.e. the transition relation  $R$  is suffix, fusion and limit closed [8]. A *path*  $x$  through a Kripke structure  $\mathcal{K}$  is an infinite sequence of states  $s_i, s_{i+1}, s_{i+2} \dots$  such that  $(s_j, s_{j+1}) \in R$  for any  $j \geq i$ . A *fullpath*  $x$  through a Kripke structure  $\mathcal{K}$  is an infinite sequence of states  $s_0, s_1, s_2 \dots$ , where  $s_0$  is the root. Given a path  $x = s_i, s_{i+1}, \dots$  and a state  $s_k \in x$  such that  $k > i$ , we denote its finite prefix  $x^{\leq k} = s_i, s_{i+1}, \dots, s_k$  and its infinite suffix  $x^{\geq k} = s_k, s_{k+1}, \dots$ . The notation  $\mathcal{K} \upharpoonright x(i)$  denotes a Kripke structure with the set of states of  $\mathcal{K}$  restricted to those that are  $R$ -reachable from  $x(i)$  and  $\text{fullpaths}(\mathcal{K})$  is the set of all fullpaths in  $\mathcal{K}$ . Given the structure  $\mathcal{K} = (S, R, L)$ , the relation  $\models$  evaluates path formulae in a given path  $x$  and state formulae at the state index  $i$  of  $x$  and is defined on atoms by  $\mathcal{K}, x, i \models p$  iff  $p \in L(x(i))$ . Omitting standard definitions for Booleans, we present the relation  $\models$  for temporal connectives and path quantifier  $E$ :

$$\begin{aligned} \mathcal{K}, x, i \models E\pi &\text{ iff there exists a path } y \in \text{fullpaths}(\mathcal{K} \upharpoonright x(i)) \text{ such that } \mathcal{K}, y \models \pi. \\ \mathcal{K}, x, i \models \circ\pi &\text{ iff } \mathcal{K}, x, i+1 \models \pi. \\ \mathcal{K}, x, i \models \pi_1\mathcal{U}\pi_2 &\text{ iff there exists } k \geq i \text{ with } \mathcal{K}, x^{\geq k} \models \pi_2 \text{ and } \mathcal{K}, x^{\geq j} \models \pi_1 \text{ for all } 0 \leq j \leq k-1. \end{aligned}$$

For any set  $\Sigma$  of state formulae,  $\mathcal{K}, x, i \models \Sigma$  iff  $\mathcal{K}, x, i \models \sigma$ , for all  $\sigma \in \Sigma$ . Moreover, if for any fullpath  $x \in \text{fullpaths}(\mathcal{K})$ , we have  $\mathcal{K}, x, 0 \models \Sigma$ , then we simply write  $\mathcal{K} \models \Sigma$ . For a state formula  $\varphi$ , the set of its models,  $\text{Mod}(\varphi)$ , is formed by all triples  $(\mathcal{K}, x, i)$  such that  $\mathcal{K}, x, i \models \varphi$ . Then  $\varphi$  is *satisfiable* ( $\text{Sat}(\varphi)$ ) if  $\text{Mod}(\varphi) \neq \emptyset$ , otherwise  $\varphi$  is *unsatisfiable* ( $\text{UnSat}(\varphi)$ ). For state formulae  $\varphi$  and  $\varphi'$ , if  $\text{Mod}(\varphi) = \text{Mod}(\varphi')$  then  $\varphi$  and  $\varphi'$  are logically equivalent denoted as  $\varphi \equiv \varphi'$ . Satisfiability and logical equivalence are generalised to sets of state formulae  $\Sigma$ , in the natural way (formally by substituting  $\varphi$  with  $\Sigma$  in the relevant definitions and stating that  $\Sigma$  is satisfied when all its formulae are satisfied).

For each of BTL logics ECTL<sup>#</sup>, ECTL<sup>+</sup>, ECTL and CTL its syntax is defined over a fixed set of propositions Prop, such that the definition of state formulae is the same as for CTL<sup>\*</sup> (Def. 1), and the eventuality  $\diamond\varphi$  is the abbreviation for  $\mathbf{T}\mathcal{U}\varphi$ . The specific for these logics restrictions on the CTL<sup>\*</sup> grammar in Definition 1 generate the corresponding sets for path formulae, as in Definition 2. For technical convenience, here we define  $\square$  as the basic language operator.

► **Definition 2** (Paths formulae for ECTL<sup>#</sup>, ECTL<sup>+</sup>, ECTL and CTL).

$$\begin{aligned} \pi_{\text{ECTL}^{\#}} &::= \sigma \mid \pi_1 \wedge \pi_2 \mid \neg\pi \mid \circ\sigma \mid \sigma\mathcal{U}(\sigma \wedge \diamond\sigma) \mid \square(\sigma \vee \square\sigma) \mid \sigma\mathcal{U}(\square\sigma) \mid \square(\sigma\mathcal{U}\sigma) \\ \pi_{\text{ECTL}^+} &::= \sigma \mid \pi_1 \wedge \pi_2 \mid \neg\pi \mid \circ\sigma \mid \sigma\mathcal{U}\sigma \mid \square\sigma \mid \square\diamond\sigma \mid \diamond\square\sigma. \\ \pi_{\text{ECTL}} &::= \sigma \mid \neg\pi \mid \circ\sigma \mid \sigma\mathcal{U}\sigma \mid \square\sigma \mid \square\diamond\sigma \mid \diamond\square\sigma. \\ \pi_{\text{CTL}} &::= \sigma \mid \neg\pi \mid \circ\sigma \mid \sigma\mathcal{U}\sigma \mid \square\sigma. \end{aligned}$$

► **Definition 3** (Literals). *Let Prop be a fixed set of CTL (ECTL) propositions, and let  $\rho \in \text{Prop}$ . Then the set of CTL (ECTL) literals is defined as  $\text{Lit} ::= \mathbf{f} \mid \mathbf{t} \mid \rho \mid \neg\rho$ .*

It is well known that any given branching-time formula  $\varphi$  can be converted to a formula  $\text{NNF}(\varphi)$  - the Negation Normal Form of  $\varphi$  obtained by pushing negations inwards until they only apply to literals. The conversion is based on well-known equivalences which ensure that  $\varphi$  and  $\text{NNF}(\varphi)$  have exactly the same models, i.e. are logically equivalent. Consequently, we assume that inputs for the tableaux procedure in CTL and ECTL are given in NNF. For simplicity, we will write  $\sim\varphi$  instead of  $\text{NNF}(\neg\varphi)$ . Also, for a finite set  $\Phi = \{\varphi_1, \dots, \varphi_n\}$ , we let  $\sim\Phi = \text{NNF}(\neg\bigwedge_{i=1}^n \varphi_i)$ .

Further, it is important to note that the nesting of “pure path formulae”, totally unrestricted in  $\text{CTL}^*$ , is now restricted in its sublogics by relevant grammar cases for paths formulae. For example, a  $\text{CTL}^*$  formula (1) is not an  $\text{ECTL}^\#$  formula. Rewriting it as  $\mathbf{A}(\mathbf{TU}(\circ p \wedge \mathbf{E}\circ\neg p))$  we can see

$$\mathbf{A}\diamond(\circ p \wedge \mathbf{E}\circ\neg p) \quad (1)$$

that  $\circ p \wedge \mathbf{E}\circ\neg p$  is neither a state formula nor of the form  $\Box\sigma$ . Note that the validity of (1) which is indicative for  $\text{CTL}^*$ , is directly linked to the limit closure property [8]. Similarly, a  $\text{ECTL}^\#$  formula  $\mathbf{A}((p\mathcal{U}\Box q) \wedge (s\mathcal{L}\Box\neg q))$  is not an  $\text{ECTL}^+$  formula because  $p\mathcal{U}\Box q$  and  $s\mathcal{L}\Box\neg q$ , hence their conjunction, are not admissible  $\text{ECTL}^+$  formulae. Further, an  $\text{ECTL}^+$  formula (2) does not belong to ECTL

$$\mathbf{E}(\Box\diamond q \wedge \diamond\Box\neg q) \quad (2)$$

as  $\Box\diamond q \wedge \diamond\Box\neg q$  is not an admissible ECTL path formula. Finally, the fairness constraint (3) expressible in ECTL cannot be constructed in CTL syntax as every temporal operator

$$\mathbf{E}\Box\diamond q \quad (3)$$

in a CTL formula must be preceded by a path quantifier. Note that it is important to distinguish the problem if a formula of a superlogic belongs to a sublogic and the problem if a formula of a superlogic can be expressed in a sublogic. For example,  $\mathbf{E}(\Box\diamond q \vee \diamond\Box\neg q)$ , similarly to formula (2) does not belong to ECTL but unlike (2), it is expressible in this logic, as  $\mathbf{E}(\Box\diamond q \vee \diamond\Box\neg q) \equiv \mathbf{E}\Box\diamond q \vee \mathbf{E}\diamond\Box\neg q$  which is an ECTL formula if we define  $\vee$  via  $\wedge$ .

■ **Table 1** Classification of context-based tableaux systems for CTL-type logics and relevant difficult cases of concatenations of temporal operators and path quantifiers.

BTL Logics	$\mathbf{E}\Box\diamond q$	$\mathbf{E}(\Box\diamond q \wedge \diamond\Box\neg q)$	$\mathbf{A}((p\mathcal{U}\Box q) \vee (s\mathcal{L}\Box\neg r))$	$\mathbf{A}\diamond(\circ p \wedge \mathbf{E}\circ\neg p)$	One-pass Tableaux
$\mathcal{B}(\mathcal{U}, \circ)$ (CTL)	X	X	X	X	This paper
$\mathcal{B}(\mathcal{U}, \circ, \Box\diamond)$ (ECTL)	✓	X	X	X	This paper
$\mathcal{B}^+(\mathcal{U}, \circ, \Box\diamond)$ ( $\text{ECTL}^+$ )	✓	✓	X	X	✓
$\mathcal{B}^+(\mathcal{U}, \circ, \mathcal{U}\Box)$ ( $\text{ECTL}^\#$ )	✓	✓	✓	X	✓
$\mathcal{B}^*(\mathcal{U}, \circ)$ ( $\text{CTL}^*$ )	✓	✓	✓	✓	X

Table 1 represents BTL logics classified by their expressiveness using “ $\mathcal{B}$ ” for “Branching”, followed by the set of only allowed modalities as parameters;  $\mathcal{B}^+$  indicates admissible Boolean combinations of the modalities and  $\mathcal{B}^*$  reflects “no restrictions” in either concatenations

of the modalities or Boolean combinations between them following the notation initially proposed in [8] and further tuned in [15]. The top row of the figure represents the indicative formulae (1)-(3) for the listed logics. The last column in Table 1 reflects the development of the context-based one-pass tableaux technique for CTL-type logics: the method has been developed for ECTL<sup>#</sup> ([4] where the motivation was to tackle complex cases of fairness). In this paper we introduce the technique for CTL and ECTL, while the case of ECTL<sup>+</sup> can be tackled effectively by the technique developed for ECTL<sup>#</sup>. Indeed, ECTL<sup>+</sup> and ECTL<sup>#</sup> have similar cases of the Boolean combination of eventualities in the scope of A and E, namely disjunctions of the eventualities in the scope of the A quantifier and conjunctions of eventualities in the scope of the E quantifier, see [4] for details. Thus, Table 1 also reflects syntactical cases of concatenations of temporal operators and path quantifiers that are difficult for context-based one-pass tableaux. To tackle these cases, in addition to  $\alpha$ - and  $\beta$ -rules, that are standard to the tableaux, novel  $\beta^+$ -rules which use the context to force the eventualities to be fulfilled as soon as possible, were introduced. As ECTL<sup>#</sup> is more expressive than ECTL<sup>+</sup> in allowing new type of fairness constraints that use the  $\mathcal{U}$  operator, the relevant rules introduced in [4] would cover all difficult concatenations of operators in ECTL<sup>+</sup>. Hence, simply treating the case of one-pass context-based tableaux for ECTL<sup>+</sup> as solved by the relevant development for a richer logic ECTL<sup>#</sup>, in this paper we concentrate on bridging the gap in our roadmap in supplying BTL logics by this technique, by developing the method for CTL and ECTL. The ultimate target of this roadmap - the one-pass context-based tableaux for CTL<sup>\*</sup> remains extremely difficult and an open problem.

### 3 Context-based One-pass Tableau Method for CTL

We precede the presentation of the method by the introduction of a number of important concepts. Firstly, we introduce a concept of *basic modality* which reflects the restrictions on forming the basic admissible combinations of temporal operators in the scope of a path quantifier. Recall that formulae of CTL and ECTL logics are written in NNF. Abbreviating by Q either of the path quantifiers A or E, we consider a basic modality of CTL or ECTL logic to be of the form QT, where T is a temporal operator. The structure QT is generated by the grammar rules for these logics in Def. 2. We can identify all basic modalities in a given formula  $\varphi$  by finding its most embedded modality(es), say  $M_1$ , then looking at the next basic modality in which  $M_1$  is embedded, etc. For example, basic modalities for CTL are structures  $Q\circ$ ,  $QU$ , and  $Q\Box$  while for ECTL these will be  $Q\circ$ ,  $QU$ ,  $Q\Box$ ,  $Q\Diamond\Box$  and  $Q\Box\Diamond$ . If we analyse a CTL formula  $E\circ A\circ p$  then the most embedded basic modality,  $M_1$ , would be  $A\circ p$ , which is embedded as  $E\circ M_1$ . These are generalised in Definition 4.

► **Definition 4** (ECTL<sup>#</sup>, ECTL<sup>+</sup>, ECTL and CTL Basic Modalities).

$$\begin{aligned} M_{ECTL} &::= c \mid Q\circ M \mid Q(MUM) \mid Q\Box M \mid Q\Box\Diamond M \mid Q\Diamond\Box M. \\ M_{CTL} &::= c \mid Q\circ M \mid Q(MUM) \mid Q\Box M. \end{aligned}$$

where  $c$  stands for a purely classical formula (we can consider a purely classical formula as a zero-degree basic modality) and  $M$  stands for any basic modality of CTL in the definition of  $M_{CTL}$  and of ECTL in the definition of  $M_{ECTL}$ . Note that we have “derived” cases of basic modalities for  $\Diamond M$  and  $MRM$ . In what follows, every CTL modality  $QU$  or  $Q\Diamond$  is called *eventuality*.

CTL tableau rules are based on fixpoint characterisation of its basic modalities: (in the equations below  $\nu$  and  $\mu$  stand for “minimal fixpoint” and “maximal fixpoint” operators, respectively)



$$\begin{aligned} E\Box\varphi &= \nu\rho(\varphi \wedge E\circ\rho) & E(\varphi \mathcal{R}\psi) &= \nu\rho(\psi \wedge (\varphi \vee E\circ\rho)) \\ A\Box\varphi &= \nu\rho(\varphi \wedge A\circ\rho) & A(\varphi \mathcal{R}\psi) &= \nu\rho(\psi \wedge (\varphi \vee A\circ\rho)) \end{aligned} \quad (4)$$

$$\begin{aligned} E\Diamond\varphi &= \mu\rho(\varphi \vee E\circ\rho) & E(\varphi \mathcal{U}\psi) &= \mu\rho(\psi \vee (\varphi \wedge E\circ\rho)) \\ A\Diamond\varphi &= \mu\rho(\varphi \vee A\circ\rho) & A(\varphi \mathcal{U}\psi) &= \mu\rho(\psi \vee (\varphi \wedge A\circ\rho)) \end{aligned} \quad (5)$$

This fixpoint characterisation of basic CTL and ECTL modalities as maximal or minimal fixpoints give rise to their analytical classification as  $\alpha$ - or  $\beta$ -formulae which are associated, in the tableau with  $\alpha$ - and  $\beta$ -rules:  $\mathbf{Q}\Box$ , and  $\mathbf{Q}\mathcal{R}$  as maximal fixpoints are classified as  $\alpha$ -formulae while  $\mathbf{Q}\Diamond$  and  $\mathbf{Q}\mathcal{U}$  as minimal fixpoints are  $\beta$ -formulae. This is also reflected in the known equivalences:

$$\begin{aligned} E\Box\varphi &= \varphi \wedge E\circ E\Box\varphi & E(\varphi \mathcal{R}\psi) &= \psi \wedge (\varphi \vee E\circ E(\varphi \mathcal{R}\psi)) \\ A\Box\varphi &= \varphi \wedge A\circ A\Box\varphi & A(\varphi \mathcal{R}\psi) &= \psi \wedge (\varphi \vee A\circ A(\varphi \mathcal{R}\psi)) \end{aligned} \quad (6)$$

$$\begin{aligned} E\Diamond\varphi &= \varphi \vee E\circ E\Diamond\varphi & E(\varphi \mathcal{U}\psi) &= \psi \vee (\varphi \wedge E\circ E(\varphi \mathcal{U}\psi)) \\ A\Diamond\varphi &= \varphi \vee A\circ A\Diamond\varphi & A(\varphi \mathcal{U}\psi) &= \psi \vee (\varphi \wedge A\circ A(\varphi \mathcal{U}\psi)) \end{aligned} \quad (7)$$

The tableau method determines if a given set of CTL state formulae is satisfiable or not. We precede the formal introduction of the technique by its informal overview. The initial node of the tableaux is labelled by a CTL formula in NNF. To expand the root, and any subsequent node, we apply one of the following rules:  $\alpha$ - and  $\beta$ -rules, the “next-state” rule, which reflects a “jump” from a “state” to a “pre-state”, and, finally, characteristic to our approach,  $\beta^+$ -rules, where the use of the context (of an eventuality) is essential. The use of the context in these rules, which is a collection of state formulae accompanying the eventuality in the label of the node, forces the soonest fulfillment of eventualities. We apply  $\alpha$ -,  $\beta$ -, and  $\beta^+$ -rules repeatedly until we reach a node labelled by  $\mathbf{F}$  or by an inconsistent set of formulae, or a node whose labels have already occurred within the path under consideration. In the former case the expansion of the given branch terminates with  $\perp$  as its leaf. In the latter case, a repetitive node in the branch means that the branch has a loop – where some subformulae of the given formula are satisfied forever – which could be “bad” or “good”. A loop is “bad” when it has a node which contains an unfulfilled eventuality, i.e. none of the nodes of the loop satisfies it. In our procedure, the application of  $\beta^+$ -rules to eventualities is essential to distinguish between “good” and “bad” loops - if  $\beta^+$ -rules have already been applied to every eventuality occurring in the branch then we have a ‘good loop’ and this branch represents a model for the given formula. Otherwise, we choose an eventuality to which a corresponding  $\beta^+$ -rule has not been applied.

► **Definition 5** (Syntactically Consistent Set of Formulae). *A set  $\Sigma$  of state formulae  $\sigma$  is syntactically consistent abbreviated as  $\Sigma_{\top}$  if  $\mathbf{F} \notin \Sigma$  and  $\{\sigma, \sim\sigma\} \not\subseteq \Sigma$  for any  $\sigma$ ; otherwise,  $\Sigma$  is inconsistent denoted as  $\Sigma_{\perp}$ .*

► **Definition 6** (Tableau, Consistent Node, Closed branch). *A tableau for a set of CTL state formulae  $\Sigma$  is a labelled tree  $\langle T, \tau, \Sigma \rangle$ , where  $T$  is a tree, and  $\tau$  is a mapping of the nodes of  $T$  to the state formulae, elements of  $\Sigma$ , such that the following two conditions hold: (i) The root is labelled by the set  $\Sigma$ . (ii) For any other node  $m \in T$ , its label  $\tau(m)$  is a set of state formulae obtained as the result of the application of one of the rules in Figures 1, 2 and 4 to its parent node  $n$ . Given the applied rule is  $R$ , we term  $m$  an  $R$ -successor of  $n$ . A node  $n$  of*

a tree  $T$  is consistent, abbreviated as  $n_{\top}$ , if its label,  $\tau(n)$ , is a syntactically consistent set of formulae (see Def. 5), else  $n$  is inconsistent, abbreviated as  $n_{\perp}$ . If a branch  $b$  of  $\tau$ , contains  $n_{\perp} \in b$ , then  $b$  is closed else  $b$  is open.

$(\wedge) \frac{\Sigma, \sigma_1 \wedge \sigma_2}{\Sigma, \sigma_1, \sigma_2}$	$(Q\Box) \frac{\Sigma, Q\Box\sigma}{\Sigma, \sigma, Q\Box\sigma}$
$(\vee) \frac{\Sigma, \sigma_1 \vee \sigma_2}{\Sigma, \sigma_1 \mid \Sigma, \sigma_2}$	$(QU) \frac{\Sigma, Q(\sigma_1 \mathcal{U} \sigma_2)}{\Sigma, \sigma_2 \mid \Sigma, \sigma_1, Q\Box(\sigma_1 \mathcal{U} \sigma_2)}$
$(QR) \frac{\Sigma, Q(\sigma_1 \mathcal{R} \sigma_2)}{\Sigma, \sigma_1, \sigma_2 \mid \Sigma, \sigma_2, Q\Box(\sigma_1 \mathcal{R} \sigma_2)}$	$(Q\Diamond) \frac{\Sigma, Q\Diamond\sigma}{\Sigma, \sigma \mid \Sigma, Q\Box\Diamond\sigma}$

■ **Figure 1**  $\alpha$ - and  $\beta$ -Rules.

The rules in Figure 1 follow the standard for the tableaux classification of rules into  $\alpha$ -rules and  $\beta$ -rules that for the formulae with CTL modalities are based on their analytic classification reflected in Equations (6)-(7). Thus, if a node,  $n$ , in the tableau graph is labelled by a set of formulae,  $\Sigma, \varphi$ , and a designated formula for the application of tableau rules,  $\varphi$ , is an  $\alpha$ -formula -  $Q\Box$  or  $QR$ , then a corresponding  $\alpha$ -rule applies, while if  $\varphi$  is a  $\beta$ -formula -  $Q\Diamond$  or  $QU$  then a corresponding  $\beta$ -rule applies. In the latter case we treat  $\Sigma$  as a (possibly empty) context for the eventuality  $\varphi$ . These applications of  $\alpha$ - and  $\beta$ -rules generate a set of formulae in the conclusion as a label for the successor node,  $n + 1$ , in case of an  $\alpha$ -rule, or as labels of two successors of  $n$ , in case of a  $\beta$ -rule.

When a node  $n$  is labelled by an *elementary set of formulae* – i.e. a set which exclusively formed by literals and formulae of the form  $Q\Box\sigma$  – then this structure is analogous to the construction to a “state” in the terminology of [19]; it enables us to construct the successors of  $n$  corresponding to “pre-states” [19]. According to the next proposition we are guaranteed to reach such a tree structure, where the last node of every branch, at this stage of the construction, is a state.

► **Proposition 7.** *Any set of CTL state formulae has a tableau  $T$  such that the last node of every branch is labelled by an elementary set of state formulae.*

**Proof.** Repeatedly apply to every expandable node any applicable  $\alpha$ - or  $\beta$ -rule until all expandable nodes are elementary. Then, the next-state rule must be applied to every expandable node. ◀

$(Q\Box) \frac{\Sigma, A\Box\sigma_1, \dots, A\Box\sigma_\ell, E\Box\sigma'_1, \dots, E\Box\sigma'_k}{\sigma_1, \dots, \sigma_\ell, \sigma'_1 \ \& \ \dots \ \& \ \sigma_1, \dots, \sigma_\ell, \sigma'_k}$ where $\Sigma$ is a set of literals.
--

■ **Figure 2** Next-state Rule. (“&” joins AND-successors in the conclusion.)

Proposition 7 enables the application of the so-called “next-state rule” depicted in Figure 2. Applying this rule we split the current branch at node  $n$  where the set  $\Sigma, A\Box\sigma_1, \dots, A\Box\sigma_\ell, E\Box\sigma'_1, \dots, E\Box\sigma'_k$  is satisfied, into  $k$  branches (i.e. into the number of branches equal to the number of  $E\Box$  constraints) where the successors of  $n$  along these branches are AND-successors, and are labelled each by a different set  $\sigma_1, \dots, \sigma_\ell, \sigma'_i$ , for  $i \in \{1, \dots, k\}$ . This rule splits branches in a “conjunctive” way, and we use the symbol  $\&$  to represent the generation of AND-successors of





## 14:10 One-Pass Context-Based Tableaux Systems for CTL and ECTL

sequence of  $A\circ$ ) because these formulas would be necessarily repeated along any branch - indeed, if we use  $\sim \Sigma$  instead of  $\sim \Sigma'$  we will generate a branch for each  $A\Box$  that will be immediately closed.

$$\boxed{\begin{array}{l} (QU)^+ \frac{\Sigma, Q(\sigma_1 \mathcal{U} \sigma_2)}{\Sigma, \sigma_2 \mid \Sigma, \sigma_1, Q\circ Q((\sigma_1 \wedge \sim \Sigma') \mathcal{U} \sigma_2)} \quad (Q\Diamond)^+ \frac{\Sigma, Q\Diamond\sigma}{\Sigma, \sigma \mid \Sigma, Q\circ Q((\sim \Sigma') \mathcal{U} \sigma)} \\ \text{where } \Sigma' = \Sigma \setminus \{(A\circ)^i A\Box\sigma \mid i \geq 0 \text{ and } (A\circ)^i A\Box\sigma \in \Sigma\} \text{ and } (A\circ)^i \text{ stands for } i \text{ times } A\circ. \end{array}}$$

■ **Figure 4**  $\beta^+$ -Rules.

► **Definition 12** (Next-Step Variant). *A state formula  $Q(\sim \Sigma' \mathcal{U} \sigma)$  obtained by the application of a  $\beta^+$ -rule to formula  $Q(\sigma_1 \mathcal{U} \sigma_2)$  or  $Q\Diamond\sigma$  is called the next-step variant of  $Q(\sigma_1 \mathcal{U} \sigma_2)$ .*

$$\begin{array}{c} A(\mathbf{F}\mathcal{R}\neg q), \mathbf{E}(p\mathcal{U}q) \\ \swarrow \quad \searrow \\ A(\mathbf{F}\mathcal{R}\neg q), q \quad A(\mathbf{F}\mathcal{R}\neg q), p, \mathbf{E}\circ \mathbf{E}((p \wedge \mathbf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q) \end{array}$$

(EU)<sup>+</sup>

■ **Figure 5** Application of rule (EU)<sup>+</sup> (we mark in grey the eventuality to which the  $\beta^+$ -rule applies).

► **Example 13.** Figure 5 reflects the application of the rule (EU)<sup>+</sup> to the left-most expandable node in Figure 3 (labelled by  $\mathbf{E}(p\mathcal{U}q), A(\mathbf{F}\mathcal{R}\neg q)$ ). Here, the context of the eventuality  $\mathbf{E}(p\mathcal{U}q)$  is the  $A\mathcal{R}$ -formula. The rule (EU)<sup>+</sup> splits the tableau into two branches. The left successor is labelled by  $q, A(\mathbf{F}\mathcal{R}\neg q)$  and the right successor is labelled by  $p, \mathbf{E}\circ \mathbf{E}(p \wedge \mathbf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q, A(\mathbf{F}\mathcal{R}\neg q)$ , where the middle formula  $\mathbf{E}\circ \mathbf{E}(p \wedge \mathbf{E}(\mathbf{T}\mathcal{U}q))\mathcal{U}q$  is the next-step variant of the eventuality  $\mathbf{E}(p\mathcal{U}q)$  and it contains the NNF of the negation of the context for this eventuality, i.e.  $\mathbf{E}(\mathbf{T}\mathcal{U}q) = \text{NNF}\neg A(\mathbf{F}\mathcal{R}\neg q)$ .

## 4 Systematic Tableau Construction

In this section we define an algorithm,  $\mathcal{A}^{sys}$ , that constructs a *systematic tableau*. Let us observe that, due to the rule (Q $\circ$ ), any open tableau should have a collection of open branches including all the (Q $\circ$ )-successors of any node labelled by an elementary set of formulae. These collections of branches are called *bunches*. Any open bunch of the systematic tableau, constructed by the algorithm  $\mathcal{A}^{sys}$  introduced in this section, enables the construction of a model for the initial set of formulae.

The algorithm  $\mathcal{A}^{sys}$  constructs an *expanded* tableau (see Definition 25) for the given input.  $\mathcal{A}^{sys}$  applied to the input  $\Sigma_0$ , denoted as  $\mathcal{A}^{sys}(\Sigma_0)$ , returns a systematic tableau  $\mathcal{A}_{\Sigma_0}^{sys}$ . Intuitively, “expanded” means “complete” in the sense that any possible rule has been already applied at every node. Though the best way to implement this algorithm is a depth-first construction, for clarity, we formulate it as a breadth-first construction of a collection of subtrees. The procedure `Uniform_Tableau`, in the above Algorithm 1, was introduced in Definition 10 along with the notion of a uniform set of state formulae. The notation  $T_1[\ell \leftarrow T_2]$  stands for the tableau  $T_1$  where the expandable  $\ell$  is substituted by the tableau  $T_2$ . In particular,  $T[\ell \leftarrow \text{Uniform\_Tableau}(\Sigma)]$  is the tableau  $T$  where the expandable  $\ell$  is substituted by the `Uniform_Tableau`( $\Sigma$ ). Procedure `Eventuality_Selection` chooses an

■ **Algorithm 1** Systematic Tableau Construction.

---

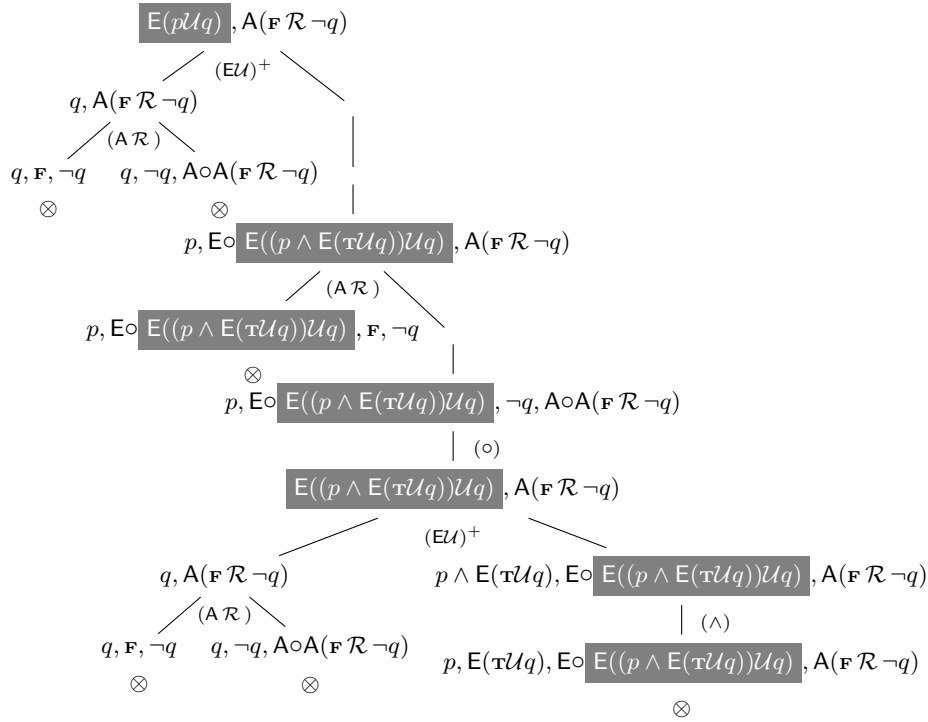
```

1: procedure SYSTEMATIC_TABLEAU( $\Sigma_0$ )           ▷ where  $\Sigma_0$ : set of CTL state formulae
2:   if  $\Sigma_0$  is not uniform then  $T := \text{Uniform\_Tableau}(\Sigma_0)$ 
3:   while  $T$  has at least one expandable node do
4:     ▷ Invariant: Any expandable node of  $T$  is labelled by an uniform set
5:     Choose any node  $\ell$  in  $T$  such that  $\tau(\ell)$  is expandable           ▷  $\tau(\ell)$  is uniform
6:     if there is no eventuality in  $\tau(\ell)$  then  $T := T[\ell \leftarrow \text{Uniform\_Tableau}(\tau(\ell))]$ 
7:     else
8:       Eventuality_Selection( $\tau(\ell)$ )
9:       Apply_ $\beta^+$ -rule( $\tau(\ell)$ )
10:      Let  $\ell_1, \ell_2$  the two children of  $\ell$ 
11:      for  $i = 1 \dots 2$  do
12:        if  $\ell_i$  is expandable and  $\tau(\ell_i)$  is not uniform then
13:           $T := T[\ell_i \leftarrow \text{Uniform\_Tableau}(\tau(\ell_i))]$ 

```

---

eventuality to which the corresponding  $\beta^+$ -rule ( $(\text{QU})^+$  or  $(\text{Q}\diamond)^+$ ) can be applied. Procedure  $\text{Apply}_{\beta^+}\text{-rule}(\Sigma)$  applies the corresponding  $\beta^+$ -rule to the selected eventuality, it also keeps as . the next-step variant (Definition 12) of such eventuality.



■ **Figure 6** A closed tableau for  $\{E(pUq), A(FR¬q)\}$ .

► **Example 14.** The application of the Algorithm 1 to the set  $\{E(pUq), A(FR¬q)\}$  shown in Figure 6 selects the eventuality  $E(pUq)$  and applies the rule  $(\text{EU})^+$  as explained in Example 13. The left successor node is labelled by  $q, A(FR¬q)$ . Further expansion of this node by applying the rule  $(\text{AR})$  generates two inconsistent successor nodes. Applying the  $\text{AR}$ -rule

to the right successor, we obtain the left successor node which is inconsistent and a right successor whose label is an elementary set. Thus, we apply the “next-state” rule generating the successor labelled by the set of two formulae - arguments of  $E\circ$  and  $A\circ$ . In this “pre-state” we select the eventuality  $\mathcal{EU}$  and generate two successor nodes applying again the  $\beta^+$ -rule. The left successor is subsequently expanded by two inconsistent successors of the  $A\mathcal{R}$ -rule. The right successor is expanded by the  $\wedge$ -rule and then, since  $\text{NNF}(\neg E(\tau\mathcal{U}q)) = A(\mathbf{F}\mathcal{R}\neg q)$ , the node is syntactically inconsistent, because it contains  $E(\tau\mathcal{U}q)$  and  $\sim E(\tau\mathcal{U}q)$  (see Definition 5).

A tableau for  $E(p\mathcal{U}q), A(\mathbf{F}\mathcal{R}\neg q)$  is also exhibited in [1, 12]. Note the direct correspondence between our context-based tableau (Figure 6) and the one in [1, 12]- they have exactly the same nodes. The right-most branch, in our case, closes by (syntactical) inconsistency, likewise all the other branches. The difference is that, in this branch, the inconsistency comes from the use of the context in the selected eventuality. The corresponding branch in the tableau in [1, 12] is closed by the detection of a “bad loop”. Intuitively, whenever the tableau in [1, 12] detects a “bad loop”, our tableau is closed by contradiction.

When the input is a satisfiable set, the systematic tableau aims to obtain a loop-node that makes branches eventuality-covered. Next, we define both concepts.

► **Definition 15 (Loop-node).** *Let  $b$  be a tableau branch and  $n_i \in b$  ( $0 \leq i$ ). Then  $n_i$  is a loop-node if there exists  $n_j \in b$  ( $0 \leq j < i$ ) such that  $\tau(n_i) \subseteq \tau(n_j)$ . We say that  $n_j$  is a companion node of  $n_i$ .*

► **Definition 16 (Eventuality-covered Branch).** *A tableau branch  $b = n_0, n_1, \dots, n_i$  is eventuality-covered if  $n_i$  is a loop-node, with a companion node  $n_j$  ( $0 \leq j < i$ ), both labelled by a uniform set  $\Sigma$  such that every eventuality in  $\tau(n_i)$  is selected in some node  $n_k$  ( $j \leq k < i$ ).*

The procedure `Eventuality_Selection` performs in some fair way that ensures that any open branch will ever be *eventuality-covered*.

► **Definition 17 (Non-expandable Node).** *A node  $n$  is non-expandable if  $\tau(n) = \Sigma_\perp$  or  $n$  is a loop-node of branch  $b$  which is eventuality-covered. Otherwise,  $n$  is expandable.*

Consequently, an expandable node is either a node that is not a loop-node or a loop-node whose branch is not eventuality-covered.

► **Definition 18 (Bunch in a Tableau, Closed Bunch and Tableau).** *A bunch  $b$  is a collection of branches that is maximal with respect to  $(Q\circ)$ -successor, i.e. every  $(Q\circ)$ -successor of any node in  $b$  is also in  $b$ . A bunch  $b$  is a closed bunch if, and only if, at least one of its branches is closed, otherwise it is open. A tableau is closed if, and only if, all its bunches are closed.*

Therefore, any open tableau has at least one open bunch, formed by one or more open branches. Open branches are ended in a loop node. Open bunches represent models, specifically *cyclic models* as defined in Appendix A.

## 5 Extending the Tableau from CTL to ECTL

In this section we explain a (relatively easy) way to extend the CTL tableau method to the more expressive logic ECTL. This is achieved by adding the new rules given in Figure 7. The  $\alpha$ -rule ( $Q\Box\Diamond$ ) and the  $\beta$ -rule ( $Q\Diamond\Box$ ) that respectively correspond to the following logical equivalences for the basic modalities that extend CTL to ECTL:

$$\begin{aligned} E\Box\Diamond\sigma &\equiv E\Diamond\sigma \wedge E\circ E\Box\Diamond\sigma & E\Diamond\Box\sigma &\equiv E\Box\sigma \vee (E\Diamond\sigma \wedge E\circ E\Diamond\Box\sigma) \\ A\Box\Diamond\sigma &\equiv A\Diamond\sigma \wedge A\circ A\Box\Diamond\sigma & A\Diamond\Box\sigma &\equiv A\Box\sigma \vee (A\Diamond\sigma \wedge A\circ A\Diamond\Box\sigma) \end{aligned} \quad (8)$$

There are no additional  $\beta^+$ -rules: eventualities  $Q\Diamond\sigma$  introduced by the rules in Figure 7 are CTL-modalities handled by the  $\beta^+$ -rules of the method for CTL.

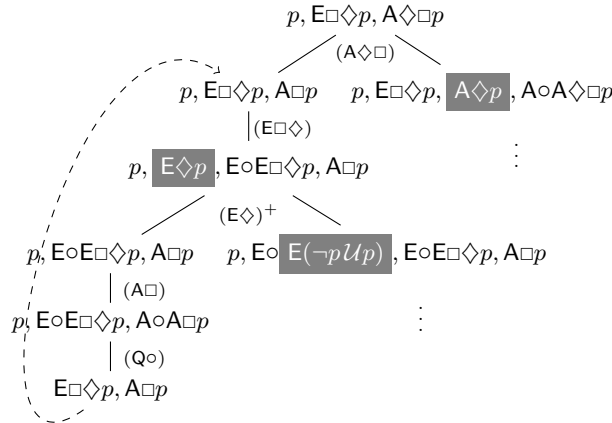
$$\boxed{(\mathbf{Q}\Box\Diamond) \frac{\Sigma, \mathbf{Q}\Box\Diamond\sigma}{\Sigma, \mathbf{Q}\Diamond\sigma, \mathbf{Q}\circ\mathbf{Q}\Box\Diamond\sigma} \quad (\mathbf{Q}\Diamond\Box) \frac{\Sigma, \mathbf{Q}\Diamond\Box\sigma}{\Sigma, \mathbf{Q}\Box\sigma \mid \Sigma, \mathbf{Q}\Diamond\sigma, \mathbf{Q}\circ\mathbf{Q}\Diamond\Box\sigma}}$$

■ **Figure 7** RULES FOR EXTENDING CTL TO ECTL.

To complete the extension, let us recall that in [4] some (subsumption-like) simplification rules are needed to ensure the termination of the tableau method for the logic ECTL#. Though the method for CTL does not need any of such rules, the handling of the more expressive modalities in ECTL – by the rules in Figure 7 – combined with our  $\beta^+$ -rules, requires the following simplification rule:

$$(\Box\mathbf{QU}) \{ \mathbf{Q}((\sigma_1 \wedge \chi)\mathcal{U}\sigma_2), \mathbf{Q}(\sigma_1\mathcal{U}\sigma_2) \} \longrightarrow \{ \mathbf{Q}((\sigma_1 \wedge \chi)\mathcal{U}\sigma_2) \} \quad (9)$$

By means of this rule, any next-step variant of an eventuality  $\varphi$  subsumes the original eventuality  $\varphi$  that could appear repeatedly after the application of one of the rules in Figure 7.



■ **Figure 8** ECTL tableau for  $\{p, E\Box\Diamond p, A\Diamond\Box p\}$  ( $\vdots$  means this branch expansion is not depicted).

► **Example 19.** Figure 8 shows an open tableau with the application of the two rules added to extend CTL to ECTL. We outline a single branch where the ECTL-rules of Figure 7 exclusively apply in the first two steps whilst the rest of steps always apply CTL-rules. We only show the left-most branch because it is an expanded open branch from which the model  $\langle p \rangle^\omega$  can be constructed.

## 6 Conclusion

We introduced a one-pass context-based tableau method for temporal logics CTL and ECTL, providing the soundness and completeness arguments and illustrating the method on a number of examples. The distinctive feature of the method presented in the paper, is that the core tableau construction is based on the concept of a context of an eventuality. The method

developed in the paper is much simpler than the analogous technique obtained earlier for a richer logic - ECTL<sup>#</sup> where two types of context (both outer and inner contexts) are used. The construction in this paper only uses the “outer” context, however, similar to ECTL<sup>#</sup>, generates tableaux as AND-OR trees.

Our results provide intuitive tableau methods that serve as decision procedures of CTL and ECTL satisfiability. The results of this paper also give us formalisms which are well suited for the automation and are amenable for the implementation, and for the formulation of a dual sequent calculi. All these enable a potential application of the developed tableau methods in certified model checking.

The two tableau methods presented here have double-exponential time worst case complexity. Indeed, a trivial adaptation of [11] allows us to say that the so-called *closure* – the set of all formulas that could appear in a tableau – has in the worst case size  $O(2^{O(2^n)})$ , where  $n$  is the input formula size (this complexity characterisation matches the one of [1, 12]). However, in practice the worst case is very unusual. More often, for example when the context of an eventuality mostly contains modalities  $A\Box$  (which is typical in reactive systems specifications), the number of possible contexts is much smaller and consequently performance is much better.

---

## References

- 1 P. Abate, R. Goré, and F. Widmann. One-pass tableaux for computation tree logic. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 32–46, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-75560-9\_5.
- 2 A. Abuin, A. Bolotov, U. Díaz-de-Cerio, M. Hermo, and P. Lucio. Towards certified model checking for PLTL using one-pass tableaux. In Johann Gamper, Sophie Pinchinat, and Guido Sciavicco, editors, *26th International Symposium on Temporal Representation and Reasoning, TIME 2019, October 16-19, 2019, Málaga, Spain*, volume 147 of *LIPICs*, pages 12:1–12:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.TIME.2019.12.
- 3 M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Inf.*, 20(3):207–226, September 1983. doi:10.1007/BF01257083.
- 4 A. Bolotov, M. Hermo, and P. Lucio. Branching-time logic ECTL<sup>#</sup> and its tree-style one-pass tableau: Extending fairness expressibility of ECTL+. *Theoretical Computer Science*, 813:428–451, 2020. doi:10.1016/j.tcs.2020.02.015.
- 5 J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 1020 states and beyond. *Information and Computation*, 98(2):142–170, 1992. doi:10.1016/0890-5401(92)90017-A.
- 6 E. M. Clarke and E. A. Emerson. Using Branching Time Temporal Logic to Synthesise Synchronisation Skeletons. *Science of Computer Programming*, pages 241–266, 1982. doi:10.1016/0167-6423(83)90017-5.
- 7 E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986. doi:10.1145/567067.567080.
- 8 E. A. Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 995–1072. MIT Press, Cambridge, USA, 1990.
- 9 E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985. doi:10.1016/0022-0000(85)90001-7.
- 10 E. A. Emerson and J. Y. Halpern. Sometimes and not never revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.

- 11 J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. Dual systems of tableaux and sequents for PLTL. *Journal of Logic and Algebraic Programming*, 78(8):701–722, 2009. doi:10.1016/j.jlap.2009.05.001.
- 12 R. Goré. And-or tableaux for fixpoint logics with converse: LTL, CTL, PDL and CPDL. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 2014. doi:10.1007/978-3-319-08587-6\_3.
- 13 R. Goré. Tableau methods for modal and temporal logics. In Marcello D’Agostino, Dov M. Dov Gabbay, Reiner Hähnle, and Joachim Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Springer, Netherlands, Dordrecht, 1999.
- 14 R. Kashima. An axiomatization of ECTL. *J. Log. Comput.*, 24(1):117–133, 2014. doi:10.1093/logcom/ext005.
- 15 N. Markey. Temporal logics. Course notes, Master Parisien de Recherche en Informatique, Paris, France, 2013. URL: <http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/NM-coursTL13.pdf>.
- 16 A. Mebsout and C. Tinelli. Proof certificates for SMT-based model checkers for infinite-state systems. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design, FMCAD ’16*, pages 117–124, 2016. doi:10.1109/FMCAD.2016.7886669.
- 17 Stefan Schwendimann. A new one-pass tableau calculus for pctl. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 277–291. Springer, 1998. doi:10.1007/3-540-69778-0\_28.
- 18 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.
- 19 P. Wolper. The tableau method for temporal logic: An overview. *Logique Et Analyse*, 28(110-111):119–136, 1985.

## A Interpretation of CTL-type Logics Over Cyclic Structures

In this appendix we define cyclic models and discuss their ability to characterise satisfiability in branching temporal logics.

► **Definition 20** (Cyclic Sequence, Cyclic Path and Cyclic Kripke structure). *Let  $z$  be a finite sequence of states  $z = s_0, s_1, \dots, s_j$  such that, for every  $0 \leq k < j$ ,  $(s_k, s_{k+1}) \in R$ . Then,  $z$  is cyclic iff there exists  $s_i$ ,  $0 \leq i \leq j$  such that  $(s_j, s_i) \in R$ . Let  $z$  be a finite cyclic sequence, the subsequence  $s_i, \dots, s_j$  of  $z$  is called a loop and  $s_i$  is called the cycling element. We denote the loop as  $\langle s_i, \dots, s_j \rangle^\omega$ . A cyclic path over  $z$  is an infinite sequence  $\text{path}(z) = s_0, s_1, \dots, s_{i-1} \langle s_i, s_{i+1}, \dots, s_j \rangle^\omega$ . A Kripke structure  $\mathcal{K}$  is cyclic if every fullpath is a cyclic path over a cyclic sequence of states.*

Cyclic paths are also known as *ultimately periodic* paths.

The fact that CTL (ECTL) satisfiability can be reduced to the interpretation over cyclic models only is derived from the existence of the finite model property [9], see also [14]. Hence, for any CTL (ECTL) formula  $\varphi$ , such that  $\text{Mod}(\varphi) \neq \emptyset$ , there always exists a model  $\mathcal{K} \in \text{Mod}(\varphi)$  such that  $\mathcal{K}$  is cyclic. Therefore, when speaking about the satisfiability in CTL (hence ECTL) we can consider *cyclic* Kripke structures.



## B Soundness and Completeness

Since CTL and ECTL are sublogics of ECTL<sup>#</sup> and the tableau method presented here is the adaptation of the method in [4], in this section we essentially adapt to CTL the soundness and completeness proofs developed in [4]. We firstly prove the soundness and completeness of the tableau method for CTL, and then we extend both results to ECTL.

To prove the soundness of our tableau method for CTL (Theorem 22), we show that every tableau rule in Figures 1, 2 and 4 are sound (or preserve satisfiability) in the sense of the next Lemma 21.

► **Lemma 21** (Soundness of the Tableau Rules for CTL). *Consider all the rules in Figures 1, and 2 and 4.*

1. For any  $\alpha$ -rule  $\frac{\Sigma}{\Sigma'}$  :  $Sat(\Sigma)$  if and only if  $Sat(\Sigma')$ .
2. For any  $\beta$ - and  $\beta^+$ -rule  $\frac{\Sigma}{\Sigma_1|\Sigma_2}$  :  $Sat(\Sigma)$  if and only if  $Sat(\Sigma_1)$  or  $Sat(\Sigma_2)$ .
3. If  $\Sigma$  is a set of consistent literals, then  $Sat(\Sigma, A\circ\sigma_1, \dots, A\circ\sigma_\ell, E\circ\sigma'_1, \dots, E\circ\sigma'_k)$  if and only if  $Sat(\sigma_1, \dots, \sigma_\ell, \sigma'_i)$  for all  $1 \leq i \leq k$ .

**Proof.** All the items follows very easily by the “systematic” application of the semantic definitions of the modalities, except the “if” direction for the two of  $\beta^+$ -rules. We will prove here the “if” direction of the rules  $(QU)^+$  for  $Q = E$  and  $Q = A$ , because the rules  $(Q\Diamond)^+$  are particular cases by abbreviation  $\Diamond\sigma = \tau\mathcal{U}\sigma$ .

For the “if” direction of rule  $(EU)^+$ , let  $\mathcal{K} \models \Sigma, E(\sigma_1\mathcal{U}\sigma_2)$  and let  $x$  be the path in  $\mathcal{K}$  such that  $\mathcal{K}, x, i \models \Sigma, E(\sigma_1\mathcal{U}\sigma_2)$ . Then, let  $j$  be the least  $i \geq 0$  such that  $\mathcal{K}, x, i \models \sigma_2$ . If  $j = i = 0$ , then  $\mathcal{K}, x, 0 \models \Sigma, \sigma_2$ . Otherwise, if  $j > 0$  then  $\mathcal{K}, x, m \models \sigma_1$ , for all  $0 \leq m < j$ . Consider  $k$  to be the greatest such  $m$  for which  $\mathcal{K}, x, k \models \Sigma$ . Hence,  $\mathcal{K}, x, h \models \sim\Sigma$ , for all  $h$  such that  $k + 1 \leq h < j$ . In particular, by definition of  $\Sigma'$  (obtained from  $\Sigma$ ) it is easy to see that  $\mathcal{K}, x, h \models \sigma$  for every  $\sigma \in (\Sigma \setminus \Sigma')$  and for all  $h$  such that  $0 \leq h < j$ . Therefore,  $\mathcal{K}, x, h \models \sim\Sigma'$ , for all  $h$  such that  $k + 1 \leq h < j$ . Thus,  $\mathcal{K}, x, k \models \sigma_1, E\circ E((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)$ .

For the “if” direction of rule  $(AU)^+$ , let us suppose that

$$\text{UnSat}(\Sigma, \sigma_2) \text{ and } \text{UnSat}(\Sigma, \sigma_1, A\circ A((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)).$$

We will show that  $\text{UnSat}(\Sigma, A(\sigma_1\mathcal{U}\sigma_2))$ . For that, let us consider any arbitrary  $\mathcal{K}$  such that  $\mathcal{K} \models \Sigma$  to show that  $\mathcal{K} \not\models A(\sigma_1\mathcal{U}\sigma_2)$ . By the above unsatisfiability hypothesis, if  $\mathcal{K} \models \Sigma$ , then both  $\mathcal{K} \not\models \sigma_2$  and  $\mathcal{K} \not\models \sigma_1 \wedge A\circ A((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)$ . Then, there are two possible cases. First, if  $\mathcal{K} \models \neg\sigma_1 \wedge \neg\sigma_2$ , then it is obvious that  $\mathcal{K} \not\models A(\sigma_1\mathcal{U}\sigma_2)$ . Second, if  $\mathcal{K} \models \neg\sigma_2 \wedge \neg A\circ A((\sigma_1 \wedge \sim\Sigma')\mathcal{U}\sigma_2)$ , then there exists  $x_1 \in \text{fullpaths}(\mathcal{K})$  and  $i_1 > 0$  that satisfy both  $\mathcal{K}, x_1, j \models \neg\sigma_2$  for all  $j$  such that  $0 \leq j \leq i_1$ , and  $\mathcal{K}, x_1, i_1 \models \neg\sigma_1 \vee \Sigma'$ . Since all the formulae in  $\Sigma \setminus \Sigma'$  are satisfied in all states along all paths, indeed  $\mathcal{K}, x_1, i_1 \models \neg\sigma_1 \vee \Sigma$ . Therefore, if  $\mathcal{K}, x_1, i_1 \models \neg\sigma_1$ , then obviously  $\mathcal{K} \not\models A(\sigma_1\mathcal{U}\sigma_2)$ . Otherwise, if  $\mathcal{K}, x_1, i_1 \models \Sigma$ , applying the same reasoning for  $\mathcal{K} \upharpoonright x_1(i_1)$  as we did above for  $\mathcal{K}$ , we can conclude that there should be a path  $x_2 \in \text{fullpaths}(\mathcal{K} \upharpoonright x_1(i_1))$  and some  $i_2 > 0$  such that either  $\mathcal{K} \upharpoonright x_1(i_1), x_2, j \models \neg\sigma_2$  for all  $j$  such that  $i_1 \leq j \leq i_2$  and  $\mathcal{K} \upharpoonright x_1(i_1), x_2, i_2 \models \neg\sigma_1 \vee \Sigma$ . Hence, if  $\mathcal{K} \upharpoonright x_1(i_1), x_2, i_1 \models \neg\sigma_1$ , then trivially  $\mathcal{K} \not\models A(\sigma_1\mathcal{U}\sigma_2)$ . Otherwise,  $\mathcal{K} \upharpoonright x_1(i_1), x_2, i_1 \models \Sigma$ . Hence, there are two possible scenarios: 1.) After a finite number of iterations we get a path  $y = x_1^{\leq i_1} x_2^{\leq i_2} \dots x_k^{\leq i_k}$  such that  $\mathcal{K}, y, j \models \neg\sigma_2$  for all  $j$  such that  $0 \leq j \leq i_k$  and  $\mathcal{K}, y, i_k \models \neg\sigma_1$ . 2.) The infinite iteration of the second case yields a path  $y = x_1^{\leq i_1} x_2^{\leq i_2} \dots x_k^{\leq i_k} \dots$  (that exists by the limit closure property) such that  $\mathcal{K}, y, i \models \neg\sigma_2$  for all  $i \geq 0$ . In both scenarios we have  $\mathcal{K} \not\models A(\sigma_1\mathcal{U}\sigma_2)$  holds for any arbitrary  $\mathcal{K}$  that satisfies  $\Sigma$ . Thus,  $\text{UnSat}(\Sigma, A(\sigma_1\mathcal{U}\sigma_2))$ . ◀



► **Theorem 22** (Soundness of the Tableau Method for CTL). *Given any set of state formulae  $\Sigma$ , if there exists a closed tableau for  $\Sigma$  then  $\text{UnSat}(\Sigma)$ .*

**Proof.** In a closed tableau for  $\Sigma$ , the set of formulae labelling at least one leaf in each bunch is inconsistent and therefore unsatisfiable. Then, by Lemma 21, the labelling of the root node,  $\Sigma$ , is unsatisfiable. ◀

Next, we prove the refutational completeness of the tableau method for CTL (Theorem 29). For that, we firstly define the notion of stage and prove some auxiliary properties on the stages and bunches of the systematic tableau, that are necessary to prove that every open bunch in the systematic tableau represents a model of the initial set of formulae (Lemma 28).

► **Definition 23** (Stage). *Given a branch,  $b$  of a tableau  $T$ , a stage in  $T$  is every maximal subsequence of successive nodes  $n_i, n_{i+1}, \dots, n_j$  in  $b$  such that  $\tau(n_k)$  is not a  $(Q\circ)$ -child of  $\tau(n_{k-1})$ , for all  $k$  such that  $i < k \leq j$ . We denote by  $\text{stages}(b)$  the sequence of all stages of  $b$ . The successor relation on  $\text{stages}(b)$  is induced by the successor relation on  $b$ . The labelling function  $\tau$  is extended to stages as the union of the original  $\tau$  applied to every node in a stage.*

► **Definition 24** ( $\alpha\beta^+$ -saturated Stage). *We say that a stage  $s = n_i, \dots, n_j$  in  $\mathcal{A}_\Sigma^{\text{sys}}$  is  $\alpha\beta^+$ -saturated if and only if it satisfies the following conditions:*

1. For all  $\sigma_1 \wedge \sigma_2 \in \tau(s)$ :  $\{\sigma_1, \sigma_2\} \subseteq \tau(s)$ .
2. For all  $Q\Box\sigma \in \tau(s)$ :  $\{\sigma, Q\Box\sigma\} \subseteq \tau(s)$ .
3. For all  $\sigma_1 \vee \sigma_2 \in \tau(s)$ :  $\sigma_1 \in \tau(s)$  or  $\sigma_2 \in \tau(s)$ .
4. For all  $Q(\sigma_1 \mathcal{R} \sigma_2) \in \tau(s)$ :  $\{\sigma_1, \sigma_2\} \subseteq \tau(s)$  or  $\{\sigma_2, Q\Box(Q(\sigma_1 \mathcal{R} \sigma_2))\} \subseteq \tau(s)$ .
5. For all  $Q(\sigma_1 \mathcal{U} \sigma_2) \in \tau(s)$ :  $\sigma_2 \in \tau(s)$  or  $\{\sigma_1, Q\Box(Q(\sigma_1 \mathcal{U} \sigma_2))\} \subseteq \tau(s)$  or  $\{\sigma_1, Q\Box(Q((\sigma_1 \wedge \sim \Sigma') \mathcal{U} \sigma_2))\} \subseteq \tau(s)$   
where  $\Sigma' = (\tau(n_i) \setminus \{Q(\sigma_1 \mathcal{U} \sigma_2)\}) \setminus \{(A\circ)^i A\Box\varphi \mid i \geq 0 \text{ and } (A\circ)^i A\Box\varphi \in \tau(n_i)\}$ .
6. For all  $Q(\diamond\sigma) \in \tau(s)$ :  $\sigma \in \tau(s)$  or  $\{Q\Box(Q(\diamond\sigma))\} \subseteq \tau(s)$  or  $\{Q\Box(Q((\sim \Sigma') \mathcal{U} \sigma))\} \subseteq \tau(s)$   
where  $\Sigma' = (\tau(n_i) \setminus \{Q(\diamond\sigma)\}) \setminus \{(A\circ)^i A\Box\varphi \mid i \geq 0 \text{ and } (A\circ)^i A\Box\varphi \in \tau(n_i)\}$ .

► **Definition 25** (Expanded Bunch and Tableau). *An open branch  $b$  is expanded if each stage  $s \in \text{stages}(b)$  is  $\alpha\beta^+$ -saturated and  $b$  is eventuality-covered. A bunch is expanded if all its open branches are expanded. A tableau is expanded if all its open bunches are expanded.*

The construction of the systematic tableau applies exactly one  $\beta^+$ -rule to exactly one selected eventuality (if any) at the first node of the stage, and then applies exhaustively all the applicable  $\alpha$ - and  $\beta$ -rules to the formulas in the stage, until the branch closes, or its leaf is labelled by an elementary set, or it contains a loop-node. Consequently, the following Proposition 26 holds which can be trivially proved by construction.

► **Proposition 26.** *Given any set of state formulae  $\Sigma$ , the systematic tableau  $\mathcal{A}_\Sigma^{\text{sys}}$  is expanded.*

Next we prove a crucial property of the systematic tableau management of eventualities by means of the selection policy.

► **Proposition 27.** *Let  $b$  be an open branch of  $\mathcal{A}_\Sigma^{\text{sys}}$  and let  $Q(\sigma_1 \mathcal{U} \sigma_2)$  be a formula that is selected at some stage  $s_i \in \text{stages}(b)$ . Then, there exists some stage  $s_k \in \text{stages}(b)$  (for some  $k \geq i$ ) such that  $\sigma_2 \in \tau(s_k)$  and  $\sigma_1 \in \tau(s_j)$  for all  $j \in \{i, \dots, k-1\}$ .*

**Proof.** By construction, the uniform set labelling the first node at each stage  $s_j$  ( $j \geq i$ ) of  $b$  has the form  $\Sigma_{s_j}, Q((\sigma_1 \wedge (\sim \Sigma_{s_i} \wedge \dots \wedge \sim \Sigma_{s_{j-1}})) \mathcal{U} \sigma_2)$  where each  $\Sigma_{s_j}$  is the context of the selected formula containing the next-step variant of  $Q(\sigma_1 \mathcal{U} \sigma_2)$  at the first node of each stage  $s_j$ . Since

## 14:18 One-Pass Context-Based Tableaux Systems for CTL and ECTL

no other  $\beta^+$ -rule is applied each  $\Sigma_{s_j}$  is a subset of the finite set formed by all state formulae that are subformulae of some formula in  $\Sigma_{s_i}$  and their negations. Hence, there are a finite number of different  $\Sigma_{s_j}$ . Therefore, after finitely many applications of the  $\beta^+$ -rule,  $\Sigma_{s_h} = \Sigma_{s_j}$ , for some  $h \geq i$ , for some  $j \in \{i, \dots, h-1\}$ , and  $\sigma_1 \wedge (\sim \Sigma_{s_i} \wedge \dots \wedge \sim \Sigma_{s_{h-1}}) \in \tau(s_h)$ . In particular,  $\sim \Sigma_{s_h} \in \tau(s_h)$ , hence,  $\Sigma_{s_h}$  must be inconsistent. Since  $b$  is open, this is a contradiction. This means that, for some  $k \geq i$  the application of the corresponding  $\beta^+$ -rule should force that  $\sigma_2 \in \tau(s_k)$ . In addition, by Proposition 26 and Definition 24(5),  $\sigma_1 \in \tau(s_j)$  for all  $j \in \{i, \dots, k-1\}$ .  $\blacktriangleleft$

► **Lemma 28** (Model Existence). *Let  $\Sigma$  be any set of formulae. For any expanded bunch  $H$  of  $\mathcal{A}_\Sigma^{sys}$ , there exists a Kripke structure  $\mathcal{K}_H$  such that  $\mathcal{K}_H \models \Sigma$ .*

**Proof.** Let  $H$  be any expanded bunch of  $\mathcal{A}_\Sigma^{sys}$ . We define  $\mathcal{K}_H = (S, R, L)$  such that  $S = \bigcup_{b \in H} \text{stages}(b)$  and for any  $s \in S$ :  $L(s) = \{p \mid p \in \tau(n) \cap \text{Prop for some node } n \in s\}$ ; and  $R$  is the relation induced in  $\text{stages}(b)$  for each  $b \in H$ . Any branch in  $b \in H$  is open, hence  $b$  ends in a loop-node. Moreover, every eventuality has been selected in some stage of  $b$ . Hence, there exists a (possibly empty) uniform set  $\Sigma_\ell$  such that for some  $i \geq 0$ :  $b = s_0, s_1, \dots, s_{i-1}, s_i, s_{i+1}, \dots, s_j, n_\ell$ , where each  $s_h$  stands for a stage and  $n_\ell$  is a non-expandable loop-node labelled by  $\Sigma_\ell$  whose companion node is the first node at stage  $s_i$ . We are going to prove the following fact:

$$\mathcal{K}_H, s_a, 0 \models \sigma \text{ for any } a \in \{0, \dots, j\} \text{ and any formula } \sigma \text{ in } L(s_a)$$

by structural induction on the formula  $\sigma$ .

The base of the induction, for  $\sigma = p \in \text{Prop}$ , follows by definition of  $\mathcal{K}_H$ .

The cases where  $\sigma$  has one of the forms  $\sigma_1 \wedge \sigma_2$ ,  $\mathbf{Q}\Box\sigma$ ,  $\sigma_1 \vee \sigma_2$  and  $\mathbf{Q}(\sigma_1 \mathcal{R} \sigma_2)$  are trivial by Definition 24 and the induction hypothesis. Hence, to complete the inductive proof we will show that  $\mathcal{K}_H, s_a, 0 \models \mathbf{Q}(\sigma_1 \mathcal{U} \sigma_2)$  for any  $\mathbf{Q}(\sigma_1 \mathcal{U} \sigma_2) \in L(s_a)$ . The case for all  $\mathbf{Q}\diamond\sigma \in L(s_a)$  follows as a particular case by  $\diamond\sigma \equiv \mathbf{T}\mathcal{U}\sigma$ .

Consider any  $\mathbf{Q}(\sigma_1 \mathcal{U} \sigma_2) \in L(s_a)$ . Since  $b$  is eventuality-covered and  $n_\ell$  is a loop-node,  $\mathbf{Q}(\sigma_1 \mathcal{U} \sigma_2)$  must be the selected eventuality at some node between the states  $s_a$  and  $s_j$ . Hence, by Proposition 27 and the definition of  $\mathcal{K}_H$ , there should be a state  $s_k \in S$  (for some  $a \leq k \leq j$ ) such that  $\sigma_2 \in L(s_k)$  and  $\sigma_1 \in L(s_z)$  for all  $z \in \{a, \dots, k-1\}$ . Then, by induction hypothesis,  $\mathcal{K}_H, s_k, 0 \models \sigma_2$  and  $\mathcal{K}_H, s_z, 0 \models \sigma_1$  for all  $z \in \{a, \dots, k-1\}$ . Therefore,  $\mathcal{K}_H, s_a, 0 \models \mathbf{Q}(\sigma_1 \mathcal{U} \sigma_2)$ .

To complete the proof, we show that the successor relation between states in  $\mathcal{K}_H$  is well-defined. For that, consider any tableau node in any stage  $s_a$  that is labelled by an elementary set

$$\{\Sigma, \mathbf{A}\circ\sigma_1, \dots, \mathbf{A}\circ\sigma_n, \mathbf{E}\circ\sigma'_1, \dots, \mathbf{E}\circ\sigma'_k\}$$

where  $\Sigma$  is a consistent set of literals, by rule (Q $\circ$ ),  $s_a$  has (in  $\mathcal{K}_H$ ) a successor state  $s_{a+1}^i$ , for each  $i \in \{1, \dots, k\}$ , such that  $L(s_{a+1}^i) = \{\sigma_1, \dots, \sigma_n, \sigma'_i\}$ . We can assume (by the above proved fact) that  $\mathcal{K}_H, s_{a+1}^i, 0 \models \{\sigma_1, \dots, \sigma_n, \sigma'_i\}$  for all  $i \in \{1, \dots, k\}$ . Therefore, we can infer that  $\mathcal{K}_H, s_a, 0 \models \{\Sigma, \mathbf{A}\circ\sigma_1, \dots, \mathbf{A}\circ\sigma_n, \mathbf{E}\circ\sigma'_1, \dots, \mathbf{E}\circ\sigma'_k\}$ .  $\blacktriangleleft$

Next, we prove the refutational completeness of the tableau method.

► **Theorem 29** (Refutational Completeness for CTL). *For any set of state formulae  $\Sigma$ , if  $\text{UnSat}(\Sigma)$  then there exists a closed tableau for  $\Sigma$ .*

**Proof.** Suppose the contrary, that there exists no closed tableau for  $\Sigma$ . Then the systematic tableau  $\mathcal{A}_\Sigma^{sys}$  is open. Hence, there is at least one expanded bunch  $H$  in  $\mathcal{A}_\Sigma^{sys}$ . By Lemma 28, there exists a Kripke structure  $\mathcal{K}_H$  such that  $\mathcal{K}_H \models \Sigma$ . Consequently,  $\text{Sat}(\Sigma)$ . ◀

Finally, we prove the completeness of our tableau method for CTL.

► **Theorem 30** (Termination of the Tableau Method for CTL). *For any set of state formulae  $\Sigma$ , the construction of the expanded tableau  $\mathcal{A}_\Sigma^{sys}$  terminates.*

**Proof.** Tableau rules produce a finite branching, hence König’s Lemma, applies. Therefore, it suffices to prove that every branch is finite. By Proposition 27, the application of a  $\beta^+$ -rule to a selected formula stops after a finite number of steps. Since the number of selectable eventualities in any open branch is finite, any open branch is eventuality-covered after a finite number of eventuality selections. Recall that we assume the eventuality selection strategy to be fair. ◀

► **Theorem 31** (Completeness of the Tableau Method for CTL). *For any set of state formulae  $\Sigma$ , if  $\Sigma$  is satisfiable then there exists a (finite) open expanded tableau for  $\Sigma$ .*

**Proof.** The existence of the systematic tableau  $\mathcal{A}_\Sigma^{sys}$  suffices to prove this fact, by Theorem 30. ◀

Now, we explain how the proofs of these metatheorems for CTL can be extended to ECTL. Firstly, we extend the soundness of the tableau rules, in the sense of Lemma 21, to the rules in Figure 7.

► **Lemma 32.** *For any ECTL set of state formulae  $\Sigma$  and any state formula  $\sigma$ :*

1.  $\text{Sat}(\Sigma, Q\Box\Diamond\sigma)$  if and only if  $\text{Sat}(\Sigma, Q\Diamond\sigma, Q\circ Q\Box\Diamond\sigma)$ .
2.  $\text{Sat}(\Sigma, Q\Diamond\Box\sigma)$  if and only if  $\text{Sat}(\Sigma, Q\Box\sigma)$  or  $\text{Sat}(\Sigma, Q\Diamond\sigma, Q\circ Q\Diamond\Box\sigma)$ .

**Proof.** It follows by “systematic” application of the semantic definitions of the modalities  $Q\Box\Diamond$  and  $Q\Diamond\Box$  given by the equivalences (8) in Section 5. ◀

To extend the refutational completeness result to ECTL, we firstly extend the Definition 24 with the following additional conditions for a stage to be  $\alpha\beta^+$ -saturated:

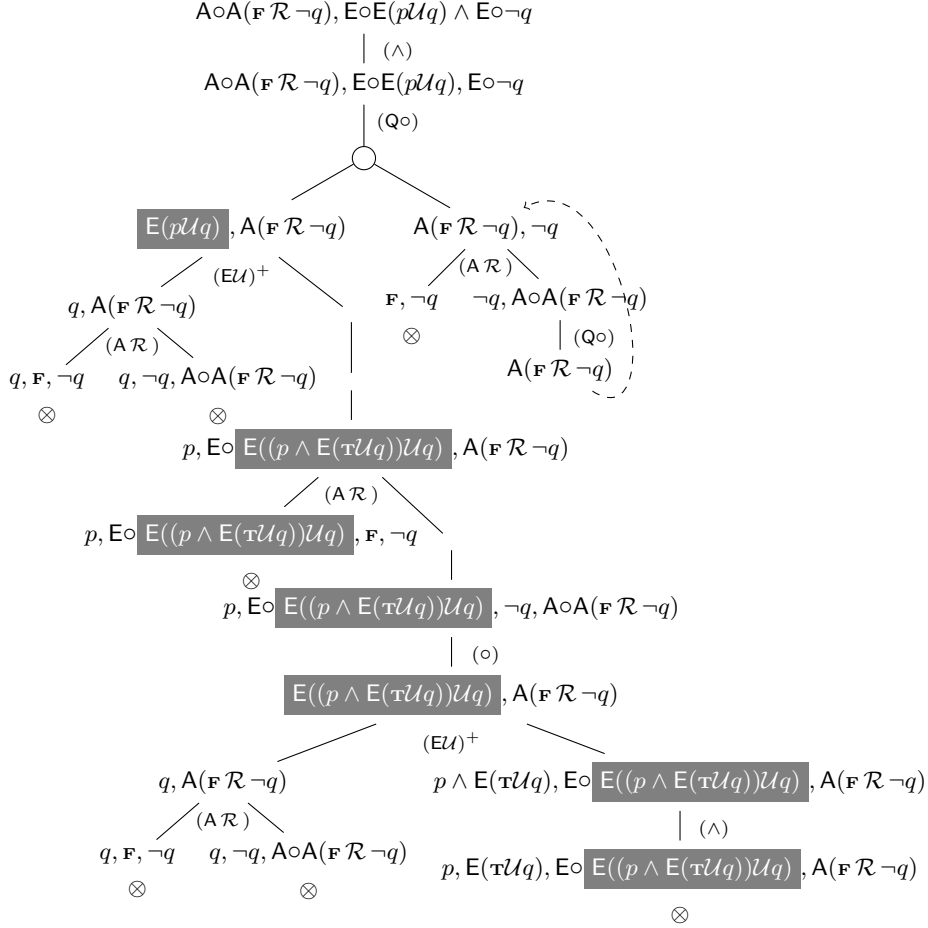
7. For all  $Q\Box\Diamond\sigma \in \tau(s)$ :  $\{Q\Diamond\sigma, Q\circ Q\Box\Diamond\sigma\} \subseteq \tau(s)$ .
8. For all  $Q\Diamond\Box\sigma \in \tau(s)$ :  $\{Q\Box\sigma\} \subseteq \tau(s)$  or  $\{Q\Diamond\sigma, Q\circ Q\Diamond\Box\sigma\} \subseteq \tau(s)$ .

It is obvious that these two additional conditions are satisfied in any stage of the systematic tableau by construction. Using these conditions, it is routine to prove that  $\mathcal{K}_H$  defined in Lemma 28 satisfies the fact that:  $\mathcal{K}_H, s_a, 0 \models \sigma$  for any  $a \in \{0, \dots, j\}$  and any formula of the forms  $Q\Box\Diamond\sigma, Q\Diamond\Box\sigma$  that belongs to  $L(s_a)$ . Therefore, refutational completeness (i.e. Theorem 29) extends to ECTL.

Finally, to extend the termination result (see proof of Theorem 30), it suffices to ensure that the rules in Figure 7 do not affect the behaviour of the  $\beta^+$ -rules on the selected eventualities in the sense that Proposition 27 is preserved. Since each application of a rule in Figure 7 introduces a new  $Q\Diamond\sigma$ , in Section 5 we have introduced the simplification rule ( $\Box Q\mathcal{U}$ ) (see (9)) to subsume any occurrence of the eventuality  $\varphi$  by any next-step variant of  $\varphi$ . Therefore, Proposition 27 holds and hence Theorem 30 trivially extends to ECTL.

### C The Running Example Tableau

In Figure 9 we depict the whole tableau for the running example we use in the paper. Note that the Q rule at step 2, denoted with a big circle generates two AND-successors, where the left successor has the closed tableau - this is explained in the paper. Hence, the bunch is closed, in spite of the open tableau at the right successor of the AND-node.



■ **Figure 9** A closed tableau for  $\{A\circ A(\mathbf{F} \mathcal{R} \neg q), E\circ E(p\mathcal{U}q) \wedge E\circ \neg q\}$ .

# TESL: A Model with Metric Time for Modeling and Simulation

Hai Nguyen Van 

Université Paris-Saclay, CNRS, LRI, Orsay, France  
<https://perso.crans.org/nguyen-van>

Frédéric Boulanger 

Université Paris-Saclay, CNRS, LRI, CentraleSupélec, Orsay, France  
[frederic.boulanger@lri.fr](mailto:frederic.boulanger@lri.fr)

Burkhart Wolff

Université Paris-Saclay, CNRS, LRI, Orsay, France  
[burkhart.wolff@lri.fr](mailto:burkhart.wolff@lri.fr)

---

## Abstract

Real-time and distributed systems are increasingly finding their way into critical embedded systems. On one side, computations need to be achieved within specific time constraints. On the other side, computations may be spread among various units which are not necessarily sharing a global clock. Our study is focused on a specification language – named TESL – used for coordinating concurrent models with timed constraints. We explore various questions related to time when modeling systems, and aim at showing that TESL can be introduced as a reasonable balance of expressiveness and decidability to tackle issues in complex systems. This paper introduces (1) an overview of the TESL language and its main properties (polychrony, stutter-invariance, coinduction for simulation), (2) extensions to the language and their applications.

**2012 ACM Subject Classification** Theory of computation → Timed and hybrid models

**Keywords and phrases** Timed Systems, Semantics, Models, Simulation

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.15

**Supplementary Material** Artifacts and source code available at [github.com/heron-solver/heron](https://github.com/heron-solver/heron).

## 1 Introduction

Designing and modeling systems nowadays still raise open problems. A very expressive language or framework can be useful to model a complex system where events are not trivially interleaved. On the opposite, an excessively expressive language is the reason for prohibitive slow-downs or even undecidability. As such, a reasonable balance between expressiveness and decidability needs to be found. In the current industrial trend for critical embedded systems, grows an increasing need for two kinds of systems:

- *Real-Time Systems* where an external input is followed by an output delivered within a specified time, named *deadline*. The correct behavior of such systems must be ensured at both logical and temporal levels.
- *Distributed Systems* where autonomous nodes communicate and cooperate to perform a common computation.

A distributed real-time system (DRTS) [34, 14] belongs to both categories and consists in autonomous computing nodes where specific timing constraints must be met. DRTS are essential as they describe more closely common real-time applications by providing fault tolerance and load sharing [35, 34, 14]. An example of a DTRTS is a modern car using CAN buses [14]. In such a setting, a middle gateway connects two CAN buses. One of them is high-speed and connects the engine, the suspension and the gearbox control. The other one



© Hai Nguyen Van, Frédéric Boulanger, and Burkhardt Wolff;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics



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

is low-speed and connects the lights, seat and door control units. The aviation industry also exhibits an increasing need for DTRS as shown by recent developments in interoperable gateways ED-247 [21].

On the side of formal modeling, various environments have emerged to tackle the issue of modeling and verifying complex systems. Some are industrial products, such as Matlab/Simulink [15], Wolfram SystemModeler [33], SCADE [7]. Some others are academic experiments, such as Ptolemy II [13], TimeSquare [12], ModHel’X [20]. Our study is centered around the inner formalisms that drive these environments, and in particular the TESL language. The main question this paper addresses is: *Can we provide a uniform framework to model distributed and real-time systems?* The paper is organized as follows: Section 2 introduces the TESL language which we believe can answer the main problem. Section 3 introduces its main properties, in terms of polychronous clocks, stutter-invariance and coinductive unfolding. Finally, in Section 4 we present some extensions and aim at showing their relevance in the scope we address.

## 2 The TESL language

The Tagged Events Specification Language (TESL) [8] originates from the idea of coordinating the execution of heterogeneous inner-parts of a model as components of the ModHel’X modeling and simulation environment. The language is inspired by CCSL [16, 26], the Tagged Signal Model [25] and from the constructive semantics of Esterel [6, 5] for the original simulation solver. In this setting, an event is modeled by a *clock*, with an associated time scale. Considering a continuous system, its behavior is discretized into a sequence of observation instants. At each instant, a clock admits a *timestamp* (also called *tag*), that stands for the metric time measured on this clock. Besides, a clock also admits a *tick* which indicates an occurrence of the event at this instant. The domain for timestamps can possibly be any totally ordered set. We emphasize the fact that the language handles chronometric time constraints, which are different from logical time constraints. Chronometric time constraints are given on durations measured between timestamps. Two forms of constraints may be specified in TESL:

- *Event-triggered causality.* Events may occur due to the occurrence of other events. For instance “I have a coffee because my office mate prepares some coffee”.
- *Time-triggered causality.* Events may occur because a time threshold has been reached. For instance “I have a coffee because it is 9am”.

### 2.1 Illustrating the Language

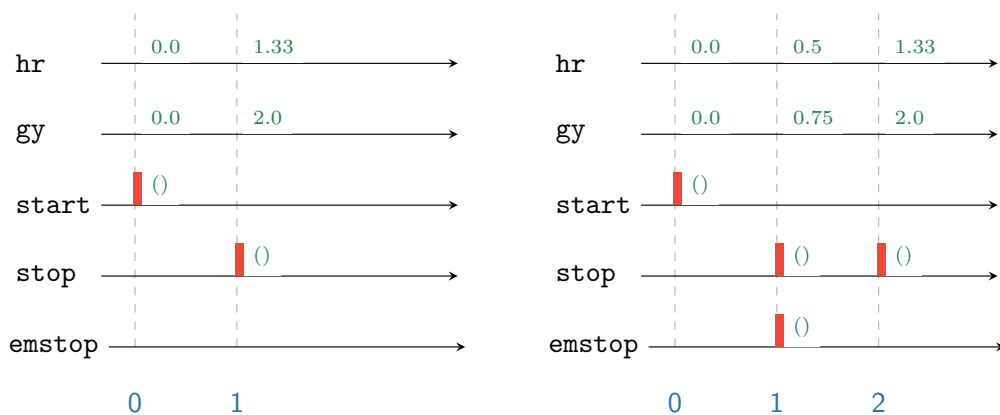
Let us model in TESL the simple behavior of a radiotherapy machine used in cancer treatment. The patient has a prescription of 2 Gy of radiation in low-dose-rate of  $1.5 \text{ Gy.h}^{-1}$ .

#### ■ Listing 1 Radiotherapy machine

```

1 rational-clock hr // Time unit in hours
2 rational-clock gy // Radiation unit in Gray
3 unit-clock start sporadic () // Start emitting rays
4 unit-clock stop // Stop emitting rays
5 unit-clock emstop // Emergency stop
6 time relation gy = 1.5 * hr
7 start time delayed by 2.0 on gy implies stop
8 emstop implies stop

```



(a) Normal situation.

(b) Emergency stop.

■ **Figure 1** Two partially satisfying runs.

Lines 1 to 5 declare clocks `hr` and `gy` with rational timestamps, and clocks `start`, `stop` and `emstop` with the unit timestamp (so there is no chronometric scale associated to them). The constraint `sporadic` enforces the occurrence of a tick on `start`. Line 6 specifies that time on `hr` flows 1.5 times as fast as on `gy`. Line 7 specifies that each time clock `start` ticks, clock `stop` will tick after a delay of 2.0 measured on the time scale of clock `gy`. Line 8 requires that each time the `emstop` clock ticks, the `stop` clock instantaneously ticks as well. The syntax of such expressions is detailed in Subsection 2.3.

Two behaviors are illustrated in Figure 1. They show possible execution traces or *runs* satisfying the TESL specification. A run consists in a sequence of synchronization *instants* (vertical dashed line with blue numbers). Each of them contains *ticks* (in red) along with *timestamps* (in green) on the time-scales of the clocks `hr`, `gy`, `start`, `stop` and `emstop`.

## 2.2 Clocks, runs and timestamps

► **Definition 1.** Let  $\mathbb{K}$  be the set of clocks,  $\mathbb{B}$  the set of booleans and  $\mathbb{T}$  the ordered domain of timestamps. The set of runs is denoted  $\Sigma^\infty$  and defined by

$$\Sigma^\infty = \mathbb{N} \rightarrow \mathbb{K} \rightarrow (\mathbb{B} \times \mathbb{T})$$

Additionally, we define two projections that extract the components of an event occurrence:

$\text{ticks}(\rho \ n \ K)$  *ticking predicate of clock  $K$  in run  $\rho$  at instant  $n$  (first projection)*

$\text{time}(\rho \ n \ K)$  *time value on clock  $K$  in run  $\rho$  at instant  $n$  (second projection)*

► **Example 2.** Let  $\rho_{\text{Fig.1a}}$  be the run shown in Figure 1a, we have  $\text{ticks}(\rho_{\text{Fig.1a}} \ 0 \ \text{start}) = \text{true}$  and  $\text{time}(\rho_{\text{Fig.1a}} \ 1 \ \text{gy}) = 2.0$ .

## 2.3 Quick overview of the syntax

We briefly introduce some expressions of the language which serve the purpose of this paper. The reader may refer to the official website of TESL<sup>1</sup> for an exhaustive description of all the features of the language. A TESL specification  $\Phi$  is described by the following grammar:

<sup>1</sup> <https://wdi.centralesupelec.fr/software/TESL/>



$$\begin{aligned}
\Phi & ::= \langle atom \rangle \wedge \dots \wedge \langle atom \rangle \\
\langle atom \rangle & ::= \langle clock \rangle \text{ sporadic } \langle timestamp \rangle \text{ on } \langle clock \rangle \\
& \quad | \langle clock \rangle \text{ implies } \langle clock \rangle \\
& \quad | \text{ time relation } (\langle clock \rangle, \langle clock \rangle) \in \langle relation \rangle \\
& \quad | \langle clock \rangle \text{ time delayed by } \langle duration \rangle \text{ on } \langle clock \rangle \text{ implies } \langle clock \rangle
\end{aligned}$$

where  $\langle clock \rangle \in \mathbb{K}$ ,  $\langle timestamp \rangle \in \mathbb{T}$ ,  $\langle duration \rangle \in \mathbb{T}$  and  $\langle relation \rangle \subseteq \mathbb{T} \times \mathbb{T}$ .

To provide a quick understanding, we briefly and informally explain the semantics:

- $K \text{ sporadic } \tau \text{ on } K_{\text{meas}}$  requires a tick on clock  $K$  at an instant where the timestamp on  $K_{\text{meas}}$  is  $\tau$  ;
- $K_{\text{master}} \text{ implies } K_{\text{slave}}$  models instantaneous causality by specifying that at each instant where  $K_{\text{master}}$  ticks,  $K_{\text{slave}}$  ticks as well ;
- $\text{time relation } (K_1, K_2) \in R$  relates the time frames of clocks  $K_1$  and  $K_2$  by specifying that at each instant, the timestamps on  $K_1$  and  $K_2$  have to be in relation  $R$  ;
- $K_{\text{master}} \text{ time delayed by } \delta\tau \text{ on } K_{\text{meas}} \text{ implies } K_{\text{slave}}$  stands for delayed causality by duration. At each instant  $k$  where  $K_{\text{master}}$  ticks, it requires a tick on  $K_{\text{slave}}$  at an instant where the timestamp on  $K_{\text{meas}}$  is  $\tau'$ , with  $\tau'$  the sum of  $\delta\tau$  and the timestamp on  $K_{\text{meas}}$  at instant  $k$ . In other words, it states that each tick on  $K_{\text{master}}$  must be followed by a tick on  $K_{\text{slave}}$  after a delay  $\delta\tau$  measured on the time scale of  $K_{\text{meas}}$ .

### 3 Properties of the language

#### 3.1 Polychronous clocks and time islands

One of the most prominent properties of the TESL language lies in *polychronous clocks* [23], a *global clock* does not necessarily drive the system. In the context of distributed systems, there exists as many clocks as there are computing nodes: all run at different rates and their clocks may possibly drift along. This is why, an additional mechanism of *synchronization* is necessary to coordinate these subworkers to achieve a common desired computation.

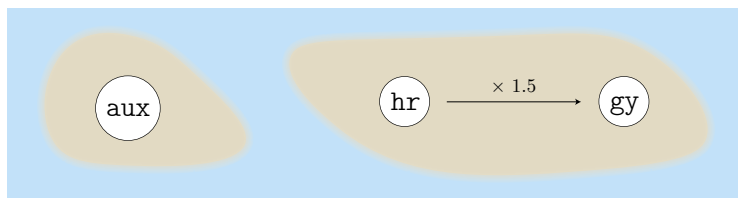
- *Metric level.* There are similarities with time dilation as in *special relativity* [19] where time seems to flow more slowly for a stationary observer than for a moving observer. The drift increases with the speed of the moving observer. For instance, GPS satellites suffer from time drifting and it is necessary to take into account these effects.
- *Temporal level.* Modern computing also exhibits this idea where temporal cycles may speed up or slow down. Current predominant processors adjust their clock speed with respect to environmental variables (energy, heat, noise), this is called *throttling*. Today's multicore processors consist of multiple computing units which may run faster or slower for these reasons, while possibly being used to achieve a distributed computation.

We illustrate this statement with the running example by adding an independent computing unit used for auxiliary computation needs. Whenever its computation is finished, it will trigger an event to indicate that it is ready. Let us simply declare a clock `aux` whenever this computing unit yields its signal. Besides, we can also create a scenario where we require this to occur at timestamp 0.5. The following line can be added to the specification in Listing 1:

```
rational-clock aux sporadic 0.5
```

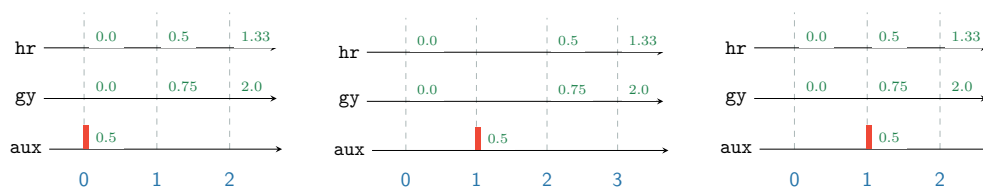


In this setting, clocks `hr` and `gy` are said to belong to the same *time island* as their timeframes are arithmetically related. On the other hand, clock `aux` belongs to another independent time island. There may also be other clocks living around as the specification is permissive and allows other clocks to exist even though they were not specified.



■ **Figure 2** Graphic representation of time islands.

Let us consider the specification in Listing 1, with the additional `aux` clock as declared above. Figure 3 depicts three runs which satisfy this specification. For presentation purposes, only three clocks `hr`, `gy` and `aux` are displayed. On the leftmost figure, we observe that `aux` ticks at 0.5 when it is 0.0 on `hr`. On the center figure, `aux` ticks at 0.5 when it is between 0.0 and 0.5 on `hr`. On the rightmost figure, `aux` ticks at 0.5 when it is 0.5 on `hr`. We see therefore that there exists an infinite number of satisfying runs as the timeframe on clock `aux` is left completely unrelated to the other time frames. However, we developed a simulation solver for TESL that supports symbolic runs, and hence captures this infinity of runs in a finite number of symbolic runs using symbolic timestamps.



■ **Figure 3** Examples of satisfying runs with additional clock `aux` in an independent time island.

### 3.2 Stutter Invariance

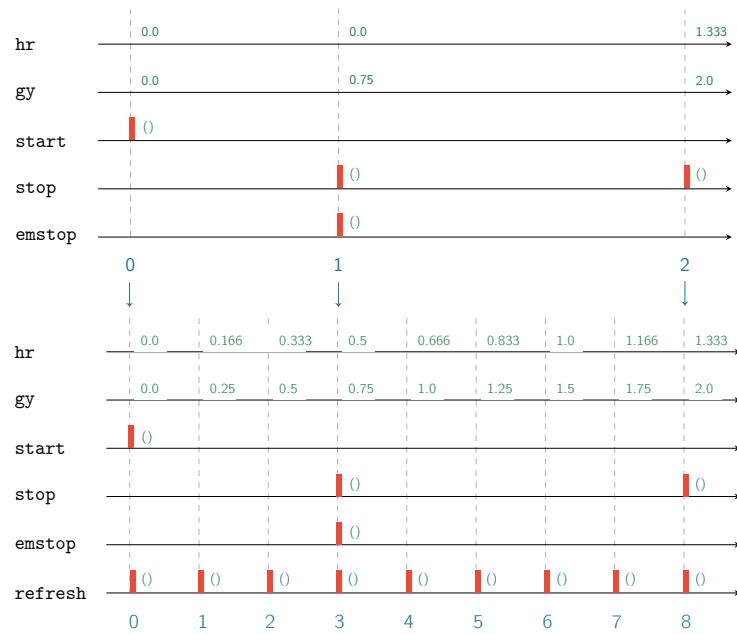
A fundamental concept of concurrent and distributed systems is *stutter invariance*. In finite-state model checking, it is an essential requirement for partial-order reduction techniques. When composing automata, the addition of stutter, or silent instants, allows the accommodation for their different alphabets. From a point of view in language theory, the membership of any word in a language shall be preserved even if a letter is duplicated. In our setting to model and compose submodels, we need stutter invariance in order to provide *compositionality*. For instance, when composing two specifications, we may have to add observation instants to a run that satisfies a specification in order to observe events on clocks that belong to the other specification. In other words, stuttering is necessary to refine specifications [22]. Stutter invariance also allows one to observe a model more often than necessary without changing its behavior.

In TESL, composing specifications is simply performed by the conjunction of TESL-formulae. To illustrate the idea of stutter-invariance with the running example, let us assume that we require the system to trigger some refresh mechanism every 10 minutes. We would add the following lines to the specification:

```
refresh sporadic 0.0 on hr
refresh time delayed by <10/60> on hr implies refresh
```

If we consider the run from Figure 1b and wish to compose it with this refreshing mechanism, a satisfying run is shown in Figure 4. The top of the figure shows the original run as in Figure 1b, whereas the bottom depicts a run where new instants have been added. A one-to-one correspondence is observed between run instants in the top and the bottom figure. Both runs exhibit the same first instant where `start` is triggered, with `refresh` additionally ticking in the second run. However, the second instant of the second run exists due to the refreshing requirement at 0.166 on clock `hr`, which is not present on top.

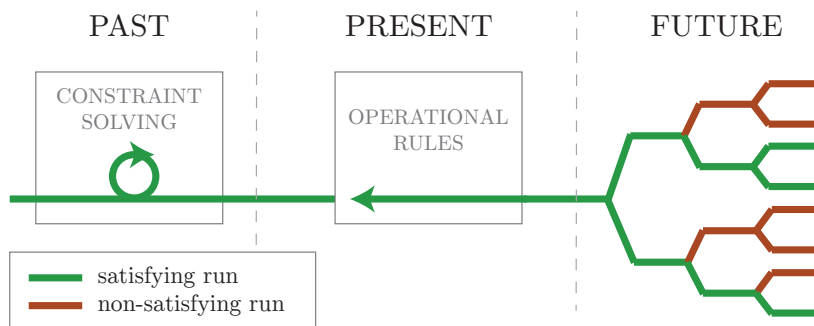
Stutter-invariance is illustrated by the fact that a run may be dilated and new instants added while still satisfying the specification.



■ **Figure 4** The example of radiotherapy run dilated.

### 3.3 Unfolding Specifications

The language allows the specification of runs that can be constructed and described by operational rules. In [29], we introduced an operational semantics of the language whose main ideas are summarized in Figure 5. The general concept of the operational semantics revolves around a 3-component pattern *past-present-future*. The past component contains the run we are constructing (which we also call the *run context*), the present component contains TESL-formulae to consume for the construction of the current instant, while the future component contains TESL-formulae to consume for future instants. The system considers each TESL formula as a consumable resource, and its consumption produces a “smaller” resource, which allows to constructively build the past component. Finally, the past component is a symbolic run and contains logical primitives which are sent to a SMT-solver in order to decide the satisfiability of the constructed run. Put differently, we reduced the problem of solving a TESL specification to a simpler constraint solving problem.



■ **Figure 5** Usage of the operational semantics.

## 4 Extensions

In this section, we propose two extensions of the language. From the original implementation of TESL, we have experimentally broadened its scope by adding two features on formulae and clocks. The addition of such has increased the language expressiveness without compromising constraint solving. To provide an insight, we illustrate them with an application example. We designed and experimented their semantics by implementing them into an experimental solver, named Heron<sup>2</sup> [29]. This implementation is a path-exhaustive multicore simulation solver built with MLton/MPL [36, 37]. It directly implements the operational semantics and the presented extensions. It can also be used for system testing and monitoring.

### 4.1 Precedence formula (and timed automata)

The first extension we propose is built around the precedence operator as found in CCSL. A appreciable motivation lies in modeling Synchronous Dataflows [24, 26]. In this model, each component provides an interface with inputs and outputs, and respectively a number of input tokens (to be read) and another of output tokens (to be written). When wiring two components, it is necessary that the  $n$ -th output writing event will precede the  $n$ -th input reading event. Precedence allows to specify this kind of indexed requirement over the order of event occurrence.

We extend the syntax of TESL as shown in Subsection 2.3 with

$$\begin{array}{lcl}
 \langle atom \rangle & ::= & \dots \\
 & | & \langle clock \rangle \text{ weakly precedes } \langle clock \rangle \\
 & | & \langle clock \rangle \text{ strictly precedes } \langle clock \rangle
 \end{array}$$

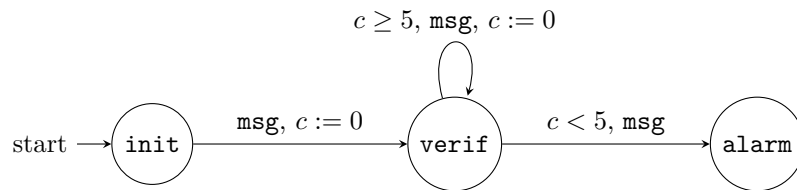
Informally,  $K_1$  **weakly precedes**  $K_2$  means that each tick on clock  $K_2$  may be uniquely mapped to a tick on  $K_1$  in the past or current instants (as a one-to-one correspondence).  $K_1$  **strictly precedes**  $K_2$  is analogous but maps to instants that are strictly in the past.

► **Remark 3.** Mallet *et al.* showed that the decidability of this type of formula could be handled with counter automata [27]. In our framework, we modeled this formula in a similar way by embedding run contexts with arithmetic constraints containing counter expressions. Again, we reduced this problem to a constraint solving problem.

<sup>2</sup> <https://github.com/heron-solver/heron>

## 15:8 TESL: A Model with Metric Time for Modeling and Simulation

To illustrate our interest in this operator, we consider timed automata [2, 1] as introduced by Alur and Dill. An additional and distinct mechanism made of clocks (also referred as *chronometers*) is used to store and specify metric timing constraints. On the implementation side, they extend classical finite-state automata with timing constraints. This formalism allows time to progress inside states while transitions are instantaneous, meaning that transitioning from one state to another is fast enough to be abstracted. In this subsection, we describe how this model of computation can be encoded with TESL extended with precedence. Let us give in Figure 6 a simple timed automaton (extracted from [4]) which models a system in which an alarm is triggered whenever the delay between receiving two messages is less than 5 seconds.



■ **Figure 6** An example of timed automata from [4].

To model the timed automaton in Figure 6, we declare TESL-clocks that will simulate the events occurring at a lower level (suffixed by `_enter` and `_leave`). Other clocks are also declared for transitions.

```

// Set of states: {init, verific, alarm}
unit-clock state_init_enter
unit-clock state_init_leave
unit-clock state_verif_enter
unit-clock state_verif_leave
unit-clock state_alarm_enter
unit-clock state_alarm_leave
  
```

We also need to declare TESL-clocks related to the behavior of TA-clocks, in particular when resetting them.

```

// Set of clocks: {c}
unit-clock c_reset
rational-clock c sporadic 0.0
  
```

Likewise, we need a TESL-clock to model the reading of a symbol (so-called *action*).

```

// Set of actions: {msg}
unit-clock read_msg
  
```

We proceed by encoding in TESL each transition of the timed automaton. We model the first transition from `init` to `verif`, which must read symbol `msg` and reset clock `c`, as:

```

// Transition t1 = init -> verific: msg, c := 0
state_init_leave when read_msg implies trigger_t1
trigger_t1 implies state_verif_enter
trigger_t1 implies c_reset
  
```

The second transition from `verif` to itself can be triggered when reading `msg` if time on clock `c` is greater than or equal to 5, which will eventually lead to resetting `c`. This means that the transition can be triggered if more than 5.0 units of time have elapsed on `c` since

the last time  $c$  has been reset. When using this transition, one will remain in state `verif` while resetting  $c$  to 0 each time a message has been read.

```
// Transition t2 = verf -> verf: c >= 5, msg, c := 0
c_reset time delayed by 5.0 on c with reset on trigger_t3
    implies trigger_t2_min
trigger_t2_min weakly precedes trigger_t2
state_verif_leave ^ read_msg implies trigger_t2 ∨ trigger_t3
trigger_t2 implies state_verif_enter
trigger_t2 implies c_reset
```

The third transition from `verif` to `alarm` is triggered when a new message has been received before 5.0 units of time have elapsed. We model this as:

```
// Transition t3 = verf -> alarm: c < 5, msg
c_reset time delayed by 5.0 on c with reset on trigger_t2
    implies trigger_t3_max
trigger_t3 strictly precedes trigger_t3_max
state_verif_leave ^ read_msg implies trigger_t2 ∨ trigger_t3
trigger_t3 implies state_alarm_enter
```

Figure 7 shows a run prefix exhibiting the behavior of our encoding of the timed automaton. At instant 0, time on clock  $c$  is 0.0 and we enter in state `init`. At instant 1, 5.0 units of time have elapsed. At instant 2, 5.0 additional units of time have elapsed and `read_msg` has been triggered, thus the transition is triggered (`trigger_t1`). The TA-clock  $c$  is reset and leaves state `init` to enter `verif`. Also, a minimum limit has been set on triggering transition  $t_2$  as it can only be fired after elapsing at least 5.0 units of time (as depicted by `trigger_t2_min` at instant 4). At instant 4, symbol `msg` is read and transition  $t_2$  is triggered to re-enter in the same state `verif`. Finally, at instant 5, the symbol `msg` is read again and transition  $t_3$  is triggered to enter `alarm`. A tick on `trigger_t3` is possible as it precedes `trigger_t3_max`. Likewise, `trigger_t3_max` defines a maximum limit to ensure any  $t_3$ -transition triggering only before.

## 4.2 Previous operator (and PID controllers)

Another useful operator is `pre` with similar syntax and semantics as in Lustre [18]. This operator simply allows to refer to the previous timestamp on a clock. Hence, a substantial part of feedback systems can be modeled accurately as they require registers to store previous values. The power of computation is significantly augmented and allows us to model more complex systems, such as mathematical sequences and series (*e.g.*, Fibonacci), differential calculus (derivatives, Euler’s integrator), or digital filters.

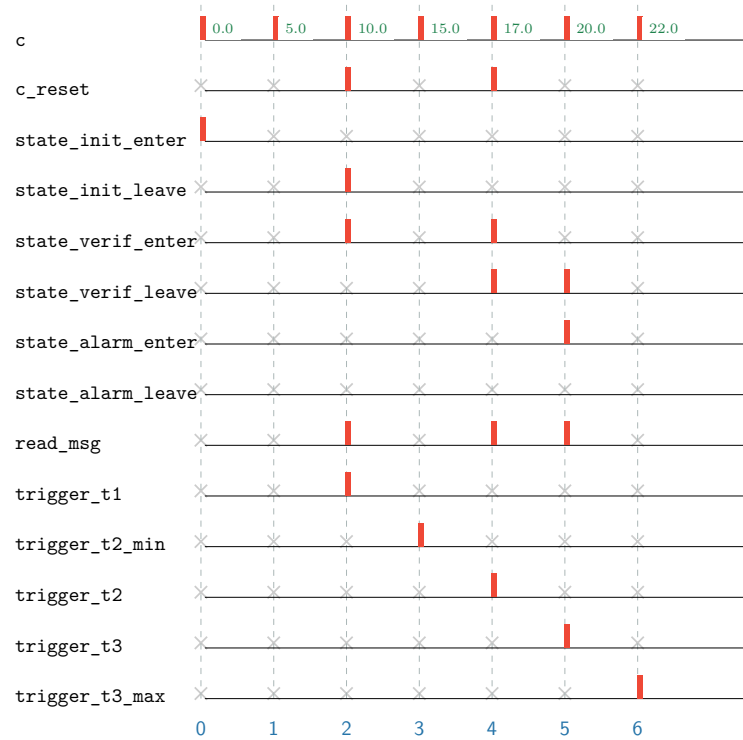
Since this operator refers to the value of a signal at a previous instant, we generalized TESL clocks as *flows*. A flow is a clock where timestamps are no longer required to be monotonic. As a matter of fact, these “timestamps” are simply called *values*.

We extend the syntax of TESL as shown in Subsection 2.3, with:

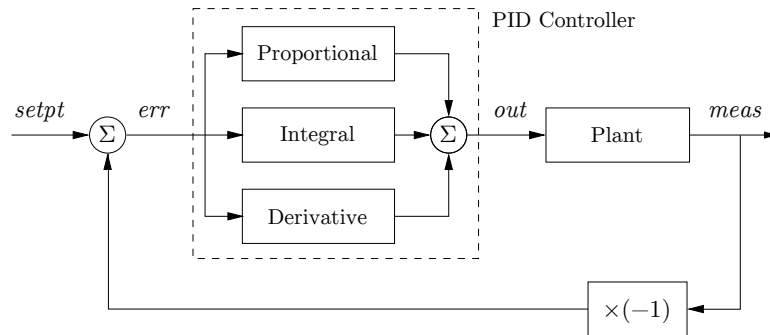
$$\langle clock \rangle ::= K \in \mathbb{K} \quad | \quad \text{pre } \langle clock \rangle$$

This extension is useful at modeling feedback systems. Let us illustrate this with the ubiquitous algorithm of automatic control theory: the Proportional-integral-derivative (PID) controller [39]. In this theory, a PID controller delivers a control signal to a process in order to bring a process output closer to a reference setpoint (*e.g.*, cruise control in cars, autopilots in airplanes).

15:10 TESL: A Model with Metric Time for Modeling and Simulation



■ **Figure 7** A satisfying run prefix to encode a timed automaton.



■ **Figure 8** General diagram of a process using a PID controller.

The block diagram in Figure 8 shows the structure of the controller. Basically, the system receives as input the error signal **err**, *i.e.* the difference between the reference setpoint **setpt** and the process output **out**, and computes a control signal based on the sum of a term proportional to the error, an integral term and a derivative term. Each of the three terms is parameterized by a multiplying factor, respectively  $K_p$ ,  $K_i$  and  $K_d$ , which are commonly called *gains*. Thereafter, the controller output enters a *transfer function* which translates the control signal **out** into the process output **meas**. For instance in automotive control theory, this occurs when converting the position of the gas pedal into the generated car velocity. This new output will be used to feed the error back at the next computing cycle. It is possible to describe this system straightforwardly in TESL as in Listing 2.

■ **Listing 2** The PID controller

```

// Time
time relation dt = 1.0
time relation t = [0.0] -> (pre t) + dt
// Gain
time relation Kp = 0.1
time relation Ki = 0.2
time relation Kd = 0.2
// Setpoint
time relation setpt = 40.0
// Control signal
time relation err      = setpt - meas
time relation integr  = [0.0] -> (pre integr) + (err * dt)
time relation derivat = [0.0] -> (err - (pre err)) / dt
time relation out     = (Kp * err) + (Ki * integr) + (Kd * derivat)
// Simple actuation
time relation meas    = [0.0] -> (pre meas) + (pre out)

```

When running this example, the solver yields the output shown by the extract in Listing 3.

■ **Listing 3** An extract of the satisfying run found by Heron of the PID controller

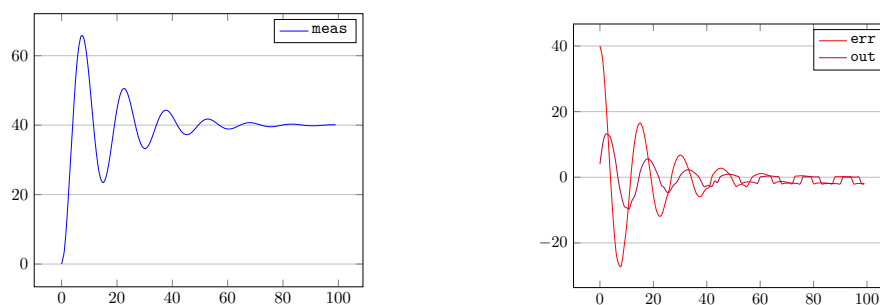
```

### Solver has successfully returned 1 model
## Simulation result [Ox1ADAB]:

```

	meas	err	integr	derivat	out
[1]	0.0	40.0	0.0	0.0	4.0
[2]	4.0	36.0	36.0	-4.0	10.0
[3]	14.0	26.0	62.0	-10.0	13.0
[4]	27.0	13.0	75.0	-13.0	13.0
[5]	40.0	-1.0	74.0	-14.0	12.0
...					

Additionally, the values of the flows `meas`, `err` and `out` are plotted in Figure 9. As expected, we observe that the process output `meas` is brought closer to the reference setpoint `setpt = 40.0`. Besides, the error signal and the control signal `out` gradually decrease to 0.0 as the need to damp out oscillations progressively decreases.



■ **Figure 9** Plotting values for `meas`, `err` and `out`.

## 5 Related Work

In the family of synchronous programming languages [3], Lustre [18], Esterel [6, 5] and Signal [17] are known to provide polymorphic time (time domains of various type). However, their time model is purely logical, which is not suited when dealing with modeling non-discretizable systems. Prelude [32] and Zélus [9] overcome this with continuous dynamics.

All these previous models derive clocks from a global root clock, which constrains models to flow from a single reaction clock. Polychrony (clocks possibly living in various independent timeframes) overcomes this restriction by allowing specifications with more relaxed and concurrent execution of systems. This feature can be observed in the Signal language or polychronous automata [23]. Compared to TESL, they do not allow metric time constraints.

TESL is also inspired by CCSL which supports asynchronous constraints on events. It admits an executable [38] and denotational semantics [11, 28]. However, time in CCSL is purely logical and durations are counted as a number of ticks on a clock.

On a more theoretical-side, timed automata [2, 1] support both discrete events and metric time. However, clocks are global and uniform, they necessarily progress at the same rate.

All in all, TESL attempts to overcome these limitations and provides a general-purpose specification language of synchronous and asynchronous constraints with clocks over poly-morphic time while supporting polychrony, and mixing logical and metric time.

## **6 Future work**

The outcome of our study leads us to various future opportunities:

- An effort is currently running towards a machine-checkable formalization of the operational and denotational semantics into the Isabelle/HOL proof assistant [31, 30]. We successfully proved that the operational semantics was correct and complete with respect to the denotational semantics. Proving both extensions of the paper is a future direction.
- Numerous questions about model-checking remain unanswered. In our experiments, we have observed that unfolded specifications could be refolded with abstract interpretation techniques. This would offer a finite-representation of these infinite-state systems, thereby providing means to decide safety and liveness properties of such systems.
- In addition, the TESL language seems to be suited for modeling and simulation of systems with time of various kind. With the new extensions we propose and their implementation in an existing efficient solver, we believe TESL can become a relevant asset as a simulation engine for simulation platforms, such as the GEMOC Studio [10].

## **7 Conclusion**

This study introduces a language – named TESL – suited for the modeling and simulation of complex systems with multi-level time considerations. For this purpose, we illustrated how the language is suited for various applications of time in models. We first illustrated the main properties of the language (absence of a global root clock, stutter invariance). Then, we introduced two extensions of the language along with two applications depicted by (1) an encoding of timed automata, and (2) an implementation of a PID controller.

Most of the widely used formalisms suffer from restrictions in their model of time, which we attempt to address. Some consider time as purely logical and may not be suited for real-time systems as computing cycles may not necessarily flow at a fixed rate. Some other consider time as global which is restrictive towards distributed systems where time does not flow at the same rate in the different components, and may not be synchronized. We believe our approach is complementary to state-of-the-art environments and may help to circumvent their drawbacks by considering time in its whole nature.



---

References

---

- 1 Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 8–22, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
- 2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.
- 3 A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.
- 4 Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebelen, and Pierre McKenzie. *Systems and Software Verification*. Springer Berlin Heidelberg, 2001. doi:10.1007/978-3-662-04558-9.
- 5 G. Berry. The constructive semantics of pure Esterel, 1996.
- 6 Gérard Berry. The foundations of Esterel. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*, pages 425–454. MIT Press, Cambridge, MA, USA, 2000.
- 7 Gérard Berry. SCADE: Synchronous design and validation of embedded control software. In S. Ramesh and Prahladavaradan Sampath, editors, *Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, pages 19–33, Dordrecht, 2007. Springer Netherlands.
- 8 Frédéric Boulanger, Christophe Jacquet, Cécile Hardebolle, and Iuliana Prodan. TESL: a language for reconciling heterogeneous execution traces. In *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2014)*, pages 114–123, Lausanne, Switzerland, October 2014. doi:10.1109/MEMCOD.2014.6961849.
- 9 Timothy Bourke and Marc Pouzet. Zélus: A synchronous language with odes. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, HSCC '13*, page 113–118, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2461328.2461348.
- 10 Benoit Combemale, Betty H.C. Cheng, Robert B. France, Jean-Marc Jezequel, and Bernhard Rumpe. *Globalizing Domain-Specific Languages*, volume 9400 of *LNCS, Programming and Software Engineering*. Springer International Publishing, 2015.
- 11 Julien Deantoni, Charles André, and Régis Gascon. CCSL denotational semantics. Research Report RR-8628, Inria, November 2014. URL: <https://hal.inria.fr/hal-01082274>.
- 12 Julien DeAntoni and Frédéric Mallet. Timesquare: Treat your models with logical time. In Carlo A. Furia and Sebastian Nanz, editors, *Objects, Models, Components, Patterns - 50th International Conference, TOOLS 2012, Prague, Czech Republic, May 29-31, 2012. Proceedings*, volume 7304 of *Lecture Notes in Computer Science*, pages 34–41. Springer, 2012. doi:10.1007/978-3-642-30561-0\_4.
- 13 J. Eker, J. W. Janneck, E. A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.
- 14 K. Erciyès. *Distributed Real-Time Systems*. Springer International Publishing, 2019. doi:10.1007/978-3-030-22570-4.
- 15 Sulaymon Eshkabilov. *MATLAB®/Simulink® Essentials: MATLAB®/Simulink® for Engineering Problem Solving and Numerical Analysis*. Lulu Publishing Services, November 2016.
- 16 Kelly Garcés, Julien Deantoni, and Frédéric Mallet. A Model-Based Approach for Reconciliation of Polychronous Execution Traces. In *SEAA 2011 - 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, Oulu, Finland, August 2011. IEEE. URL: <https://hal.inria.fr/inria-00597981>.
- 17 P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier. Synchronous data flow programming with the language SIGNAL. *IFAC Proceedings Volumes*, 20(2):359–364, 1987. 2nd IFAC Workshop on Adaptive Systems in Control and Signal Processing 1986, Lund, Sweden, 30 June-2 July 1986.

- 18 N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.
- 19 Michael J W Hall. Concepts in special relativity. In *General Relativity: An Introduction to Black Holes, Gravitational Waves, and Cosmology*, 2053-2571, pages 1–1 to 1–11. Morgan & Claypool Publishers, 2018. doi:10.1088/978-1-6817-4885-6ch1.
- 20 Cécile Hardebolle and Frédéric Boulanger. Exploring multi-paradigm modeling techniques. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 85(11/12):688–708, November/December 2009. doi:10.1177/0037549709105240.
- 21 Yannick Hildenbrand. Ed-247 (vistas) gateway for hybrid test systems. In *Aerospace Systems and Technology Conference*. SAE International, October 2018. doi:10.4271/2018-01-1949.
- 22 Leslie Lamport. What good is temporal logic? *Information Processing 83*, R. E. A. Mason, ed., Elsevier Publishers, 83:657–668, May 1983. URL: <https://www.microsoft.com/en-us/research/publication/good-temporal-logic/>.
- 23 Paul Le Guernic, Thierry Gautier, Jean-Pierre Talpin, and Loïc Besnard. Polychronous automata. In *TASE 2015, 9th International Symposium on Theoretical Aspects of Software Engineering*, pages 95–102, Nanjing, China, September 2015. IEEE Computer Society.
- 24 E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.
- 25 Edward A. Lee and Alberto Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Trans. CAD*, 17(12), 1998.
- 26 Frédéric Mallet, Julien Deantoni, Charles André, and Robert De Simone. The Clock Constraint Specification Language for building timed causality models. *Innovations in Systems and Software Engineering*, 6(1-2):99–106, March 2010.
- 27 Frédéric Mallet and Robert de Simone. Correctness issues on MARTE/CCSL constraints. *Science of Computer Programming*, 106:78–92, 2015. Special Issue: Architecture-Driven Semantic Analysis of Embedded Systems. doi:10.1016/j.scico.2015.03.001.
- 28 Mathieu Montin and Marc Pantel. Mechanizing the denotational semantics of the clock constraint specification language. In El Hassan Abdelwahed, Ladjel Bellatreche, Mattéo Golfarelli, Dominique Méry, and Carlos Ordóñez, editors, *Model and Data Engineering*, pages 385–400, Cham, 2018. Springer International Publishing.
- 29 Hai Nguyen Van, Thibaut Balabonski, Frédéric Boulanger, Chantal Keller, Benoît Valiron, and Burkhardt Wolff. A symbolic operational semantics for TESL. In Alessandro Abate and Gilles Geeraerts, editors, *Formal Modeling and Analysis of Timed Systems*, pages 318–334, Cham, 2017. Springer International Publishing.
- 30 Tobias Nipkow and Gerwin Klein. *Concrete Semantics: With Isabelle/HOL*. Springer Publishing Company, Incorporated, 2014.
- 31 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- 32 Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, 2011. URL: <https://hal.inria.fr/inria-00638936>.
- 33 Kirill Rozhdestvensky, Vladimir Ryzhov, Tatiana Fedorova, Kirill Safronov, Nikita Tryaskin, Shaharin Anwar Sulaiman, Mark Ovinis, and Suhaimi Hassan. *Description of the Wolfram SystemModeler*, pages 23–87. Springer Singapore, Singapore, 2020. doi:10.1007/978-981-15-2803-3\_2.
- 34 Werner Schutz. *The Testability of Distributed Real-Time Systems*. Kluwer Academic Publishers, USA, 1993.
- 35 J. A. Stankovic. Misconceptions about real-time computing: a serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.
- 36 Stephen Weeks. Whole-program compilation in mlton. In *Proceedings of the 2006 Workshop on ML, ML '06*, page 1, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1159876.1159877.

- 37 Sam Westrick, Rohan Yadav, Matthew Fluet, and Umut A. Acar. Disentanglement in nested-parallel programs. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi: 10.1145/3371115.
- 38 M. Zhang and F. Mallet. An executable semantics of Clock Constraint Specification Language and its applications. In *Formal Techniques for Safety-Critical Systems: 4th International Workshop, FTSCS 2015*, pages 37–51, Cham, 2016. Springer.
- 39 Karl Johan Åström and Richard M. Murray. *Feedback Systems*. Princeton University Press, Princeton, 2010. URL: <https://www.degruyter.com/view/title/563028>.



# Complexity of Qualitative Timeline-Based Planning

Dario Della Monica 

University of Udine, Italy

<https://users.dimi.uniud.it/~dario.dellamonica/>

dario.dellamonica@uniud.it

Nicola Gigante 

University of Udine, Italy

<https://users.dimi.uniud.it/~nicola.gigante/>

nicola.gigante@uniud.it

Salvatore La Torre 

University of Salerno, Italy

<https://docenti.unisa.it/salvatore.latorre>

slatorre@unisa.it

Angelo Montanari 

University of Udine, Italy

<https://users.dimi.uniud.it/~angelo.montanari>

angelo.montanari@uniud.it

---

## Abstract

The timeline-based approach to automated planning was originally developed in the context of space missions. In this approach, problem domains are expressed as systems consisting of independent but interacting components whose behaviors over time, the *timelines*, are governed by a set of temporal constraints, called *synchronization rules*. Although timeline-based system descriptions have been successfully used in practice for decades, the research on the theoretical aspects only started recently. In the last few years, some interesting results have been shown concerning both its expressive power and the computational complexity of the related planning problem. In particular, the general problem has been proved to be EXPSPACE-complete. Given the applicability of the approach in many practical scenarios, it is thus natural to ask whether computationally simpler but still expressive fragments can be identified. In this paper, we study the timeline-based planning problem with the restriction that only *qualitative* synchronization rules, i.e., rules without explicit time bounds in the constraints, are allowed. We show that the problem becomes PSPACE-complete.

**2012 ACM Subject Classification** Computing methodologies → Temporal reasoning

**Keywords and phrases** Timeline-based planning, qualitative temporal constraints, complexity

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.16

**Funding** The authors acknowledge the partial support by the Italian INdAM-GNCS project *Ragionamento Strategico e Sintesi Automatica di Sistemi Multi-Agente*. Nicola Gigante and Angelo Montanari are partially supported by the PRID project *ENCASE - Efforts in the uNderstanding of Complex interActing SystEms*.

## 1 Introduction

Timeline-based planning is an approach to automated planning and scheduling that arose in connection to space operations [16]. In this setting, planning domains are modeled as systems made of independent but interacting components, whose behavior over time, the *timelines*, is governed by a set of temporal constraints. Compared to other well-established *action-based* planning formalisms such as STRIPS [7] and PDDL [15], timelines conform to the *declarative* paradigm, and are very effective in modeling the behavior of complex systems where multiple components have to interact to obtain a common goal rather than scenarios where the target of the planning process is a single agent. Moreover, being born in a context



© Dario Della Monica, Nicola Gigante, Salvatore La Torre, and Angelo Montanari; licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 16; pp. 16:1–16:13

Leibniz International Proceedings in Informatics



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

where scheduling of operations is often as important as planning, timeline-based languages are particularly tailored to *temporal reasoning*, and to the modeling of real-time systems. These features have proven to be quite useful in practice, as timeline-based planning systems have been successfully deployed both at NASA [2, 3] and ESA [8, 9] for short- to long-term mission planning over the last three decades [4, 6].

From a theoretical perspective, timeline-based modeling languages are interesting for their rich syntax and powerful semantics. However, the study of their expressiveness and computational complexity has started only recently. The expressive power of the language has been compared with action-based counterparts [11] and the general plan-existence problem for timeline-based planning has been proved to be EXPSpace-complete [5, 12]. Timeline-based *games* [13], where some of the components are under control of the environment and the controller has to ensure the satisfaction of the constraints independently from the environment's choices, have also been studied to formalize and extend the practice of *flexible timelines* used in current timeline-based planning systems. Deciding whether there is a winning strategy for a timeline-based game has been proved to be 2EXPTIME-complete.

Despite the high computational complexity of the related decision problems, timeline-based models have been routinely and successfully used in complex real-world scenarios for decades. This apparent paradox indeed suggests the existence of interesting fragments of the general theory with more tractable complexities that are still suitable for meaningful real-world applications. Starting from this observation, this paper identifies the *qualitative* fragment of the timeline-based planning problems, *i.e.*, problems where the system behavior is described without referring to explicit time bounds. This restriction is nevertheless suitable for many practical scenarios as also testified by the plethora of approaches based on qualitative time models in many application domains including automated planning [10].

We prove that the plan-existence problem for this fragment is PSPACE-complete as opposed to the EXPSpace-completeness of the general problem. The proof is given by means of an automata-theoretic construction that builds on the technique developed by Della Monica *et al.* [5] to prove the complexity in the general case. We also discuss some interesting consequences that this result brings to the table.

The paper is structured as follows. Section 2 recalls the basic syntax and semantics of timeline-based planning problems, and introduces the *qualitative* fragment studied here. Then, Section 3 proves that the problem can be solved in polynomial space, while Section 4 proves that it is also PSPACE-hard. Section 5 discusses the consequences of these results together with some interesting future developments.

## 2 Qualitative timeline-based planning

In this section, we introduce the notion of qualitative timeline-based planning. To this end, we recall the main definitions about timeline-based planning. We start with the definition of state variable, which is the basic building block of the framework.

- **Definition 1** (State variable). *A state variable is a tuple  $x = (V_x, T_x, D_x)$ , where:*
- $V_x$  is the finite domain of the variable;
  - $T_x : V_x \rightarrow 2^{V_x}$  is the value transition function, which maps each value  $v \in V_x$  to the set of values that can (immediately) follow it;
  - $D_x : V_x \rightarrow \mathbb{N}^+ \times (\mathbb{N}^+ \cup \{+\infty\})$  is a function that maps each  $v \in V_x$  to the pair  $(d_{min}^{x=v}, d_{max}^{x=v})$  of minimum and maximum durations allowed for intervals where  $x = v$ .

The behavior of a state variable  $x$  over time is modeled by a *timeline*, *i.e.*, a finite sequence of *tokens* each one denoting a value  $v$  and a time interval  $d$  meaning that  $x$  evaluates to  $v$  within  $d$ . Formally:

► **Definition 2** (Timelines). A token for  $x$  is a tuple  $\tau = (x, v, d)$ , where  $x$  is a state variable,  $v \in V_x$  is the value held by the variable, and  $d \in \mathbb{N}^+$  is the duration of the token. A timeline for a state variable  $x = (V_x, T_x, D_x)$  is a finite sequence  $\mathbb{T} = \langle \tau_1, \dots, \tau_k \rangle$  of tokens for  $x$ .

For any token  $\tau_i = (x, v_i, d_i)$  in a timeline  $\mathbb{T} = \langle \tau_1, \dots, \tau_k \rangle$  we can define the functions  $\text{start-time}(\mathbb{T}, i) = \sum_{j=1}^{i-1} d_j$  and  $\text{end-time}(\mathbb{T}, i) = \text{start-time}(\mathbb{T}, i) + d_i$ , hence mapping each token to the corresponding time interval  $[\text{start-time}, \text{end-time})$  (right endpoint excluded). The *horizon* of a timeline  $\mathbb{T}$  is defined as the end time  $\text{end-time}(\mathbb{T}, k)$  of its last token  $\tau_k$ . When there is no ambiguity, we write  $\text{start-time}(\tau_i)$  and  $\text{end-time}(\tau_i)$  to denote, respectively,  $\text{start-time}(\mathbb{T}, i)$  and  $\text{end-time}(\mathbb{T}, i)$ .

The problem domain and the goal are modeled by a set of temporal constraints, called *synchronization rules*. A synchronization rule is of the form:

$$\begin{aligned} \text{rule} &:= a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k, \text{ with} \\ \mathcal{E}_i &:= \exists a_1[x_1 = v_1] a_2[x_2 = v_2] \dots a_n[x_n = v_n] \cdot \mathcal{C}_i \end{aligned}$$

where  $x_0, \dots, x_n$  are state variables,  $v_0, \dots, v_n$  are values, with  $v_i \in V_{x_i}$  for all  $i$ ,  $a_0, \dots, a_n \in \mathcal{N}$  are symbols from a set of token names, and  $\mathcal{C}$  is a conjunction of atomic clauses as described below. The semantics of synchronization rules is only informally recalled because of space concerns (see [13, Definitions 7 and 8] for a formal definition). Each rule consists of a *trigger* ( $a[x_0 = v_0]$ ) and a disjunction of *existential statements*. It is satisfied if for each token satisfying the trigger, at least one of the disjuncts is satisfied. The trigger can also be empty ( $\top$ ), in which case the rule is said to be *triggerless* and asks for the satisfaction of the body without any precondition. Each existential statement requires the existence of some tokens such that the clause  $\mathcal{C}$  is satisfied. The clause is in turn a conjunction of *atoms*, that is, atomic relations between endpoints of the quantified tokens, of the form:

$$\text{term} := \text{start}(a) \mid \text{end}(a) \mid t \qquad \text{atom} := \text{term} \leq_{[l, u]} \text{term}$$

where  $a \in \mathcal{N}$ ,  $l \in \mathbb{N}$ ,  $t \in \mathbb{N}$ , and  $u \in \mathbb{N} \cup \{+\infty\}$ . As an example, the atom  $\text{start}(a) \leq_{[l, u]} \text{end}(b)$  relates the two mentioned endpoints of the tokens  $a$  and  $b$  by stating that the distance between them must be at least  $l$  and at most  $u$ . When  $u = +\infty$ , there is no upper bound on the distance between endpoints. In this case, the atom is said to be *unbounded*, and *bounded* otherwise. An atom  $\text{term} \leq_{[l, u]} \text{term}$  with  $l = 0$  and  $u = +\infty$  is said to be *qualitative*, and the subscript is usually omitted in this case. An endpoint of a token can also be related with an absolute time point (e.g.  $\text{start}(a) \leq_{[0, 3]} 4$ ). Such an atom is called a *time-point atom*.

A timeline-based planning problem consists of a set of state variables and a set of rules that represent the problem domain and the goal.

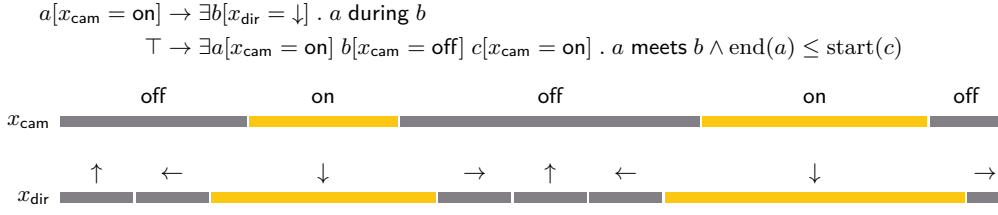
► **Definition 3** (Timeline-based planning problem). An instance of a timeline-based planning problem, commonly referred to as a timeline-based planning problem, is a pair  $P = (\text{SV}, S)$ , where  $\text{SV}$  is a set of state variables and  $S$  is a set of synchronization rules over  $\text{SV}$ .

A solution plan for a given timeline-based planning problem is a set of timelines, one for each state variable.

► **Definition 4** (Solution plan). A solution plan for a problem  $P = (\text{SV}, S)$  is a set of timelines  $\Gamma = \{\mathbb{T}_x \mid x \in \text{SV}\}$ , one for each  $x \in \text{SV}$ , all with the same horizon, such that  $v_{i+1} \in T_x(v_i)$  and  $d_{\min}^{x=v_i} \leq d_i \leq d_{\max}^{x=v_i}$  for all tokens  $\tau_i = (x, v_i, d_i) \in \mathbb{T}_x$ , and all the rules in  $S$  are satisfied.

We know from [12] that the problem of deciding whether there exists a solution plan for a given timeline-based planning problem is EXPSpace-complete. In this paper, we focus on the *qualitative* version of timeline-based planning problems.





■ **Figure 1** An example of timeline-based planning problem. Two state variables are used to represent the on/off state of a camera ( $x_{\text{cam}}$ ) and its pointing direction ( $x_{\text{dir}}$ ). The transition function of  $x_{\text{dir}}$  forces the camera to only move counterclockwise.

► **Definition 5** (Qualitative timeline-based planning problem). *A timeline-based planning problem  $P = (\text{SV}, S)$  is qualitative if  $D_x(v) = (1, +\infty)$  for each  $x \in \text{SV}$  and  $v \in V_x$ , and all the synchronization rules in  $S$  make only use of qualitative atoms, and no time-point atoms.*

Qualitative synchronization rules do not allow one to express *real-time* constraints, but they are still a quite expressive formalism. Equalities between endpoints and between whole tokens are definable, *i.e.*,  $\text{start}(a) = \text{start}(b)$  can be written as  $\text{start}(a) \leq \text{start}(b) \wedge \text{start}(b) \leq \text{start}(a)$  and  $a = b$  as  $\text{start}(a) = \text{start}(b) \wedge \text{end}(a) = \text{end}(b)$ . Moreover, one can express non-strict versions of all Allen’s interval relations [1]: one can define *a meets b* as  $\text{end}(a) = \text{start}(b)$ , *a during b* as  $\text{start}(a) \leq \text{start}(b) \wedge \text{end}(b) \leq \text{end}(a)$ , and so on.

Figure 1 shows a possible solution for a problem with two state variables,  $x_{\text{cam}}$  and  $x_{\text{dir}}$ , with  $V_{x_{\text{cam}}} = \{\text{on}, \text{off}\}$  and  $V_{x_{\text{dir}}} = \{\uparrow, \leftarrow, \downarrow, \rightarrow\}$ , that respectively represent the on/off state of a camera and its pointing direction. The transition function  $T_{x_{\text{dir}}}$  of  $x_{\text{dir}}$  is such that the camera can only stay still or move counterclockwise, *e.g.*  $T_{x_{\text{dir}}}(\leftarrow) = \{\leftarrow, \downarrow\}$ . The rules are built on the set  $\mathcal{N} = \{a, b, \dots\}$  of token names. The first rule requires the camera to point down every time it is switched on (*e.g.*, to point towards ground from an airplane). The objective of the system is to perform two shoots, provided that the camera is switched off between them in order to cool down. This goal is encoded by the second *triggerless* synchronization rule.

### 3 Complexity of qualitative timeline-based planning

In this section, we give a *polynomial-space* algorithm to decide whether there exists a solution plan for a given qualitative timeline-based planning problem.

Given a qualitative timeline-based planning problem  $P$ , we show how to build a *non-deterministic finite automaton* (NFA)  $\mathcal{A}$  whose accepted words correspond to the solution plans of  $P$ . The approach, inspired by the one adopted by Della Monica *et al.* [5] for the general case, has been not only tailored to the qualitative setting but also refined to obtain the desired complexity result.

#### 3.1 Plans as words

Let  $P = (\text{SV}, S)$  be a qualitative timeline-based planning problem and let  $V = \bigcup_{x \in \text{SV}} V_x$ . Without loss of generality, we consider only qualitative timeline-based planning problems whose state variables have *trivial* transition functions, *i.e.*, for each  $x \in \text{SV}$  and  $v \in V_x$ , either  $T_x(v) = V_x$  or  $T_x(v) = \emptyset$ . The first step is to encode timelines and plans as words that can be fed to an automaton. Let the alphabet associated with  $P$  be  $\Sigma_P = \{\sigma : \text{SV} \rightarrow V \cup \{\circ\} \mid \sigma(x) \in V_x \cup \{\circ\}\}$ , *i.e.*, symbols map each state variable  $x$  to a value within its domain  $V_x$



or to a special symbol  $\circ$ . By fixing an ordering among the variables, we will also denote such maps as tuples in the standard way. Observe that the size of the alphabet is at most exponential in the size of  $P$ , precisely  $|\Sigma_P| \leq (|V| + 1)^{|\text{SV}|}$ .

Let the set of *initial symbols* be defined as  $\Sigma_P^I = \{\sigma \in \Sigma_P \mid \forall x \in \text{SV} . \sigma(x) \neq \circ\}$ . Each plan can be encoded with a word in  $\Sigma_P^I \Sigma_P^*$ , and *vice versa*, where each occurrence of  $v \in V_x$  denotes a time point where  $x$  is updated to  $v$  and each occurrence of  $\circ$  a time point where the value of  $x$  stays unchanged. Formally, let  $\bar{\sigma} = \langle \sigma_0, \dots, \sigma_{|\bar{\sigma}|-1} \rangle$  be a word in  $\Sigma_P^I \Sigma_P^*$ . Given a state variable  $x \in \text{SV}$ , let  $\{i_0, i_1, \dots, i_{k-1}\} = \{i \mid \sigma_i(x) \neq \circ\}$ , with  $i_{j-1} < i_j$  for all  $j \in \{1, \dots, k-1\}$ , *i.e.*, the ordered set of positions where the value of  $x$  changes. Then, the word  $\bar{\sigma}$  induces a *timeline*  $\mathbb{T}_x$  defined as  $\mathbb{T}_x = \langle (x, v_{i_0}, i_1 - i_0), (x, v_{i_1}, i_2 - i_1), \dots, (x, v_{i_{k-1}}, i_k - i_{k-1}) \rangle$ , where  $v_i = \sigma_i(x)$  and  $i_k = |\bar{\sigma}|$ , as the timeline for  $x$  induced by  $\bar{\sigma}$ , while the plan induced by  $\bar{\sigma}$  is the set of all timelines, one for each  $x \in \text{SV}$ , induced by  $\bar{\sigma}$ . A converse correspondence of plans to words can be defined accordingly.

### 3.2 Blueprints and viewpoints

A key concept in our construction is the *blueprint*: a description of a possible way to satisfy a synchronization rule. Here, we use a significantly revised notion of blueprint with the respect to the one introduced in [5]. This new notion allows us to gain in succinctness and plays a crucial role in achieving the wished complexity bound.

We begin with some notation and terminology. Let  $\mathcal{P}$  be a *preorder* over its domain  $\text{dom}(\mathcal{P})$ . For every  $x \in \text{dom}(\mathcal{P})$ , we define  $\text{pred}(\mathcal{P}, x)$  as the set of immediate predecessors of  $x$  in  $\mathcal{P}$  and  $\text{succ}(\mathcal{P}, x)$  as the set of immediate successors of  $x$  in  $\mathcal{P}$ . We define  $\text{maximals}(\mathcal{P})$  as the set of elements of  $\text{dom}(\mathcal{P})$  with no successors in  $\mathcal{P}$  and  $\text{minimals}(\mathcal{P})$  as the set of elements of  $\text{dom}(\mathcal{P})$  with no predecessors in  $\mathcal{P}$ . Finally, given  $K \subseteq \text{dom}(\mathcal{P})$ , we say that  $K$  is  $\mathcal{P}$ -complete if for every  $x \in \text{dom}(\mathcal{P})$  there is  $y \in K$  such that  $x \preceq_{\mathcal{P}} y$  or  $y \preceq_{\mathcal{P}} x$  and that  $K$  is minimally  $\mathcal{P}$ -complete if it is  $\mathcal{P}$ -complete and its elements are pairwise incomparable in  $\mathcal{P}$ .

We assume, *w.l.o.g.*, that at least one rule with  $a[x = v]$  as trigger belongs to  $S$  for each  $x \in \text{SV}$  and  $v \in V_x$ . Now, fix a synchronization rule  $\mathcal{R} \equiv a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \dots \vee \mathcal{E}_m$ , and one of its existential statements  $\mathcal{E} \equiv \exists a_1[x_1 = v_1] \dots a_n[x_n = v_n] . \mathcal{C}$ , *i.e.*,  $\mathcal{E} = \mathcal{E}_j$  for some  $j \in \{1, \dots, m\}$ . We assume, *w.l.o.g.*, that the atom  $\text{start}(a_i) \leq \text{end}(a_i)$  occurs in  $\mathcal{C}$  for all  $i \in \{0, \dots, n\}$ ; we also assume that, for every  $i, j$  with  $x_i = x_j$  and  $i \neq j$ , if one among  $\text{start}(a_i) \leq \text{start}(a_j)$ ,  $\text{end}(a_i) \leq \text{end}(a_j)$ , and  $\text{start}(a_i) \leq \text{end}(a_j)$  occurs in  $\mathcal{C}$ , then  $\text{end}(a_i) \leq \text{start}(a_j)$  occurs in  $\mathcal{C}$  as well. Recall that  $v_i \in V_{x_i}$  for all  $i \in \{0, \dots, n\}$ . Then,  $\mathcal{E}$  defines a preorder  $\mathcal{P}_{\mathcal{E}}$  over the domain  $\text{dom}(\mathcal{P}_{\mathcal{E}}) = \{\text{start}(a_i), \text{end}(a_i) \mid i \in \{0, \dots, n\}\}$ , *i.e.*, over the set of endpoints of all the tokens quantified by  $\mathcal{E}$ . Intuitively, a *blueprint* for  $\mathcal{E}$ , denoted by  $B_{\mathcal{E}}$ , is an extension of the preorder defined by  $\mathcal{E}$  where an additional element is inserted between any pair of comparable elements, and before and after minimal and maximal elements, respectively. Such additional elements, denoted  $\text{pumps}(B_{\mathcal{E}})$ , are called the *pumping points* of  $B_{\mathcal{E}}$ . Formally, blueprints are defined as follows.

► **Definition 6** (Blueprint). *Let  $\mathcal{E}$  be an existential statement. A blueprint for  $\mathcal{E}$  is a preorder  $B_{\mathcal{E}}$  defined as:*

1.  $\text{dom}(B_{\mathcal{E}}) = \text{dom}(\mathcal{P}_{\mathcal{E}}) \cup \text{pumps}(B_{\mathcal{E}})$  where

$$\begin{aligned} \text{pumps}(B_{\mathcal{E}}) = & \{ \{y|x\} \mid x \in \text{dom}(\mathcal{P}_{\mathcal{E}}) \setminus \text{minimals}(\mathcal{P}_{\mathcal{E}}), y \in \text{pred}(\mathcal{P}_{\mathcal{E}}, x) \} \\ & \cup \{ \{-|x\} \mid x \in \text{minimals}(\mathcal{P}_{\mathcal{E}}) \} \cup \{ \{x|- \} \mid x \in \text{maximals}(\mathcal{P}_{\mathcal{E}}) \}; \end{aligned}$$

2. for all  $x, y \in \text{dom}(\mathcal{P}_{\mathcal{E}})$ ,  $x \preceq_{B_{\mathcal{E}}} y$  if and only if  $x \preceq_{\mathcal{P}_{\mathcal{E}}} y$ , *i.e.*, the ordering relation is unchanged over elements of  $\text{dom}(\mathcal{P}_{\mathcal{E}})$ ;

3. for every  $x \in \text{dom}(\mathcal{P}_\mathcal{E}) \setminus \text{minimals}(\mathcal{P}_\mathcal{E})$  and  $y \in \text{pred}(\mathcal{P}_\mathcal{E}, x)$ , we have  $y \preceq_{B_\mathcal{E}} \langle y|x \rangle$  and  $\langle y|x \rangle \preceq_{B_\mathcal{E}} x$ ;
4. for every  $x \in \text{minimals}(\mathcal{P}_\mathcal{E})$ , we have  $\langle \neg|x \rangle \preceq_{B_\mathcal{E}} x$ ; and
5. for every  $x \in \text{maximals}(\mathcal{P}_\mathcal{E})$ , we have  $x \preceq_{B_\mathcal{E}} \langle x| \neg \rangle$ .

Blueprints statically describe the syntactic structure of the rules. A *viewpoint* pairs a blueprint with a set of pointers to its pumping points. It is the dynamic structure that keeps track of how the rules are matching on the specific word being read.

► **Definition 7 (Viewpoint).** A viewpoint is a pair  $\mathbb{V} = \langle B_\mathcal{E}, K \rangle$ , where  $B_\mathcal{E}$  is a blueprint for an existential statement  $\mathcal{E}$  and  $K \subseteq \text{pumps}(B_\mathcal{E})$  is a subset of its pumping points that is minimally  $B_\mathcal{E}$ -complete.

Given  $\mathbb{V} = \langle B_\mathcal{E}, K \rangle$ ,  $K$  is the *frontier* of  $\mathbb{V}$ , and its elements are its *frontier points*. Moreover,  $\mathbb{V}$  is said to be *minimal* or *maximal* if  $K = \text{minimals}(B_\mathcal{E})$  or  $K = \text{maximals}(B_\mathcal{E})$ , respectively.

A viewpoint keeps track of how a blueprint is being matched over a plan encoding; in particular, its frontier separates the already matched part from the rest of the blueprint that still needs to be matched. When a symbol is read, each frontier point can either *pump* (i.e., stay unchanged) or *step* (i.e., advance to point to other pumping points). Formally, a viewpoint  $\mathbb{V} = \langle B_\mathcal{E}, K \rangle$  can *evolve* into another viewpoint  $\mathbb{V}' = \langle B_\mathcal{E}, K' \rangle$ , written  $\mathbb{V} \rightarrow \mathbb{V}'$ , if and only if for all  $k \in K$  either  $k \in K'$  (i.e.,  $k$  pumps) or  $k' \notin K'$  for all  $k'$  such that  $k' \preceq_{B_\mathcal{E}} k$  (i.e.,  $k$  steps). If  $\mathbb{V} = \langle B_\mathcal{E}, K \rangle$  evolves into  $\mathbb{V}' = \langle B_\mathcal{E}, K' \rangle$ , the points of  $\mathcal{P}_\mathcal{E}$  that move into the matched part define the set of points that are *consumed* over  $\mathbb{V} \rightarrow \mathbb{V}'$ , namely:

$$\text{consumed}(\mathbb{V}, \mathbb{V}') = \{x \in \text{dom}(\mathcal{P}_\mathcal{E}) \mid y \preceq_{B_\mathcal{E}} x \preceq_{B_\mathcal{E}} y' \text{ for some } y \in K, y' \in K'\}.$$

We say that a state variable  $x \in \text{SV}$  is *open* in  $\mathbb{V}$  if there is some  $i \in \{0, \dots, n\}$  and some  $k \in K$  such that  $x_i = x$  and  $\text{start}(a_i) \preceq_{B_\mathcal{E}} k \preceq_{B_\mathcal{E}} \text{end}(a_i)$ , i.e., the frontier says that the start of a token for  $x$  has matched but its end has not.

Depending on which symbol is currently being read, only some of the possible evolutions of a viewpoint are admissible.

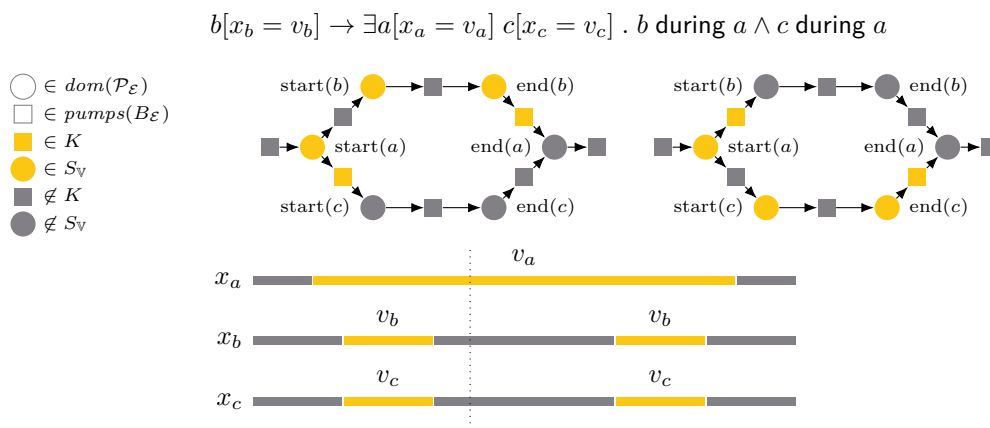
► **Definition 8 (Evolution of a viewpoint).** Let  $\mathbb{V} = \langle B_\mathcal{E}, K \rangle$  and  $\mathbb{V}' = \langle B_\mathcal{E}, K' \rangle$  be two viewpoints over the blueprint  $B_\mathcal{E}$  of some existential statement  $\mathcal{E}$  and let  $\sigma \in \Sigma_P$ . We say that  $\mathbb{V}$  can evolve into  $\mathbb{V}'$  when reading  $\sigma$ , written  $\mathbb{V} \xrightarrow{\sigma} \mathbb{V}'$ , if the following conditions hold:

1. if  $\sigma(x) \neq \circ$  and  $x$  is open in  $\mathbb{V}$ , then  $\text{end}(a_i) \in \text{consumed}(\mathbb{V}, \mathbb{V}')$  for some  $i$  with  $x_i = x$ ,
2. if  $T_{x_i}(v_i) = \emptyset$ , then  $\text{end}(a_i) \notin \text{consumed}(\mathbb{V}, \mathbb{V}')$ ,
3.  $\text{consumed}(\mathbb{V}, \mathbb{V}')$  is compatible with  $\sigma$ , that is:
  - a.  $\sigma(x_i) = v_i$  for every  $\text{start}(a_i) \in \text{consumed}(\mathbb{V}, \mathbb{V}')$ ,
  - b.  $\sigma(x_i) \neq \circ$  for every  $\text{end}(a_i) \in \text{consumed}(\mathbb{V}, \mathbb{V}')$ , and
  - c. if  $\text{start}(a_i) \in \text{consumed}(\mathbb{V}, \mathbb{V}')$ , then  $\text{end}(a_i) \notin \text{consumed}(\mathbb{V}, \mathbb{V}')$ .

### 3.3 Automaton construction

The above notions allow us to build the automaton  $\mathcal{A}_P$  for a given qualitative timeline-based planning problem  $P = (\text{SV}, S)$ . The states of the automaton consist of sets of viewpoints over the rules of  $P$ . However, in order to keep the size of the states small, some combinations of viewpoints are excluded *a priori* by forcing a total order on the viewpoints of a state that are built on the same blueprint.

Formally, let  $\mathbb{V} = \langle B_\mathcal{E}, K \rangle$  and  $\mathbb{V}' = \langle B_\mathcal{E}, K' \rangle$  be two viewpoints over the same blueprint. We define  $\mathbb{V} \preceq \mathbb{V}'$  if and only if for every  $k \in K$  there is a  $k' \in K'$  such that  $k \preceq_{B_\mathcal{E}} k'$ . Let  $\mathbb{V}_\mathcal{E}^{\text{max}} = \langle B_\mathcal{E}, \text{maximals}(B_\mathcal{E}) \rangle$ . Then,  $\mathbb{V}$  is said to be *final* for  $B_{\mathcal{E}_i}$  if  $\text{consumed}(\mathbb{V}, \mathbb{V}_\mathcal{E}^{\text{max}})$  contains only elements of the form  $\text{end}(a)$  for some  $a \in \mathcal{N}$ .



■ **Figure 2** Two incomparable viewpoints for the same rule (on the top) and a plan where the rule is triggered twice. The current time-point in the plan is represented by the dotted line. The left (resp., right) viewpoint takes care of the satisfaction of the rule when triggered by the first (resp., second) occurrence of  $v_b$ . The incomparability is due to the left viewpoint trying to satisfy a rule triggered by an earlier occurrence of  $v_b$  by using a latter occurrence of  $v_c$  and, vice versa, the right viewpoint trying to satisfy a rule triggered by a latter occurrence of  $v_b$  by using an earlier occurrence of  $v_c$ . We will show that these situations can be avoided.

Now, let  $\Upsilon_P$  be the set of all the viewpoints of the existential statements of the rules of  $P$ . The automaton  $\mathcal{A}_P$  is defined as follows.

► **Definition 9 (Automaton construction).** Let  $P = (\text{SV}, S)$  be a qualitative timeline-based planning problem. The NFA  $\mathcal{A}_P = (Q_P, \Sigma_P, q_P^I, Q_P^F, \Delta_P)$  associated with  $P$  is such that:

1. the set of states consists of the initial state  $q_P^I \notin \Upsilon_P$  and a selection of the subsets of  $\Upsilon_P$ :

$$Q_P = \{q_P^I\} \cup \{\Upsilon \subseteq \Upsilon_P \mid \forall \mathbb{V} \preceq \mathbb{V}' \text{ or } \mathbb{V}' \preceq \mathbb{V} \text{ for all } \mathbb{V} = \langle B_\mathcal{E}, K \rangle, \mathbb{V}' = \langle B_\mathcal{E}, K' \rangle \in \Upsilon\};$$

2. the final states  $Q_P^F$  are defined as follows:
  - a.  $\Upsilon \in Q_P^F$  if and only if  $\Upsilon$  is made of final viewpoints and for every triggerless rule  $\top \rightarrow \mathcal{E}_1 \vee \dots \vee \mathcal{E}_k$  in  $S$ , there is  $i \in \{1, \dots, k\}$  such that  $\Upsilon$  contains a final viewpoint for  $B_{\mathcal{E}_i}$ ;
  - b. if there are no triggerless rules, then  $q_P^I \in Q_P^F$ ;
3. for all  $\Upsilon, \Upsilon' \subseteq Q_P \setminus \{q_P^I\}$  and  $\sigma \in \Sigma_P$ ,  $(\Upsilon, \sigma, \Upsilon') \in \Delta_P$  iff:
  - a. for every  $\mathbb{V} \in \Upsilon$ , there is a  $\mathbb{V}' \in \Upsilon'$  such that  $\mathbb{V} \xrightarrow{\sigma} \mathbb{V}'$ ;
  - b. for every  $\mathbb{V}' \in \Upsilon'$ , there is a  $\mathbb{V} \in \Upsilon$  such that  $\mathbb{V} \xrightarrow{\sigma} \mathbb{V}'$ ; and
  - c. if there is a synchronization rule  $a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k$  in  $S$  and  $\sigma(x_0) = v_0$ , then there are  $\mathbb{V} \in \Upsilon$  and  $\mathbb{V}' \in \Upsilon'$  such that:
    - $\mathbb{V} = \langle B_{\mathcal{E}_i}, K \rangle$  for some  $i \in \{1, \dots, k\}$ ;
    - $\mathbb{V} \xrightarrow{\sigma} \mathbb{V}'$ ;
    - $\text{start}(a_0) \in \text{consumed}(\mathbb{V}, \mathbb{V}')$ ;
4. for all  $\Upsilon' \subseteq Q_P \setminus \{q_P^I\}$  and  $\sigma \in \Sigma_P$ ,  $(q_P^I, \sigma, \Upsilon') \in \Delta_P$  iff  $\sigma \in \Sigma_P^I$  and  $(\Upsilon^I, \sigma, \Upsilon') \in \Delta_P$  for some set  $\Upsilon^I$  of minimal viewpoints.

Note that if any possible set of viewpoints were a valid state, the size of the automaton would be doubly exponential. Instead, the *symmetry-breaking condition* imposed by Item 1 of Definition 9, i.e., for every  $\Upsilon \in Q_P$  and every  $\mathbb{V} = \langle B_\mathcal{E}, K \rangle, \mathbb{V}' = \langle B_\mathcal{E}, K' \rangle \in \Upsilon$ , we have  $\mathbb{V} \preceq \mathbb{V}'$  or  $\mathbb{V}' \preceq \mathbb{V}$ , allows us to shrink the size of  $\mathcal{A}_P$  to be only exponential in the size of  $P$ . Figure 2 shows an example of incomparable viewpoints. We will show that these situations can be avoided, thus obtaining an automaton of at most *exponential* size.

► **Lemma 10** (Automaton size). *Let  $P$  be a qualitative timeline-based planning problem. The size of its associated automaton  $\mathcal{A}_P$  is at most exponential in the size of  $P$ .*

**Proof.** Let  $P = (\text{SV}, S)$  be a qualitative timeline-based planning problem and consider the set  $Q_P$  of states of its associated automaton  $\mathcal{A}_P$  as defined in Definition 9. For each viewpoint  $\mathbb{V} = \langle B_{\mathcal{E}}, K \rangle$ , let  $S_{\mathbb{V}} = \{x \in \text{dom}(\mathcal{P}_{\mathcal{E}}) \mid \exists k \in K. x \preceq_{B_{\mathcal{E}}} k\}$  be the set of elements covered by  $\mathbb{V}$ , i.e., those elements of the blueprint  $B_{\mathcal{E}}$  that have already been matched over the word, as witnessed by  $\mathbb{V}$ . Notice that  $\mathbb{V} \preceq \mathbb{V}'$  implies  $S_{\mathbb{V}} \subseteq S_{\mathbb{V}'}$ . Moreover, the sets of covered elements for all the viewpoints  $\mathbb{V}$  for a blueprint  $B_{\mathcal{E}}$  form a lattice with regard to set inclusion, where  $\perp = \emptyset$  (which is the set of elements covered by the minimal viewpoint  $\mathbb{V}_{\perp} = \langle B_{\mathcal{E}}, \text{minimals}(B_{\mathcal{E}}) \rangle$ ), and  $\top = \text{dom}(\mathcal{P}_{\mathcal{E}})$  (which is the set of elements covered by the maximal viewpoint  $\mathbb{V}_{\top} = \langle B_{\mathcal{E}}, \text{maximals}(B_{\mathcal{E}}) \rangle$ ). According to the definition of the automaton states (Item 1 of Definition 9), the viewpoints for a blueprint  $B_{\mathcal{E}}$  included in any state  $\Upsilon \in Q_P$  form a total order. Since the number of disjuncts occurring in  $P$  as well as the distance from  $\perp$  to  $\top$  in the aforementioned lattice is polynomial in the size of  $P$  (in fact, it is linear), the size of  $\Upsilon$  is polynomial, and the size of  $Q_P$  is thus at most *exponential* in the size of  $P$ . ◀

### 3.4 Soundness and completeness

Here we prove that the automaton construction of Definition 9 correctly captures qualitative timeline-based planning problems.

Let  $P = (\text{SV}, S)$  be a qualitative timeline-based planning problem that admits a solution plan  $\Gamma = \{\top_x \mid x \in \text{SV}\}$ . For each  $\mathcal{R} \in S$ , if  $\mathcal{R}$  is not triggerless, then we denote by  $\mathcal{T}_{\Gamma}^{\mathcal{R}}$  the set of tokens in  $\Gamma$  triggering  $\mathcal{R}$ ; if  $\mathcal{R}$  is triggerless, we let  $\mathcal{T}_{\Gamma}^{\mathcal{R}} = \{\top_{\mathcal{R}}\}$  (this notation is useful to handle triggerless rules uniformly). Moreover, we denote  $\mathcal{T}_{\Gamma} = \bigcup_{\mathcal{R} \in S} \mathcal{T}_{\Gamma}^{\mathcal{R}}$  the set of all the triggers occurring in  $\Gamma$ , plus one fictitious token  $\top_{\mathcal{R}}$  for each triggerless rule  $\mathcal{R} \in S$ , which is said to be triggered by  $\top_{\mathcal{R}}$ . Now, let  $\mathcal{R} \equiv a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k$  be a rule in  $S$ . Since  $\Gamma$  is a solution plan, for each token  $\tau \in \mathcal{T}_{\Gamma}^{\mathcal{R}}$ , we can identify a disjunct  $\mathcal{R}(\tau) \in \{\mathcal{E}_1, \dots, \mathcal{E}_k\}$  such that  $\mathcal{R}(\tau)$  is satisfied for  $\tau$  in  $\Gamma$ .

In order to link the words accepted by an automaton to the solution plans for  $P$ , we need to specify how *blueprints* are connected to timelines and plans.

► **Definition 11** (Blueprint instantiation). *Let  $P = (\text{SV}, S)$  be a qualitative timeline-based planning problem and let  $\Gamma = \{\top_x \mid x \in \text{SV}\}$  be a solution plan for  $P$ .*

*For every  $\mathcal{R} \in S$  and  $\tau \in \mathcal{T}_{\Gamma}^{\mathcal{R}}$ , a blueprint instantiation for  $\tau$  in  $\mathcal{R}$  is a labeling function  $\mathcal{L}_{\tau}^{\mathcal{R}(\tau)} : \text{dom}(\mathcal{P}_{\mathcal{R}(\tau)}) \rightarrow \mathbb{N}$  that maps every element  $x$  in the domain of the preorder  $\mathcal{P}_{\mathcal{R}(\tau)}$  to a time point  $\mathcal{L}_{\tau}^{\mathcal{R}(\tau)}(x)$  such that:*

1.  $x \preceq_{\mathcal{P}_{\mathcal{R}(\tau)}} y$  implies  $\mathcal{L}_{\tau}^{\mathcal{R}(\tau)}(x) \leq \mathcal{L}_{\tau}^{\mathcal{R}(\tau)}(y)$  for every  $x, y \in \text{dom}(\mathcal{P}_{\mathcal{R}(\tau)})$ ;
2. for every  $\text{start}(a_i), \text{end}(a_i) \in \text{dom}(\mathcal{P}_{\mathcal{R}(\tau)})$ , there is a (unique) token  $\tau' = (x_i, v_i, d) \in \top_{x_i}$  such that  $\text{start-time}(\tau') = \mathcal{L}_{\tau}^{\mathcal{R}(\tau)}(\text{start}(a_i))$  and  $\text{end-time}(\tau') = \mathcal{L}_{\tau}^{\mathcal{R}(\tau)}(\text{end}(a_i))$ .

When clear from the context, we omit the superscript when referring to blueprint instantiations. Intuitively,  $\mathcal{L}_{\tau}$  is a witness of the satisfaction of  $\mathcal{R}$  when triggered by some  $\tau \in \mathcal{T}_{\Gamma}^{\mathcal{R}}$ . We define a partial order over blueprint instantiations such that  $\mathcal{L}_1 \leq \mathcal{L}_2$  if and only if  $\text{dom}(\mathcal{L}_1) = \text{dom}(\mathcal{L}_2)$  and  $\mathcal{L}_1(x) \leq \mathcal{L}_2(x)$  for each  $x \in \text{dom}(\mathcal{L}_1)$ . The following statement is fundamental in proving that the symmetry-breaking condition given for the automaton states in Definition 9 does not affect the completeness of the construction.

► **Lemma 12.** *Let  $P = (\text{SV}, S)$  be a timeline-based planning problem that admits a solution plan  $\Gamma$ . Moreover, given  $\mathcal{R} \in S$ , let  $\tau_1, \tau_2 \in \mathcal{T}_{\Gamma}^{\mathcal{R}}$  be two tokens that trigger  $\mathcal{R}$ , such that  $\text{end-time}(\tau_1) \leq \text{start-time}(\tau_2)$  and  $\mathcal{R}(\tau_1) = \mathcal{R}(\tau_2)$ .*

*There exist two blueprint instantiations,  $\mathcal{L}_{\tau_1}$  for  $\tau_1$  and  $\mathcal{L}_{\tau_2}$  for  $\tau_2$ , such that  $\mathcal{L}_{\tau_1} \leq \mathcal{L}_{\tau_2}$ .*

**Proof.** Since  $\Gamma$  is a solution plan for  $P$ , there are two blueprint instantiations,  $\mathcal{L}_{\tau_1}$  for  $\tau_1$  and  $\mathcal{L}_{\tau_2}$  for  $\tau_2$ . If  $\mathcal{L}_{\tau_1} \not\leq \mathcal{L}_{\tau_2}$ , then we define blueprint instantiation  $\mathcal{L}'_{\tau_1}$  for  $\tau_1$  such that  $\mathcal{L}'_{\tau_1} \leq \mathcal{L}_{\tau_2}$ . Note that, since  $\mathcal{R}(\tau_1) = \mathcal{R}(\tau_2)$ ,  $\text{dom}(\mathcal{L}_{\tau_1}) = \text{dom}(\mathcal{L}_{\tau_2})$ . We define  $\mathcal{L}'_{\tau_1} : \text{dom}(\mathcal{L}_{\tau_1}) \rightarrow \mathbb{N}$  as follows. For every  $x \in \text{dom}(\mathcal{L}_{\tau_1})$ :

$$\mathcal{L}'_{\tau_1}(x) = \begin{cases} \mathcal{L}_{\tau_1}(x) & \text{if } \mathcal{L}_{\tau_1}(x) \leq \mathcal{L}_{\tau_2}(x) \\ \mathcal{L}_{\tau_2}(x) & \text{if } \mathcal{L}_{\tau_2}(x) < \mathcal{L}_{\tau_1}(x) \end{cases}$$

It is clear that  $\mathcal{L}'_{\tau_1} \leq \mathcal{L}_{\tau_2}$ . We have to show that  $\mathcal{L}'_{\tau_1}$  is a blueprint instantiation for  $\tau_1$ . To see that Item 1 of Definition 11 holds, let  $x \preceq_{\mathcal{P}_{\mathcal{R}(\tau_1)}} y$ . We distinguish three cases:

1. if both  $\mathcal{L}'_{\tau_1}(x) = \mathcal{L}_{\tau_1}(x)$  and  $\mathcal{L}'_{\tau_1}(y) = \mathcal{L}_{\tau_1}(y)$  or both  $\mathcal{L}'_{\tau_1}(x) = \mathcal{L}_{\tau_2}(x)$  and  $\mathcal{L}'_{\tau_1}(y) = \mathcal{L}_{\tau_2}(y)$ , then  $\mathcal{L}'_{\tau_1}(x) \leq \mathcal{L}'_{\tau_1}(y)$  holds due to  $\mathcal{L}_{\tau_1}$  and  $\mathcal{L}_{\tau_2}$  being blueprint instantiations;
2. if  $\mathcal{L}'_{\tau_1}(x) = \mathcal{L}_{\tau_1}(x)$  and  $\mathcal{L}'_{\tau_1}(y) = \mathcal{L}_{\tau_2}(y)$ , then  $\mathcal{L}_{\tau_1}(x) \leq \mathcal{L}_{\tau_2}(x)$  holds by definition. By  $x \preceq_{\mathcal{P}_{\mathcal{R}(\tau)}} y$ , we know that  $\mathcal{L}_{\tau_2}(x) \leq \mathcal{L}_{\tau_2}(y)$ , hence  $\mathcal{L}_{\tau_1}(x) \leq \mathcal{L}_{\tau_2}(y)$ , thus  $\mathcal{L}'_{\tau_1}(x) \leq \mathcal{L}'_{\tau_1}(y)$ ;
3. if  $\mathcal{L}'_{\tau_1}(x) = \mathcal{L}_{\tau_2}(x)$  and  $\mathcal{L}'_{\tau_1}(y) = \mathcal{L}_{\tau_1}(y)$ , then  $\mathcal{L}_{\tau_2}(x) < \mathcal{L}_{\tau_1}(x)$  holds by definition. By  $x \preceq_{\mathcal{P}_{\mathcal{R}(\tau)}} y$ , we know that  $\mathcal{L}_{\tau_1}(x) \leq \mathcal{L}_{\tau_1}(y)$ , hence  $\mathcal{L}_{\tau_2}(x) < \mathcal{L}_{\tau_1}(y)$ , thus  $\mathcal{L}'_{\tau_1}(x) \leq \mathcal{L}'_{\tau_1}(y)$ .

To see that Item 2 holds, consider  $\text{start}(a_i), \text{end}(a_i) \in \text{dom}(\mathcal{P}_{\mathcal{R}(\tau_1)})$ . Note that it cannot be the case that  $\mathcal{L}'_{\tau_1}(\text{start}(a_i)) = \mathcal{L}_{\tau_1}(\text{start}(a_i))$  but  $\mathcal{L}'_{\tau_1}(\text{end}(a_i)) = \mathcal{L}_{\tau_2}(\text{end}(a_i))$  (or *vice versa*), because that would imply that  $\mathcal{L}_{\tau_1}(\text{start}(a_i)) \leq \mathcal{L}_{\tau_2}(\text{start}(a_i)) \leq \mathcal{L}_{\tau_2}(\text{end}(a_i)) < \mathcal{L}_{\tau_1}(\text{end}(a_i))$ , which is impossible because, by hypothesis,  $\mathcal{L}_{\tau_1}$  maps  $\text{start}(a_i)$  and  $\text{end}(a_i)$  into the endpoints of a single token  $\tau$ , that cannot contain another token for the same variable. Thus, we have either  $\mathcal{L}'_{\tau_1}(\text{start}(a_i)) = \mathcal{L}_{\tau_1}(\text{start}(a_i))$  and  $\mathcal{L}'_{\tau_1}(\text{end}(a_i)) = \mathcal{L}_{\tau_1}(\text{end}(a_i))$  or  $\mathcal{L}'_{\tau_1}(\text{start}(a_i)) = \mathcal{L}_{\tau_2}(\text{start}(a_i))$  and  $\mathcal{L}'_{\tau_1}(\text{end}(a_i)) = \mathcal{L}_{\tau_2}(\text{end}(a_i))$ , and the thesis follows since  $\mathcal{L}_{\tau_1}$  and  $\mathcal{L}_{\tau_2}$  are blueprint instantiations themselves.  $\blacktriangleleft$

We can now prove the direct correspondence between solution plans and accepted words.

► **Lemma 13.** *Given a qualitative timeline-based planning problem  $P$  and its associated automaton  $\mathcal{A}_P$ , there is a solution plan for  $P$  if and only if  $\mathcal{L}(\mathcal{A}_P) \neq \emptyset$ .*

**Proof.** Let  $P = (\text{SV}, S)$  be a qualitative timeline-based planning problem, and let  $\mathcal{A}_P = (Q_P, \Sigma_P, q_P^I, q_P^F, \Delta_P)$  be the automaton associated with  $P$  by Definition 9. It is easy to see that, given a word  $\bar{\sigma}$  accepted by  $\mathcal{A}_P$ , the plan encoded by  $\bar{\sigma}$  is a solution plan for  $P$ . We thus focus only on the proof of the other direction.

Fix a solution plan  $\Gamma = \{\mathbb{T}_x \mid x \in \text{SV}\}$  for  $P$ , and denote  $\bar{\sigma} = \langle \sigma_0, \dots, \sigma_m \rangle$  the corresponding word. We wish to prove that  $\bar{\sigma}$  is accepted by  $\mathcal{A}_P$ . This direction of the proof needs some care because of the symmetry-breaking condition of Item 1 of Definition 9: we have to prove that it does not make  $\mathcal{A}_P$  lose some essential solutions.

We proceed by inductively defining a particular sequence  $\bar{\Upsilon} = \langle \Upsilon_0, \dots, \Upsilon_{m+1} \rangle$  of sets of viewpoints. Then, we prove that each  $\Upsilon_i$  is a state of  $\mathcal{A}_P$ , and that  $\bar{\Upsilon}$  is an accepting run of  $\mathcal{A}_P$ . We also define (again, inductively) a sequence of covers of  $\mathcal{T}_\Gamma$ , which will be used for the inductive construction of  $\bar{\Upsilon}$ : for  $i \in \{0, \dots, m\}$ , we define the cover  $\{\mathcal{T}_\mathbb{V}^i\}_{\mathbb{V} \in \Upsilon_i}$  of  $\mathcal{T}_\Gamma$ , where, intuitively,  $\mathcal{T}_\mathbb{V}^i$  is the set of tokens in  $\mathcal{T}_\Gamma$  whose satisfaction is being taken care of by  $\mathbb{V}$  in  $\Upsilon_i$ . At first, we define  $\Upsilon_0 = \{(B_{\mathcal{R}(\tau)}, \text{minimals}(B_{\mathcal{R}(\tau)})) \mid \mathcal{R} \in S, \tau \in \mathcal{T}_\Gamma^{\mathcal{R}}\}$  i.e., the set of *minimal* viewpoints over the blueprints of the existential statements involved in the satisfaction of  $P$  by  $\Gamma$ , and we define the cover  $\{\mathcal{T}_\mathbb{V}^0\}_{\mathbb{V} \in \Upsilon_0}$  of  $\mathcal{T}_\Gamma$  as follows: for every  $\mathcal{R} \in S$  and  $\tau \in \mathcal{T}_\Gamma^{\mathcal{R}}$ ,  $\tau \in \mathcal{T}_{(B_{\mathcal{R}(\tau)}, \text{minimals}(B_{\mathcal{R}(\tau)}))}$ . Then, for all  $i \in \{0, \dots, m\}$ , we choose the following elements of the sequence as follows. For each  $\mathbb{V} = \langle B_{\mathcal{E}}, K \rangle \in \Upsilon_i$  and  $\tau \in \mathcal{T}_\mathbb{V}^i$  let us define the set  $F_i^{\tau, \mathbb{V}} = \{x \in \text{dom}(B_{\mathcal{E}}) \mid \mathcal{L}_\tau^{\mathcal{E}}(x) = i\}$ . It is possible to show that for each  $\mathbb{V} \in \Upsilon_i$  and  $\tau \in \mathcal{T}_\mathbb{V}^i$  there is a unique viewpoint, denoted by  $\text{next}(\mathbb{V}, \tau)$ , such that  $\mathbb{V} \xrightarrow{\sigma_i} \mathbb{V}'$  and  $\text{consumed}(\mathbb{V}, \mathbb{V}') = F_i^{\tau, \mathbb{V}}$ . Then, we take  $\Upsilon_{i+1} = \{\text{next}(\mathbb{V}, \tau) \mid \mathbb{V} \in \Upsilon_i, \tau \in \mathcal{T}_\mathbb{V}^i\}$ , and we define the cover  $\{\mathcal{T}_\mathbb{V}^{i+1}\}_{\mathbb{V} \in \Upsilon_{i+1}}$  of  $\mathcal{T}_\Gamma$  as follows: for each  $\mathbb{V} \in \Upsilon_i$  and  $\tau \in \mathcal{T}_\mathbb{V}^i$ ,  $\tau \in \text{next}(\mathbb{V}, \tau)$ .

## 16:10 Complexity of Qualitative Timeline-Based Planning

Now, we argue that each  $\Upsilon_i$  satisfies the symmetry-breaking condition (Item 1 of Definition 9). Let  $\mathbb{V} = \langle B_{\mathcal{E}}, K \rangle, \mathbb{V}' = \langle B_{\mathcal{E}}, K' \rangle \in \Upsilon_i$ , with  $K \neq K'$ , and let  $\tau \in \mathcal{T}_{\mathbb{V}}^i$  and  $\tau' \in \mathcal{T}_{\mathbb{V}'}^i$ . We can suppose *w.l.o.g.* that  $\text{end-time}(\tau) \leq \text{start-time}(\tau')$ . By Lemma 12, we can suppose as well that  $\mathcal{L}_{\tau} \leq \mathcal{L}_{\tau'}$ . Now, let  $S_{\mathbb{V}}$  and  $S_{\mathbb{V}'}$  be the set of elements *covered* by  $K$  and  $K'$ , respectively, *i.e.*,  $S_{\mathbb{V}} = \{x \in \text{dom}(\mathcal{P}_{\mathcal{E}}) \mid \exists k \in K. x \preceq_{B_{\mathcal{E}}} k\}$ . For any  $x \in \text{dom}(\mathcal{P}_{\mathcal{E}})$ ,  $\mathcal{L}_{\tau}(x) \leq i$  iff  $x \in S_{\mathbb{V}}$  and  $\mathcal{L}_{\tau'}(x) \leq i$  iff  $x \in S_{\mathbb{V}'}$ , thanks to the way in which we defined  $\Upsilon_i$ . Then, it follows that  $S_{\mathbb{V}'} \subseteq S_{\mathbb{V}}$ , which implies that  $K$  dominates  $K'$ , hence  $\mathbb{V}' \preceq \mathbb{V}$ .

As a consequence,  $\bar{\Upsilon}$  is a sequence of  $\mathcal{A}_P$  states and we can check that  $(\Upsilon_i, \sigma_i, \Upsilon_{i+1}) \in \Delta_P$  for all  $i \in \{0, \dots, m\}$ . Thus,  $\bar{\Upsilon}$  identifies a run of  $\mathcal{A}_P$  if we replace  $\Upsilon_0$  with  $q_P^I$ .

To conclude the proof, we only need to show that the above run is accepting, *i.e.*, that  $\Upsilon_{m+1}$  contains only *final* viewpoints. This is indeed ensured by construction, since  $\mathcal{L}_{\tau}(x) \leq m$  for every token  $\tau \in \mathcal{T}_{\Gamma}^{\mathcal{R}}$ , every  $\mathcal{R} \in S$ , and every  $x \in \text{dom}(\mathcal{P}_{\mathcal{R}(\tau)})$ . Therefore, the word  $\bar{\sigma}$  is accepted by  $\mathcal{A}_P$ .  $\blacktriangleleft$

We can now finally state our main result.

► **Theorem 14** (Complexity of qualitative timeline-based planning). *Whether a qualitative timeline-based planning problem  $P$  admits a solution plan can be decided in polynomial space.*

**Proof.** Given  $P = (SV, S)$ , let  $\mathcal{A}_P$  be its associated automaton as specified by Definition 9. By Lemma 13, we know that  $\mathcal{L}(\mathcal{A}_P) \neq \emptyset$  if and only if  $P$  admits a solution plan. Then, we can build  $\mathcal{A}_P$  and check for the emptiness of its language, which in turn consists of checking for the reachability of the final states. By Lemma 10, the size of  $\mathcal{A}_P$  is at most *exponential* in the size of  $P$ . Since this automaton can be constructed *on-the-fly* and solving reachability requires logarithmic space in the size of the automaton, we get that the qualitative timeline-based planning can be decided in polynomial space.  $\blacktriangleleft$

## 4 Hardness

In this section, we show that qualitative timeline-based planning is PSPACE-hard. The proof is by a reduction from the emptiness problem for the intersection of  $n$  finite automata that is known to be PSPACE-complete (see [14]).

► **Theorem 15** (Qualitative timeline-based planning is PSPACE-hard). *Let  $P = (SV, S)$  be a qualitative timeline-based planning problem. Deciding whether  $P$  admits any solution plan is PSPACE-hard.*

**Proof.** We provide a reduction from the emptiness problem for the intersection of  $n$  finite automata. For the main definitions on finite automata, we refer the reader to [14].

For  $i \in \{1, \dots, n\}$ , let  $A_i = (\Sigma, Q_i, q_i^0, \delta_i, q_i^*)$  be a deterministic finite automaton, where  $\Sigma$  is a finite alphabet,  $Q_i$  is a finite set of states,  $q_i^0$  is the initial state,  $\delta_i: Q_i \times \Sigma \rightarrow Q_i$  is the transition function, and  $q_i^*$  is the final state.

Denote by  $A = A_1 \times \dots \times A_n$  the finite automaton obtained by the standard construction to capture the intersection of the languages  $\mathcal{L}(A_1), \dots, \mathcal{L}(A_n)$ , where, for  $i \in \{1, \dots, n\}$ ,  $\mathcal{L}(A_i)$  denotes the language accepted by  $A_i$ . Thus,  $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \dots \cap \mathcal{L}(A_n)$ .

We build a *qualitative* timeline-based planning problem  $P$  such that  $P$  admits a solution plan if and only if  $\mathcal{L}(A) \neq \emptyset$ . The overall idea is to model each finite automaton as a different state variable and then express intersection and acceptance by synchronization rules. More specifically,  $P = (SV, S)$  is defined as follows.



The set of state variables is  $SV = \{x_i \mid i \in \{1, \dots, n\}\}$ , *i.e.*, we take a variable  $x_i$  for each finite automaton  $A_i$ , with  $i \in \{1, \dots, n\}$ . Each variable  $x_i$  is equal to  $(V_i, T_i, D_i)$ , where:

1.  $V_i = Q_i \times \Sigma$ ;
2.  $D_i(v) = (1, +\infty)$ , for all  $v \in V_i$ ;
3.  $T_i((q, \sigma)) = \{\delta_i(q, \sigma)\} \times \Sigma$ , for all  $(q, \sigma) \in V_i$ .

The transition function of each state variable mirrors the transition function of the corresponding automaton, while handling the fact that automata are meant to read words over  $\Sigma$  while state variables only represent *state machines*, with no language recognition semantics.

The set  $S$  contains the following synchronization rules, which are designed in such a way that state variables change their values synchronously.

The first rule requires the existence of two sets of tokens, each containing exactly a token from each state variable and such that (i) the first set maps an initial state for each automaton, (ii) the second set maps a final state for each automaton, (iii) the tokens from the first set precede those from the second set, and (iv) the tokens in each set start and end at the same time. Formally:

$$\begin{aligned} \top \rightarrow \bigvee_{\sigma, \sigma' \in \Sigma} \exists a_1^0[x_1 = (q_1^0, \sigma)] \cdots a_n^0[x_n = (q_n^0, \sigma)] a_1^*[x_1 = (q_1^*, \sigma')] \cdots a_n^*[x_n = (q_n^*, \sigma')] . \\ \text{end}(a_1^0) \leq \text{start}(a_1^*) \wedge \bigwedge_{i=1}^{n-1} a_i^0 = a_{i+1}^0 \wedge \bigwedge_{i=1}^{n-1} a_i^* = a_{i+1}^* . \end{aligned} \quad (1)$$

The remaining rules just synchronize the different state variables so that they are aligned over tokens that refer to the same input symbol for the corresponding automaton:

$$a_i[x_i = (q_i, \sigma)] \rightarrow \bigvee_{q_{i+1} \in Q_{i+1}} \exists a_{i+1}[x_{i+1} = (q_{i+1}, \sigma)] . a_i = a_{i+1} \quad (2)$$

for each  $q_i \in Q_i$ ,  $\sigma \in \Sigma$  and  $i \in \{1, \dots, n-1\}$ .

To complete the proof, it suffices to show that the above construction is correct, that is,  $P$  admits a solution plan if and only if  $\mathcal{L}(A) \neq \emptyset$

We first show that if  $\mathcal{L}(A) \neq \emptyset$ , then  $P$  admits a solution plan. To this end, consider a word  $\bar{\sigma} = \sigma^1 \dots \sigma^m$  accepted by  $A$ . By definition, for  $i \in \{1, \dots, n\}$ , we know that  $\bar{\sigma}$  is accepted by  $A_i$ . Let  $\bar{q}_i = \langle q_i^0, \dots, q_i^m \rangle$  be the sequence of states visited along the run of  $A_i$  over  $\bar{\sigma}$ . Since  $\bar{\sigma}$  is accepted by  $A_i$ ,  $q_i^m$  must be the final state  $q_i^*$ .

Now, for all  $i \in \{1, \dots, n\}$ , let:

$$\mathbb{T}_i = \langle (x_i, (q_i^0, \sigma^1), 1), (x_i, (q_i^1, \sigma^2), 1), \dots, (x_i, (q_i^{m-1}, \sigma^m), 1), (x_i, (q_i^m, \sigma^*), 1) \rangle$$

be the timeline corresponding to  $\bar{\sigma}$ . It can be observed that, by construction,  $\mathbb{T}_i$  satisfies the transition function of  $x_i$ . Moreover, the synchronization rule (1) defined above is satisfied, since each timeline  $\mathbb{T}_i$  starts with a token where  $x_i = (q_i^0, \sigma^1)$  and ends with a token where  $x_i = (q_i^*, \sigma^*)$ . Moreover, by construction, the tokens are all of the same duration, and overlapping tokens have the same  $\sigma$  component; thus, the second set of synchronization rules (2) is satisfied as well. Hence, the plan consisting of the  $\mathbb{T}_i$  timelines is a solution plan for  $P$ .

In order to show that if  $P$  admits a solution plan, then  $\mathcal{L}(A) \neq \emptyset$ , consider a solution plan for  $P$  consisting of a set of timelines  $\mathbb{T}_i = \langle \tau_i^1, \dots, \tau_i^{m_i} \rangle$ , one for each  $x_i \in SV$ . By the rules (2), all the tokens of these timelines can be seen as being aligned one over the other forming a grid, where each *column* shares a common symbol  $\sigma$ . Moreover, by the triggerless rule (1), there are two columns of such a grid, say them  $h$  and  $k$ , with  $h < k$ , containing, respectively, the initial states and the final states for the respective automata  $A_i$ . Let  $\bar{\sigma} = \sigma^h \dots \sigma^{k-1}$  be the sequence of symbols occurring in columns  $h$  to  $k-1$  and let  $q_i^j$  be the state of  $A_i$

associated with token  $\tau_i^j$ , for  $i \in \{1, \dots, n\}$  and  $j \in \{h, \dots, k\}$ . Finally, by construction, at each step the transition function of each state variable enforces the token following any token of the form  $(q, \sigma)$  to be of the form  $(q', \sigma')$ , where  $q' = \delta(q, \sigma)$ , *i.e.*, the evolution of the timeline mirrors the transition function of the automaton. Thus, for all  $i \in \{1, \dots, n\}$ ,  $\bar{q}_i = \langle q_i^h, \dots, q_i^k \rangle$  is an accepting run of  $A_i$  over  $\bar{\sigma}$ . Thus, we conclude that  $\bar{\sigma} \in \mathcal{L}(A)$ . ◀

## 5 Concluding remarks

In this paper, we show that the problem of checking the existence of a plan for the *qualitative* fragment of timeline-based planning is PSPACE-complete.

The key step in the decision procedure builds a finite automaton that accepts a word encoding a plan if and only if the plan is a solution of the given instance of the planning problem. The construction is inspired by the one used by Della Monica *et al.* [5] to prove the EXPSPACE-completeness of the general quantitative problem, but adapted to exploit the distinctive features of the qualitative setting. In particular, blueprints were linear orders in the general case, accounting for all possible combinations of distances between pairs of endpoints satisfying the quantitative constraints of the problem. Here, blueprints are preorders, which compactly represent all the possible ways of matching a particular disjunct of a rule, leading to a smaller automaton.

The automata-theoretic construction of our solution has some interesting consequences. It provides a direct algorithm to generate a solution plan by exploiting the standard machinery to decide the emptiness of finite automata and, moreover, it can be used as a basis for interesting future research directions. For example, one may show how to perform model checking of timeline-based systems against Linear Temporal Logic (LTL), still in polynomial space.

The achieved result sheds some light on how a problem of such a high complexity could form the basis of planning systems that have been deployed in real-world scenarios for the last three-decades. The *quantitative* aspect of the problem accounts for a great part of the complexity, and while temporal reasoning is predominant in these applications, the *magnitude* of the involved timestamps does not need to be significantly high. A proper parameterized complexity analysis of the problem would complete the picture in this regard.

Last but not least, the qualitative fragment appears to be a quite expressive language on its own, powerful enough to express an interesting class of linear-time temporal properties including those captured by LTL. It would be interesting to establish the exact relationship with LTL and possibly characterize this logic in terms of a proper fragment of timeline-based planning. From a logical standpoint, the synchronization rules can be seen as a fragment of first-order logic with one successor relation and one quantifier alternation (specifically, with  $\forall\exists$  alternation). A key aspect that can be observed is that disjunctions are only allowed between existentially quantified formulas and, by relaxing this limitation, the complexity lower bound seems to rise to EXPSPACE again even in the qualitative setting.

---

## References

- 1 James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.
- 2 Javier Barreiro, Matthew Boyce, Minh Do, Jeremy Frank, Michael Iatauro, Tatiana Kichkaylo, Paul Morris, James Ong, Emilio Remolina, Tristan Smith, and David Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proceedings of the 4th International Competition on Knowledge Engineering for Planning and Scheduling*, 2012.



- 3 S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. Aspen - automating space mission operations using automated planning and scheduling. In *Proceedings of the International Conference on Space Operations*, 2000.
- 4 Steve A. Chien, Gregg Rabideau, Daniel Tran, Martina Troesch, Joshua Doubleday, Federico Nespoli, Miguel Perez Ayucar, Marc Costa Sitja, Claire Vallat, Bernhard Geiger, Nico Altobelli, Manuel Fernandez, Fran Vallejo, Rafael Andres, and Michael Kueppers. Activity-based scheduling of science campaigns for the rosetta orbiter. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 4416–4422. AAAI Press, 2015.
- 5 Dario Della Monica, Nicola Gigante, Angelo Montanari, and Pietro Sala. A novel automata-theoretic approach to timeline-based planning. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning*, pages 541–550. AAAI Press, 2018.
- 6 Alessandro Donati, Nicola Policella, Erhard Rabenau, Giovanni Righini, and Emanuele Tresoldi. An automatic planning and scheduling system for the mars express uplink scheduling problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 41(6):942–954, 2011. doi:10.1109/TSMCC.2011.2114880.
- 7 Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971. doi:10.1016/0004-3702(71)90010-5.
- 8 Simone Fratini, Amedeo Cesta, Andrea Orlandini, Riccardo Rasconi, and Riccardo De Benedictis. Apsi-based deliberation in goal oriented autonomous controllers. In *ASTRA 2011*, volume 11. ESA, 2011.
- 9 Simone Fratini and L. Donati. Apsi timeline representation framework v. 3.0. Technical report, European Space Agency - ESOC, 2011.
- 10 Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. URL: <http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB>.
- 11 Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini. Timelines are expressive enough to capture action-based temporal planning. In Curtis E. Dyreson, Michael R. Hansen, and Luke Hunsberger, editors, *Proceedings of the 23rd International Symposium on Temporal Representation and Reasoning*, pages 100–109. IEEE Computer Society, 2016. doi:10.1109/TIME.2016.18.
- 12 Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini. Complexity of timeline-based planning. In Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith, editors, *Proceedings of the 27th International Conference on Automated Planning and Scheduling*, pages 116–124. AAAI Press, 2017.
- 13 Nicola Gigante, Angelo Montanari, Andrea Orlandini, Marta Cialdea Mayer, and Mark Reynolds. On timeline-based games and their complexity. *Theor. Comput. Sci.*, 815:247–269, 2020. doi:10.1016/j.tcs.2020.02.011.
- 14 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.
- 15 Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - The Planning Domain Definition Language. Technical Report Technical Report TR98003, Yale Center for Computational Vision and Control, 1997.
- 16 Nicola Muscettola. HSTS: Integrating Planning and Scheduling. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 6, pages 169–212. Morgan Kaufmann, 1994.



# A Note on $C^2$ Interpreted over Finite Data-Words

**Bartosz Bednarczyk** 

Computational Logic Group, TU Dresden, Germany  
Institute of Computer Science, University of Wrocław, Poland  
bartosz.bednarczyk@cs.uni.wroc.pl

**Piotr Witkowski** 

Institute of Computer Science, University of Wrocław, Poland  
piotr.witkowski@cs.uni.wroc.pl

---

## Abstract

We consider the satisfiability problem for the two-variable fragment of first-order logic extended with counting quantifiers, interpreted over finite words with data, denoted here with  $C^2[\leq, succ, \sim, \pi_{bin}]$ . In our scenario, we allow for using arbitrary many uninterpreted binary predicates from  $\pi_{bin}$ , two navigational predicates  $\leq$  and  $succ$  over word positions as well as a data-equality predicate  $\sim$ . We prove that the obtained logic is undecidable, which contrasts with the decidability of the logic without counting by Montanari, Pazzaglia and Sala [27]. We supplement our results with decidability for several sub-fragments of  $C^2[\leq, succ, \sim, \pi_{bin}]$ , *e.g.* without binary predicates, without successor  $succ$ , or under the assumption that the total number of positions carrying the same data value in a data-word is bounded by an a priori given constant.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Logic and verification

**Keywords and phrases** Two-variable logic, data-words, VASS, decidability, undecidability, counting

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.17

**Funding** *Bartosz Bednarczyk*: supported by “Diamantowy Grant” no. DI2017 006447.

*Piotr Witkowski*: supported by NCN grant no. 2016/21/B/ST6/01444.

## 1 Introduction

Finite data-words [8], *i.e.* finite words, where each position carries letters from a finite alphabet as well as a data value from some countably-infinite data domain, are ubiquitous in formal verification. They can be used to describe executions of array-accessing programs [1], runs of counter machines [18], outputs of timed systems [9] or database transaction logs [28]. However, reasoning about them is not simple: the main obstacle is the unboundedness of the data domain. We discuss some of the recently proposed approaches to solve the problem.

The first solution is stemming from automata theory. To deal with data-words, the notion of class automata [5, 3], data automata [4], register automata [22] or session automata [7] were proposed. Usually, these are automata equipped with a set of registers, used to store the current data value in the memory. Of course, such registers must be suited to store information of unknown size and must be properly restrained: one can easily fall into a trap that the proposed automata model can simulate zero tests, which usually causes undecidability [26]. Unfortunately, proposed automata models lack good algorithmic properties. By way of example, the emptiness problem for class memory automata is equivalent to reachability in vector-addition systems and hence, non-elementary [16]. Moreover, the model of class automata is not closed under complementation, which results in an undecidable equivalence problem. Some weaker subclasses of class automata were considered *e.g.* in [15].

Thus, in this paper, we rather focus on declarative models like logics. Being aware of the plethora of different automata models proposed in the past, it is not hard to conclude that a similar situation should occur for logics. The most famous frameworks, tailored to reason about data-words, are temporal logics and fragments of first-order logics. The former



© Bartosz Bednarczyk and Piotr Witkowski;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 17; pp. 17:1–17:14

Leibniz International Proceedings in Informatics



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

ones were well-developed in the recent years: *e.g.* LTL with freeze quantifier, which can be used as a logical counterpart of a register, was proposed in [19]. Other examples are the temporal logic of repeating values [18], the PathLog [21] and LTL data quantification [32], just to mention a few of them. As far as the existential monadic second-order logic [6] and first-order logic are considered [4], the logics were rather neglected, probably due to their high complexity or even undecidability. The logics generally allows for quantification over words' positions, to compare elements with navigational predicates and to check whether data values of two elements coincide by means of data-equality predicate. The logics FO or EMSO are immediately undecidable. The only known decidable fragments are the two-variable fragments:  $\text{FO}^2[\textit{succ}, \sim]$ ,  $\text{FO}^2[\leq, \sim]$  and  $\text{FO}^2[\leq, \textit{succ}, \sim]$ , where  $\leq$  is a linear order over words' positions, *succ* is its induced successor relation and  $\sim$  is a data-equality predicate. The first two logics are known to be NEXPTIME-complete [28, 4], while the last one is known to be interreducible to the reachability problem in vector addition systems with states. Our work will focus on extending  $\text{FO}^2$  to make the logic more expressive yet decidable.

We encourage the reader to check the latest surveys on the topic [17, 13] or PhD theses [24, 28, 14] to improve his understanding of the state-of-the-art of the problem and to get a glimpse of the maze of data languages.

## 1.1 Our motivation

We aim at extending the framework of the two-variable logic  $\text{FO}^2$  on data-words to the realm of quantitative properties. Our goal is very modest: we would like to understand the behaviour of  $\text{FO}^2$  under the extensions of counting quantifiers. Such quantifiers can be used to express basic quantitative properties like: “*there are at least five data repartitions in the run of the machine*” or “*each request has exactly one corresponding grant with the same data value*”. The techniques dealing with counting quantifiers were well-developed in the last 10 years, see *e.g.* [29, 30, 12, 11], hence there is a hope that they work well also in the context of data-word reasoning. We hope our work will lay the foundation on an expressive specification language for data-words involving an interplay between counting capabilities and data values.

## 1.2 Our contribution

We study satisfiability problems for  $\text{C}^2[\leq, \textit{succ}, \sim, \pi_{bin}]$ , *i.e.* the two-variable logic with counting quantifiers admitting a linear order predicate  $\leq$ , its induced successor relation *succ*, a data-equality predicate  $\sim$  and a set of uninterpreted binary symbols  $\pi_{bin}$ . Our results are:

- In Section 3 we show that  $\text{C}^2[\leq, \textit{succ}, \sim, \pi_{bin}]$  is undecidable, in sharp contrast to the logic without counting [27]. The proof reuses ideas from [2] on how to encode runs of Minsky Machines on data-words. The key property is the existence of  $\text{C}^2$  formula imposing that a fresh binary relation is a one-to-one matching of domain elements, whilst being a refinement of  $\sim$ . We also discuss how the undecidability result transfers to similar logics, *e.g.* to  $\text{C}^2[\leq, \sim, \pi_{bin}]$ . Negative results are supplemented by several decidability results.
- In Section 4 we show that both  $\text{C}^2[\textit{succ}, \sim, \pi_{bin}]$  and  $\text{C}^2[\leq, \sim]$  logics are NEXPTIME-complete. The NEXPTIME lower-bound is trivially inherited from  $\text{FO}^2$ , but the upper bounds are less trivial. For the former logic, we provide a reduction to the appropriate logic on data-trees [11], for which the NEXPTIME-completeness was recently shown by the second author and his colleagues. For the latter logic, namely, for  $\text{C}^2[\leq, \sim]$ , we show that any satisfiable formula has a model with only exponentially many equivalence classes. Such a property allows us to replace the data-equality tests with equi-satisfiability of polynomially many unary predicates, which encodes the class number in binary. Finally, the tight NEXPTIME upper bound is obtained by employing as a black-box algorithm from [12] for deciding finite satisfiability for the logic on words without data values.

- In Section 5 we deal with the finite satisfiability of  $C^2[\leq, succ, \sim]$ . We employ a counting-quantifier elimination technique to get rid of seemingly more expressive concepts from the logic. The logic  $C^2[\leq, succ, \sim]$  turned out to be VASS-complete, *i.e.* complete for the class of all problems elementarily reducible to the reachability in Vector Addition Systems (solvable in ACKERMANN time [25] with a non-elementary TOWER lower bound [16]).
- Finally, in the last section, we establish the most technically challenging result of this paper, namely the VASS-completeness of  $C^2[\leq, succ, \sim, \pi_{bin}]$  under the restriction that each equivalence class has a uniform bound  $k$  on their sizes. Differently phrased, it means that a single data value can occur in a data-word only, a priori given, constant number of times. In those logics, we allow for using data-equality predicate with  $\sim_k$  instead of the full data-equality  $\sim$ . To solve the satisfiability problem, we propose a translation from  $C^2[\leq, succ, \sim_k, \pi_{bin}]$  to  $C^2[\leq, succ, \pi_{bin}]$ , that is the logic without  $\sim_k$ . The main problem is that transitivity is not expressible with only two variables, and hence we cannot hope for an “easy” translation. To achieve our goal we take an input formula  $\varphi$  and link it with some formulae imposing a colouring of the structure with some fresh letters smartly encoding information to which class given elements belong.

## 2 Preliminaries

Let  $\Sigma$  be a finite *alphabet* (*i.e.* a set of unary predicates) and let  $\mathbb{D}$  be a countably-infinite *data domain*. A *data word* is an element from  $(2^\Sigma \times \mathbb{D})^*$ . A *language* is a set of data words. In our setting, we are interested in fragments of first-order logic describing data-words. We agree that the formulae have direct access to the alphabet  $\Sigma$ , allowing to use the letters as unary predicates. To the contrary, the data-values from  $\mathbb{D}$  are stored implicitly: the only allowed operation is a comparison of data-values between positions with an equivalence relation  $\sim$  called the *data equality predicate*. In the paper, we follow the usual notations [4].

### 2.1 Logics

The two-variable<sup>1</sup> logic  $FO^2[\leq, succ, \sim]$  interpreted over finite data-words is a fragment of first-order logic featuring only two variables  $x, y$  and equipped with a vocabulary of arbitrary many unary predicates (aka letters), two *navigational predicates* over the words’ positions, namely a linear order  $\leq$  and its induced successor relation  $succ$ , and  $\sim$ . Whenever  $x \leq y$  holds, we say that  $x$  is to the left of  $y$ . Additionally, we extend the logic with an arbitrarily large set of uninterpreted binary predicates  $\pi_{bin}$ <sup>2</sup>, forming the logic  $FO^2[\leq, succ, \sim, \pi_{bin}]$ . In this paper, we mostly work with counting extensions of  $FO^2$ , denoted here with  $C^2$ . Such logics extend the previous ones with the so-called counting quantifiers  $\exists^{\geq k}, \exists^{\leq k}$ , with their natural meaning, *i.e.*  $\exists^{\geq k}x.\varphi$  is satisfied in a data-word  $w$  if at least  $k$  positions, when substituted as  $x$ , satisfy  $\varphi$ .

We are interested in the *finite satisfiability problem* phrased as “*given a formula  $\varphi$  is there a data word satisfying  $\varphi$ ?*”. The current state-of-the-art of the problem is presented in the table below. All of the claimed bounds are tight and the appropriate reference is cited (with [H] we indicate that the result is shown in this paper).<sup>3</sup>

<sup>1</sup> With  $\sigma$  in  $\mathcal{L}[\sigma]$  we indicate what kinds of binary relations can be in the logic.

<sup>2</sup> They can be used with counting quantifiers *e.g.* to express Presburger constraints over universes [31].

<sup>3</sup> Recall that VASS complexity class is composed of all problems elementarily reducible to VASS-reachability.

## 17:4 A Note on C2 Interpreted over Finite Data-Words

■ **Table 1** The complexity of the satisfiability problem for  $\text{FO}^2$  and  $\text{C}^2$  over finite data words. All stated complexity bounds are tight.

	$[], [\leq, succ]$	$[succ, \sim, \pi_{bin}]$	$[\leq, \sim]$	$[\leq, succ, \sim]$	$[\leq, \sim, \pi_{bin}]$	$[\leq, succ, \sim, \pi_{bin}]$
$\text{FO}^2$	NEXP [20]	NEXP [28]	NEXP [4]	VASS [4]	NEXP [27]	VASS [27]
$\text{C}^2$	NEXP [12]	NEXP [11]	NEXP [H]	VASS [H]	Undecidable [H]	

### 2.2 Normal Forms

It is usually very convenient to work with the formulae in tailored normal forms. In the paper we will present two of them. Reducing a formula into such forms is usually simple and requires well-known techniques, *cf.* [23, 29]. Hence, routine proofs are omitted.

We employ two types of *Scott-normal forms* for  $\text{C}^2$ , the latter being tailored especially for construction in Section 5. In the remaining sections we employ weak normal forms. Their main advantage is that they are computable in polynomial time *cf.* [12].

$$\varphi = \forall x \forall y \chi \wedge \bigwedge_{i=1}^n \forall x \exists^{\bowtie_i} C_i y \chi_i, \quad (1)$$

with  $\bowtie_i \in \{\leq, \geq\}$ , quantifier-free  $\chi, \chi_i$  and with all  $C_i$  being natural numbers. A *1-type* is a maximal consistent set of literals over  $\Sigma$  involving only the variable  $x$ . Note that the number of 1-types over  $\Sigma$  is exponential in the size of  $\Sigma$ . Likewise, a *2-type* is a maximal consistent set of literals over  $\Sigma$  involving only the variables  $x$  and  $y$  and containing the literal  $x \neq y$ . In Section 5 we use the following normal form.

$$\varphi = \forall x \forall y \alpha \wedge \bigwedge_{i=1}^n \forall x (\pi_i(x) \rightarrow \exists^{\bowtie_i} C_i y \beta_i) \wedge \bigwedge_{i=1}^{n'} \forall x (\pi'_i(x) \rightarrow \exists^{\bowtie_i} C'_i y \gamma_i), \quad (2)$$

where  $\alpha$  is quantifier-free formula,  $\bowtie_i \in \{\leq, =, \geq\}$ ,  $\pi_i, \pi'_i$  are 1-types and  $\beta_i, \gamma_i$  are 2-types and each  $\beta_i$  contains  $x \sim y$  and each  $\gamma_i$  contains  $x \not\sim y$ . Its main feature is the presence of 1-types and 2-types, *i.e.* since each element has a unique 1-type, the types and location of its witnesses  $y$  are given explicitly in the 2-types  $\beta_i$ .

### 3 Undecidability of the full logic

For a moment we move to a slightly more general framework, namely, we assume that each position of a data word carries a pair of data  $(d_1, d_2)$  from a product of two countably infinite sets  $\mathbb{D}_1$  and  $\mathbb{D}_2$ , rather than just a single datum. In this scenario, we allow to use two equivalence relations  $\sim_1$  and  $\sim_2$ , responsible, respectively, for the data tests of first and of the second coordinate. It is known that even the most natural logic for this setting, namely  $\text{FO}^2[\leq, succ, \sim_1, \sim_2]$ , becomes immediately undecidable [4]. Moreover, the  $\text{FO}^2$  logic remains undecidable even when the second datum is treated as a refinement of the first one, *i.e.* when a formula  $\forall x \forall y x \sim_2 y \rightarrow x \sim_1 y$  is a tautology [2]. Here we explain how to modify the undecidability proof from [2, Appendix A.1] to infer undecidability of  $\text{C}^2[\leq, succ, \sim, \pi_{bin}]$ .

To prove undecidability of  $\text{FO}^2[\leq, succ, \sim_1, \sim_2]$  (under the proviso that  $\sim_2$  is a refinement of  $\sim_1$ ), the authors of [2] provided a reduction from the halting problem for Minsky Machines [26]. They encoded successful runs of a machine as data words from  $\mathcal{L}$ , where:

$$\mathcal{L} = s_1 s_2 (i_1 + i_2 + d_1 + d_2 + e_1 s_1 + e_2 s_2)^* e_2 e_1.$$

An intuition behind such language is fairly simple: the letters  $i_k$  and  $d_k$  correspond to the incrementation and the decrementation of the  $k$ -th counter, while the letters  $s_k$  and  $e_k$

correspond to zero tests. Then the subwords composed of all positions between each  $s_k$  and  $e_k$  are assumed to have the equal first datum, *i.e.* are in the same  $\sim_1$  equivalence class. As the next step, the relation  $\sim_2$  was employed to match each incrementation  $i_k$  with an appropriate  $d_k$  from the same  $\sim_1$ -class. Finally, consistency between two neighbouring configurations was handled with a two-variable formula without any data-equality predicates.

Note that the equivalence relation  $\sim_2$  was only used to match occurrences of  $i_k$  with occurrences of  $d_k$  and vice-versa. The same property can be stated with a single one-to-one binary relation required to be a subset of  $\sim_1$ . And such a property is easily expressible in  $C^2$ :

$$\forall x (\forall y x \sim_2 y \rightarrow x \sim y) \wedge \forall x (\exists^{\leq 1} y x \sim_2 y \wedge \exists^{\leq 1} y y \sim_2 x)$$

With such an interpretation of  $\sim_2$ , the undecidability proof of [2] can be read without any changes as an undecidability proof for  $C^2[\leq, succ, \sim, \pi_{bin}]$ . Thus we conclude:

► **Theorem 1.** *Satisfiability of  $C^2[\leq, succ, \sim, \pi_{bin}]$  over finite data-words is undecidable, even if  $\pi_{bin}$  contains only a single binary relation and the only allowed counting quantifier is  $\exists^{\leq 1}$ .*

Note that in the presence of uninterpreted binary symbols in the language, the successor relation  $succ$  can be defined in  $C^2[\leq, \sim, \pi_{bin}]$  cf. [12, Lemma 3.1]. Hence we can also infer the undecidability of the logic without the successor relation.

► **Theorem 2.** *Satisfiability of  $C^2[\leq, \sim, \pi_{bin}]$  over finite data-words is undecidable.*

#### 4 When only one navigational binary relation is allowed

As a first step towards decidability, we consider sublogics of  $C^2[\leq, succ, \sim, \pi_{bin}]$  without uninterpreted binary symbols  $\pi_{bin}$  and with only a single binary navigational predicate.

For the case when only the  $succ$  relation is allowed, we reuse a recent result on  $C^2$  interpreted over trees with data. It was shown in [11] that the logic  $C^2[\downarrow, \sim, \pi_{bin}]$ , namely  $C^2$  with two distinguished relations interpreted, respectively, as a parent-child relation in a tree and as an equivalence relation is NEXPTIME-complete. Note that a word can be seen as a tree, where each node has at most one child. Moreover, by employing the formula  $\forall x \exists^{\leq 1} y x \downarrow y$  we can enforce that the intended tree models are actually words. Hence from [11] we conclude:

► **Theorem 3.** *The satisfiability for  $C^2[succ, \sim]$  and  $C^2[succ, \sim, \pi_{bin}]$  is NEXPTIME-complete.*

To obtain a tight NEXPTIME upper bound for the next logic, namely for  $C^2[\leq, \sim]$ , we closely follow the line of NEXPTIME-completeness proof for  $FO^2[\leq, \sim]$  from [4, Lemma 19].

We first show that any satisfiable  $C^2[\leq, \sim]$  formula  $\varphi$  has a model with at most exponentially many equivalence classes. This is done by taking an arbitrary model and performing some surgery on it. More precisely, we first mark an appropriate number of equivalence classes at the beginning (together with an appropriate number of their elements) as well as on the end. Then, if any non-marked element needs a witness, it should find one in an equivalence class of some marked element. Once such a lemma is shown, we can assign some number to each of the equivalence classes. Since there are only exponentially many of them, their numbers can be encoded with only polynomially many bits represented with only polynomially many fresh unary predicates. Thus in that setting, testing whether two positions carry the same data-value boils down to checking the number of their equivalence classes and it can be handled easily in  $FO^2$ . Finally, we rewrite the formula into a  $\sim$ -free one and use a black-box an NEXPTIME algorithm for solving  $C^2[\leq]$  from [12]. Now we show:

► **Lemma 4.** *Any satisfiable  $C^2[\leq, \sim]$ -formula  $\varphi$  has a model, in which the total number of  $\sim$ -equivalence classes  $eq(\varphi)$  is bounded exponentially in  $|\varphi|$ .*



**Proof.** Assume that  $\varphi$  is in the weak normal form (cf. Eq. 1). Let  $C$  be the maximal number appearing in the counting quantifiers and let  $t$  be the number of all possible 1-types over the vocabulary of  $\varphi$ . Note that both  $C$  and  $t$  are exponential in  $|\varphi|$ . In the forthcoming proof, we will show how to construct a model of  $\varphi$  with at most  $t \cdot 2(C+1)$  different classes.

Let  $\mathfrak{A}$  be a model of  $\varphi$ . For each 1-type  $\alpha$  we mark the first  $C+1$  positions of  $\mathfrak{A}$  with type  $\alpha$  from mutually different classes [or all of them if there are less than  $C+1$  of them in  $\mathfrak{A}$ ]. Analogously we repeat the process for the last  $C+1$  positions of type  $\alpha$ . Let  $\mathfrak{B}$  be a subword of  $\mathfrak{A}$  composed of only those positions of  $\mathfrak{A}$ , which has the same data as some marked element.

We will show that  $\mathfrak{B} \models \varphi$ . Since the described construction preserves 1-types, we conclude that  $\mathfrak{B}$  satisfies the  $\forall x \forall y \chi$  part of  $\varphi$  (because the satisfaction of  $\chi$  depends only on 1-types realized in a model). Moreover, the satisfaction of all subformulae of the form  $\forall \exists^{\leq C_i}$  are preserved too, due to the fact that  $\mathfrak{B}$  is a substructure of  $\mathfrak{A}$ . The tricky part here is show preservation of satisfaction of  $\forall x \exists^{\geq C_i} y \chi_i(x, y)$  formulae. Take an arbitrary position  $p$  from  $\mathfrak{B}$  and consider what kind of witnesses  $y$  it has in  $\mathfrak{A}$  to satisfy  $\chi(x, y)$ . All possible  $y$  from the same class as  $p$  are preserved in the construction, so they can still serve as witnesses for  $p$ . It could be also the case that  $p$  had  $k$  (where  $k \leq C$ ) witnesses from a different class, to the right of  $p$ . But since at least  $k$  classes were marked during the construction, then  $p$  can take as witnesses some  $k$  elements from those marked classes (in the worst case such elements coincide with the original ones). For witnesses to the left of  $p$  we proceed analogously. Thus, by considering all sub-cases, we infer  $\mathfrak{B} \models \varphi$ . The total number of different classes in  $\mathfrak{B}$  is bounded by  $t \cdot 2(C+1)$ , and hence is only exponential in  $|\varphi|$ .  $\blacktriangleleft$

Let  $p_0, p_1, \dots, p_m$  be fresh unary predicates, such that  $2^{m+1} \geq eq(\varphi) > 2^m$  holds for  $eq(\varphi)$  obtained from the above lemma. As we have already mentioned, once the number of equivalence classes is bounded, checking whether two elements  $x$  and  $y$  are related by  $\sim$  boils down to checking whether they encode the same number on  $p_i$  predicates. Hence, we can replace all subformulae of the form  $x \sim y$  in  $\varphi$  with a formula  $\bigwedge_{i=0}^m (p_i(x) \leftrightarrow p_i(y))$ . The formulae obtained in this way are (purely)  $C^2[\leq]$  formulae and are of polynomial size. Thus by employing an NEXPTIME algorithm for deciding fin-sat of  $C^2[\leq]$  from [12] we obtain:

► **Theorem 5.** *Satisfiability for  $C^2[\leq, \sim]$  over finite data-words is NEXPTIME-complete.*

## 5 When uninterpreted relations are disallowed

In this section, we focus on the most expressive variant of data logics without binary predicates, namely on  $C^2[\leq, succ, \sim]$ . It is known that its  $FO^2$  version is VASS-complete [4]. Here we show that the VASS-completeness transfers also to its  $C^2$  counterpart, which will be done by a model-preserving translation from  $C^2[\leq, succ, \sim]$  to  $FO^2[\leq, succ, \sim]$ . Note that since  $FO^2[\leq, succ, \sim]$  is non-elementary, we do not need to care too much about how complex complexity-wise the reduction will be, as long as its size is bounded by some elementary function. Before we start, we will assume that the input formula is in the Scott-like normal form (2) defined in Section 2.2. Our plan is to gradually remove all  $\forall \exists^{\bowtie}$  conjuncts from  $\varphi$ , replacing them with some equisatisfiable formulae without counting quantifiers. Let  $C = 1 + \max_{i=1}^n \{C_i\}$  and let us proceed as follows. Observe that any  $\forall \exists^{\bowtie} \psi$  conjunct requires, for a fixed  $x$ , at most  $C$  witnesses for its satisfaction. Hence, once we would know in advance how many witnesses for  $\psi$  the element  $x$  has, we would immediately know whether the  $\forall \exists^{\bowtie} \psi$  formula is satisfied or not. Thus, we aim at providing such information. In order to do that, we introduce fresh unary predicates labelling the elements of the model, both globally and locally in every equivalence class, numbering occurrences the certain 1-types (from the start and from the end of the model) up to the threshold  $C$ . It will suffice to eliminate the counting.



To explain the technique, let us first consider the case of  $\bigwedge_{i=1}^n \forall x (\pi_i(x) \rightarrow \exists^{\infty_i C_i} y \beta_i)$  conjuncts, which we prefer to call *class conjuncts*, since they speak about witnesses  $y$  from the same equivalence class as  $x$ . For each 1-type  $\pi$  and  $i \in \{1, 2, \dots, C+1\}$  we introduce fresh unary predicates  $cl\text{-}left_i^\pi$  and  $cl\text{-}right_i^\pi$  and we impose their interpretation, e.g. that  $cl\text{-}left_i^\pi(x)$  holds iff  $x$  is the  $i$ -th occurrence (counted from 1 from the beginning of the model) of the 1-type  $\pi$  in the equivalence class of  $x$ . Writing the formulae imposing such interpretation is easy, e.g. to impose that  $cl\text{-}left_2^\pi$  means the second occurrence of the type  $\pi$ , we write:

$$\forall x cl\text{-}left_2^\pi(x) \leftrightarrow (\pi(x) \wedge \exists y.(y < x \wedge y \sim x \wedge \pi(y)) \wedge \forall y(y < x \wedge y \sim x \wedge \pi(x) \rightarrow cl\text{-}left_1^\pi(y)))$$

► **Fact 6.** *There is an  $\text{FO}^2[\leq, succ, \sim]$  formula  $\varphi_{cl}$  such that for every model  $\mathfrak{A} \models \varphi_{cl}$  and every  $1 \leq i \leq C$  we have that  $cl\text{-}left_i^\pi(x)$  (resp.  $cl\text{-}right_i^\pi(x)$ ) holds iff  $x$  is the  $i$ -th occurrence from the beginning of the model (resp. the end) of the 1-type  $\pi$  in the equivalence class of  $x$ .*

The above fact allows us to eliminate counting quantifiers from the class conjuncts from  $\varphi$ . By way of example, consider the formula  $\pi(x) \rightarrow \exists^{\leq C_i} y. (\pi'(y) \wedge x \sim y \wedge y < x \wedge \neg succ(x, y))$ , which states that each  $x$  of the 1-type  $\pi$  should see at most  $C_i$  elements of the type  $\pi'$  (in its equivalence class) strictly to its left. By employing Fact 6 we can rewrite it into:  $\pi(x) \rightarrow \neg \exists y.(y < x \wedge x \sim y \wedge \neg succ(x, y) \wedge cl\text{-}left_{C_i+1}^\pi(x))$ . Other cases are treated similarly.

► **Lemma 7.** *Any  $\text{C}^2[\leq, succ, \sim]$  formula  $\varphi$  in the normal form can be transformed into equisatisfiable  $\text{C}^2[\leq, succ, \sim]$  formula  $\varphi'$  without counting quantifiers in the class conjuncts.*

Now we discuss how to eliminate counting quantifiers in the non-class conjuncts. The method will be similar to the previous one, but the introduced labelling will be more involved. By way of example, consider the formula  $\pi(x) \rightarrow \exists^{\geq C_i} y. (\pi'(y) \wedge x \not\sim y \wedge x < y \wedge \neg succ(y, x))$ , which states that each  $x$  of the 1-type  $\pi$  requires at least  $C_i$  witnesses, outside the equivalence class of  $x$ , of the 1-type  $\pi'$  strictly to the right of  $x$ . It would be tempting to claim that the global labelling of the last  $C$  elements with the 1-type  $\pi'$  would be sufficient for our purposes. Unfortunately, it is not: it could be the case that the last  $C$  elements are in the same class. To omit such difficulties, we label up  $C^2$  elements with the type  $\pi$  in total (from the left and from the right) with the predicates  $gl\text{-}left_i^\pi, gl\text{-}right_i^\pi$ , but we require that no more than  $C$  elements from the same class is marked (i.e. in our numbering we simply skip elements from the class containing  $C$  labelled elements). In means that if an element needs to find witnesses from outside of its class, it should find them among the marked elements. Once again, providing such a labelling is an easy exercise in  $\text{FO}^2[\leq, succ, \sim]$ .

► **Fact 8.** *There is an  $\text{FO}^2[\leq, succ, \sim]$  formula  $\varphi_{gl}$  such that for every model  $\mathfrak{A} \models \varphi_{gl}$  and every  $1 \leq i \leq C^2$  we have that  $gl\text{-}left_i^\pi(x)$  (resp.  $gl\text{-}right_i^\pi(x)$ ) holds iff  $x$  is the  $i$ -th occurrence from the beginning of the model (resp. the end) of the 1-type  $\pi$ , skipping in the enumeration all the elements already having  $C$  elements labelled with some  $gl\text{-}left_j^\pi(x)$  (resp.  $gl\text{-}right_j^\pi(x)$ ) in their equivalence class.*

Now we will discuss how to employ such a labelling to eliminate counting quantifiers in the non-class conjuncts. Recall the toy formula:  $\pi(x) \rightarrow \exists^{\geq C_i} y. (\pi'(y) \wedge x \not\sim y \wedge x < y \wedge \neg succ(y, x))$ . We need to state that an element  $x$  can see at least  $C$  elements of the 1-type  $\pi'$  to its right, outside its equivalence class. Observe that we already enumerated elements of the 1-type  $\pi'$  inside the equivalence class of  $x$ . Hence if there are  $j$  elements of the type  $\pi'$  to the right of  $x$ , i.e.  $cl\text{-}right_j^{\pi'}(y)$  is satisfied for some  $y > x$  having the same data-value as  $x$ , it suffices to state that  $x$  can see to its right an element labelled with  $gl\text{-}right_{C_i+j}^\pi$ . And this can be defined with an  $\text{FO}^2[\leq, succ, \sim]$  formula. By applying analogous reasoning, one can eliminate counting quantifiers also in the other cases. Hence we conclude the following lemma:

► **Lemma 9.** *Any  $C^2[\leq, succ, \sim]$  formula  $\varphi$  in the normal form can be transformed into equisatisfiable  $C^2[\leq, succ, \sim]$  formula  $\varphi'$  without counting quantifiers in the non-class conjuncts. Moreover,  $\varphi'$  does not introduce any counting quantifiers in the class conjuncts.*

By employing Lemma 7, Lemma 9 and VASS-completeness of  $FO^2[\leq, succ, \sim]$  we establish the main theorem of this section.

► **Theorem 10.** *For any  $C^2[\leq, succ, \sim]$  formula  $\varphi$  there exists an equisatisfiable  $FO^2[\leq, succ, \sim]$  formula  $\varphi'$  of an elementary size in  $|\varphi|$  and hence,  $C^2[\leq, succ, \sim]$  is VASS-complete.*

## 6 $C^2$ with full linear order and bounded data-tests

In this section we prove that the decidability of the full logic can be regained, under a reasonable assumption that no more than  $k$  (for a fixed number  $k$ ) elements in the model share the same data-value. To express such a restriction in the logical terms, we employ the relation  $\sim_k$ , interpreted as an equivalence relation with equivalence classes of size at most  $k$ . We show that the logic  $C^2[\leq, succ, \sim_k, \pi_{bin}]$  is VASS-complete. The proof goes via a reduction to  $C^2[\leq, succ, \pi_{bin}]$ . Since the latter logic is VASS-complete [12] we conclude the result.

More precisely, given a  $C^2[\leq, succ, \sim_k, \pi_{bin}]$  formula  $\varphi$  we will produce an equisatisfiable  $C^2[\leq, succ, \pi_{bin}]$  formula  $\varphi_{tr}$  by adding to  $\varphi$  conjuncts that encode some  $\sim_k$  properties and enable model transformations that preserve satisfiability of  $\varphi$  and the interpretations of  $\leq$  and  $succ$ . The essential part of the reduction will be to use these transformations on an arbitrary model of  $\varphi_{tr}$  to produce a model of  $\varphi$  in which  $\sim_k$  is interpreted as a bounded equivalence relation. By  $\mathcal{W}(\leq, succ, \pi_{bin})$  denote the class of all words and by  $\mathcal{W}(\leq, succ, \sim_k, \pi_{bin})$  its subclass where  $\sim_k$  is interpreted as described above.

### 6.1 Plethora of types

We make extensive use of the notions of (atomic) 1- and 2-types. In both cases, we take the notion of consistency to incorporate the constraint that the distinguished predicate  $\sim_k$  is interpreted as a reflexive and a symmetric relation (note that transitivity would require three variables and thus cannot be enforced in the same way). If  $\tau$  is a 2-type, we denote by  $\tau^{-1}$  the 2-type obtained by exchanging the variables  $x$  and  $y$  in  $\tau$ , and call  $\tau^{-1}$  the *inverse* of  $\tau$ . We denote by  $tp_1(\tau)$  the 1-type obtained by removing from  $\tau$  any literals containing  $y$ ; and we denote by  $tp_2(\tau)$  the 1-type obtained by first removing from  $\tau$  any literals containing  $x$ , and then replacing all occurrences of  $y$  by  $x$ . Evidently,  $tp_2(\tau) = tp_1(\tau^{-1})$ . We equivocate freely between finite sets of formulae and their conjunctions; thus, we treat 1-types and 2-types as formulae, where convenient. Let  $\mathfrak{A}$  be any structure interpreting  $\Sigma$ . If  $a \in A$ , then there exists a unique 1-type  $\pi$  such that  $\mathfrak{A} \models \pi[a]$ ; we denote  $\pi$  by  $tp^{\mathfrak{A}}[a]$  and say that  $a$  *realizes*  $\pi$ . If, in addition,  $b \in A \setminus \{a\}$ , then there exists a unique 2-type  $\tau$  such that  $\mathfrak{A} \models \tau[a, b]$ ; we denote  $\tau$  by  $tp^{\mathfrak{A}}[a, b]$  and say that the pair  $a, b$  *realizes*  $\tau$ . Evidently, in that case,  $\tau^{-1} = tp^{\mathfrak{A}}[b, a]$ ;  $tp_1(\tau) = tp^{\mathfrak{A}}[a]$ ; and  $tp_2(\tau) = tp^{\mathfrak{A}}[b]$ . For a fixed  $C^2$  formula in normal form (1) a  $\varphi$ -ray-type is a 2-type  $\rho$  such that  $\models \rho \rightarrow \bigvee_{h=1}^n \chi_h$ . If  $\mathfrak{A} \models \rho[a, b]$  for distinct elements  $a, b$ , then we say that the pair  $\langle a, b \rangle$  is a  $\varphi$ -ray. We call a  $\varphi$ -ray-type  $\rho$   $\varphi$ -invertible if  $\rho^{-1}$  is also a  $\varphi$ -ray-type. We call a 2-type  $\tau$   $\varphi$ -silent if neither  $\tau$  nor  $\tau^{-1}$  is a  $\varphi$ -ray-type.

We now construct an apparatus for describing the “local environment” of elements in structures. Let the  $\varphi$ -ray-types be listed in some fixed order (depending on  $\Sigma$ ) as  $\rho_1, \dots, \rho_J$ . A  $\varphi$ -star-type is an  $(J+1)$ -tuple  $\sigma = \langle \pi, v_1, \dots, v_J \rangle$ , where  $\pi$  is a 1-type over  $\Sigma$  and the  $v_j$  are non-negative integers such that  $v_j \neq 0$  implies  $tp_1(\rho_j) = \pi$  for all  $j$  ( $1 \leq j \leq J$ ). We denote the 1-type  $\pi$  by  $tp(\sigma)$ . To motivate this terminology, suppose  $\mathfrak{A}$  is a structure interpreting  $\Sigma$ . For any

$a \in A$ , we define  $\text{st}^{\mathfrak{A}}(a) = \langle \text{tp}^{\mathfrak{A}}[a], v_1, \dots, v_J \rangle$ , where  $v_j = |\{b \in A : b \neq a \text{ and } \text{tp}^{\mathfrak{A}}[a, b] = \rho_j\}|$ . Evidently,  $\text{st}^{\mathfrak{A}}[a]$  is a star-type; we call it the  $\varphi$ -star-type of  $a$  in  $\mathfrak{A}$ , and say that  $a$  realizes  $\text{st}^{\mathfrak{A}}[a]$ . Intuitively, the star-type of an element records the number of rays of each type emitted by that element. It helps to think, informally, of a star-type  $\sigma$  as *emitting* a collection of rays of various types, and of nodes as *accepting* rays. When  $\varphi$  is known from a context or arbitrary, we will simply write ray-, invertible-, silent- or star-type instead of  $\varphi$ -ray-,  $\varphi$ -invertible-,  $\varphi$ -silent- or  $\varphi$ -star-type. We say that a structure  $\mathfrak{A}$  realizes a set of 2-types (resp. star-types)  $\Phi$  if every pair of nodes (resp. every node) in  $\mathfrak{A}$  realizes a 2-type (resp. a star-type) from  $\Phi$ . Importance of the above notions of 2-, ray- and star-types is summarized in the following.

► **Proposition 11.** *Let  $\mathfrak{A}$  be a structure such that  $\mathfrak{A} \models \varphi$ . If  $\mathfrak{B}$  is a structure interpreting the same signature, and realizing the same set of 2-types and the same set of star-types as  $\mathfrak{A}$ , then  $\mathfrak{B} \models \varphi$ .*

Thus, the satisfiability of  $C^2$  formulae is invariant under arbitrary transformations of structures that preserve sets of realized 2-types and star-types. Our transformations are more constrained; for every element of a model they preserve its star-type by only allowing changes of targets of emitted ray-types. Special care must be taken in order not to emit a ray from a source node to a node which already emits a ray back to the source node. Therefore we introduce a restriction allowing to only modify rays that are invertible (rigidity), and another restriction that a node cannot emit an invertible ray-type and another (invertible- or not) ray-type to two nodes with the same 1-type (superchromaticity). This way, we may select an invertible ray-type  $\tau$ , edges  $\tau(e_1, e)$   $\tau(e'_1, e')$  and replace them by edges  $\tau(e'_1, e)$  and  $\tau(e_1, e')$  preserving star-types of all involved nodes and not introducing duplicate rays. Furthermore, during the entire procedure we employ additional precautions to preserve both linear order and its successor.

## 6.2 Towards Vass-completeness of $C^2[\leq, \text{succ}, \sim_k, \pi_{bin}]$

Fix a  $C^2[\leq, \text{succ}, \sim_k, \pi_{bin}]$  formula  $\varphi$  in normal form (1) and its interpretation  $\mathfrak{A}$ . We say that  $\mathfrak{A}$  is  $\varphi$ -rigid if  $\mathfrak{A} \models a \sim_k b$  implies that  $\langle a, b \rangle$  is an invertible ray. We say that  $\varphi$  is rigid if all models of  $\varphi$  are  $\varphi$ -rigid. Define  $\omega_k$  as  $\forall x \exists^{\leq k} y. x \sim_k y$ . Formulae  $\varphi$  and  $\varphi \wedge \omega_k$  are equivalent over  $\mathcal{W}(\leq, \text{succ}, \sim_k, \pi_{bin})$ . Moreover, the latter formula is rigid. We say that  $\mathfrak{A}$  is  $\varphi$ -semichromatic if no ray is emitted and accepted by nodes of the same 1-type. We say that  $\mathfrak{A}$  is  $\varphi$ -superchromatic if it is  $\varphi$ -semichromatic and no element emits two or more rays at least one of which is invertible, having the same absorption-type as each other. We say that  $\varphi$  is  $\varphi$ -semichromatic (resp.  $\varphi$ -superchromatic) if all models of  $\varphi$  are  $\varphi$ -semichromatic (resp.  $\varphi$ -superchromatic). The proof of the following lemma is standard (see [10]).

► **Lemma 12.** *There is a  $C^2$  formula  $\chi_\varphi$  such that  $\varphi$  and  $\varphi \wedge \chi_\varphi$  are equisatisfiable on  $\mathcal{W}(\leq, \text{succ}, \sim_k, \pi_{bin})$  and  $\varphi \wedge \chi_\varphi$  is superchromatic. Moreover, if  $\varphi$  is rigid then  $\varphi \wedge \chi_\varphi$  is so.*

Now we define formulae that encode  $\sim_k$ . Fix a set of star-types  $\mathfrak{st}$ . For  $\sigma, \rho \in \mathfrak{st}$  we write  $\sigma \sim_k \rho$  if there exists an invertible ray type  $\tau$  such that  $\tau \in \sigma$ ,  $\tau^{-1} \in \rho$  and  $\sim_k(x, y) \in \tau$ .

Let  $\mathfrak{A}$  be a rigid  $\mathcal{W}(\leq, \text{succ}, \sim_k, \pi_{bin})$ -structure over  $\mathfrak{st}$ . Structure  $\mathfrak{A}$  consists of disjoint substructures, each generated by an equivalence class of  $\sim_k^{\mathfrak{A}}$ . We call such a substructure a class in  $\mathfrak{A}$ . For a class  $\mathfrak{C}$  in  $\mathfrak{A}$  we call the set  $\{\sigma \in \mathfrak{st} \mid \sigma \text{ is realized in } \mathfrak{C}\}$  the class type of  $\mathfrak{C}$  and denote it by  $\text{ct}(\mathfrak{C})$ . Thus  $\text{ct}(\mathfrak{C})$  is a subset of  $\mathfrak{st}$ . However, not every subset of  $\mathfrak{st}$  corresponds to a class type in a  $\mathcal{W}(\leq, \text{succ}, \sim_k, \pi_{bin})$ -structure. A subset  $\text{ct}$  of  $\mathfrak{st}$  is called a class type wrt.  $\mathfrak{st}$  if there is a bijection  $\mathfrak{b}$  from  $\text{ct}$  to the  $k$ -clique  $K_k = (V, E)$  such that  $(\mathfrak{b}(\sigma), \mathfrak{b}(\rho)) \in E$  if and only if  $\sigma \sim_k \rho$ . Thus, we may identify  $\text{ct}$  with a relational structure, a

## 17:10 A Note on C2 Interpreted over Finite Data-Words

clique, being an equivalence class of  $\sim_k$ . Observe that if  $\mathfrak{C}$  is a class in  $\mathfrak{A}$  then  $\mathfrak{ct}(\mathfrak{C})$  is a class wrt.  $\mathfrak{st}$ . Thus every class wrt.  $\mathfrak{st}$  is potentially a class in some word over  $\mathfrak{st}$ . Since the size of each class type is bounded by  $k$ , the number of class types is bounded by  $\binom{|\mathfrak{st}|}{k}$ . For any  $e \in \mathfrak{C}$  we denote with  $\mathfrak{ct}^{\mathfrak{A}}(e)$  its class type in  $\mathfrak{A}$ , equal to  $\mathfrak{ct}(\mathfrak{C})$ .

Having the above definitions at hand, we may define a two-variable formula  $\psi^{\mathfrak{st}}$  that specifies necessary conditions for  $\sim_k$  to interpret a bounded equivalence relation in a structure that realizes  $\mathfrak{st}$ . Formula  $\psi^{\mathfrak{st}}$  expresses that every node has precisely one class type, that two nodes connected by  $\sim_k$  relation share the same class type, and that a node with a class type  $\mathfrak{c}$  realizes some star-type  $\sigma \in \mathfrak{c}$ . The last property together with  $\varphi$ -semichromaticity implies that star-types of elements within every equivalence class are unique. The entire formula implies that for every node in a structure we may find a set of nodes that together could form an equivalence class. Indeed, we say ‘could’ since it is not necessary the immediate case, and forming equivalence class may require structure transformations.

For  $\varphi$  in normal form (1) by  $\mathfrak{st}(\varphi)$  denote the set of star-types compatible with  $\varphi$ .

► **Lemma 13.** *Any model of a  $C^2[\leq, succ, \sim_k, \pi_{bin}]$  formula  $\varphi$  can be expanded to a model of  $\psi^{\mathfrak{st}(\varphi)}$  by interpreting fresh unary predicates.*

For a fixed  $\varphi$  to be checked for satisfiability, we set  $\varphi_{tr} ::= \varphi \wedge \omega_\varphi \wedge \chi_{\varphi \wedge \omega_\varphi} \wedge \psi^{\mathfrak{st}(\varphi \wedge \omega_\varphi \wedge \chi_{\varphi \wedge \omega_\varphi})}$ .

► **Lemma 14.** *If a  $C^2[\leq, succ, \sim_k, \pi_{bin}]$  formula  $\varphi$  is satisfiable in  $\mathcal{W}(\leq, succ, \sim_k, \pi_{bin})$  then the translation  $\varphi_{tr}$  is satisfiable in  $\mathcal{W}(\leq, succ, \pi_{bin})$ .*

**Proof.** Let  $\mathfrak{A}$  be a model of  $\varphi$  such that  $\mathfrak{A} \in \mathcal{W}(\leq, succ, \sim_k, \pi_{bin})$ . We will expand  $\mathfrak{A}$  by interpreting some fresh unary predicates to obtain a model of  $\varphi_{tr}$ . First, observe that  $\mathfrak{A}$  models  $\omega_\varphi$ , as each equivalence class of  $\sim_k^{\mathfrak{A}}$  has at most  $k$  elements. Using Lemma 12, after interpreting some fresh unary predicates,  $\mathfrak{A}$  becomes a model of  $\chi_{\varphi \wedge \omega_\varphi}$ . Then, using Lemma 13, again by interpreting some fresh unary predicates,  $\mathfrak{A}$  becomes a model of  $\psi^{\mathfrak{st}(\varphi \wedge \omega_\varphi \wedge \chi_{\varphi \wedge \omega_\varphi})}$ . The obtained structure remains in class  $\mathcal{W}(\leq, succ, \sim_k, \pi_{bin})$  and thus also in  $\mathcal{W}(\leq, succ, \pi_{bin})$  and satisfies  $\varphi_{tr}$ . ◀

We now define structure transformations. First, we define a *switch*, whose aim is only to preserve the order of elements. Let us write  $a \ll^{\mathfrak{A}} b$  iff  $a \leq^{\mathfrak{A}} b$  holds and  $succ^{\mathfrak{A}}(a, b)$  does not.

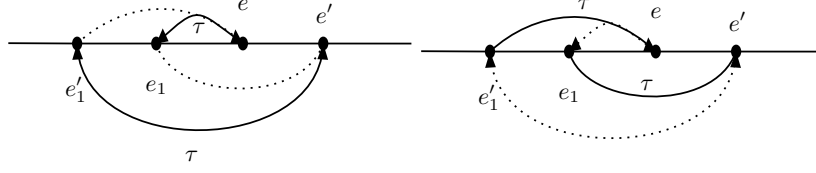
► **Definition 15.** *Let  $\mathfrak{A}$  be a  $\mathcal{W}(\leq, succ, \pi_{bin})$  structure and  $e_1, e, e'$ , be elements of  $A$  such that  $e_1 \ll^{\mathfrak{A}} e$  and  $e \ll^{\mathfrak{A}} e'$ . Define  $(e_1, e, e')$ -switch of  $\mathfrak{A}$  as the structure  $\mathfrak{B}$  which is identical to  $\mathfrak{A}$  with the exception that  $tp^{\mathfrak{B}}(e_1, e) = tp^{\mathfrak{A}}(e_1, e')$  and  $tp^{\mathfrak{B}}(e_1, e') = tp^{\mathfrak{A}}(e_1, e)$ .*

Observe that relative order of  $e_1, e$  and  $e'$  is preserved after the switch and thus both  $succ^{\mathfrak{A}}$  and  $\leq^{\mathfrak{A}}$  are preserved. The transformation we use is a sequence of two switches, as described by the following lemma.

► **Lemma 16 (Switching lemma).** *Let  $\mathfrak{A}$  be a superchromatic  $\mathcal{W}(\leq, succ, \pi_{bin})$  structure,  $e'_1, e_1, e, e'$  be elements of  $A$  such that  $e_1 \ll^{\mathfrak{A}} e$ ,  $e'_1 \ll^{\mathfrak{A}} e$ ,  $e \ll^{\mathfrak{A}} e'$ , and 2-types  $tp^{\mathfrak{A}}(e'_1, e')$  and  $tp^{\mathfrak{A}}(e_1, e)$  are both the same invertible ray type. The structure  $\mathfrak{B}$  obtained by the  $(e_1, e, e')$ -switch of  $\mathfrak{A}$  followed by the  $(e'_1, e, e')$ -switch belongs to  $\mathcal{W}(\leq, succ, \pi_{bin})$  and realizes the same set of star- and 2-types as  $\mathfrak{A}$ .*

**Proof.** Structure  $\mathfrak{A}$  satisfying assumptions of the Lemma is depicted on Fig 1(left). Note that  $tp^{\mathfrak{A}}(e'_1, e)$  and  $tp^{\mathfrak{A}}(e_1, e')$  are silent, as otherwise  $\mathfrak{A}$  would violate the superchromaticity condition. E.g.  $tp^{\mathfrak{A}}(e'_1, e)$  cannot be a ray type, as  $tp^{\mathfrak{A}}(e'_1, e')$  is invertible and  $tp^{\mathfrak{A}}(e') = tp^{\mathfrak{A}}(e)$ . The equality of 1-types hold as a conclusion of  $tp^{\mathfrak{A}}(e'_1, e) = tp^{\mathfrak{A}}(e_1, e')$ . In a similar way

$\text{tp}^{\mathfrak{A}}(e, e'_1)$  cannot be a ray type, thus the  $\text{tp}^{\mathfrak{A}}(e'_1, e)$  is silent. In a similar way  $\text{tp}^{\mathfrak{A}}(e_1, e')$  can be proven silent. After switching we obtain the structure on Fig 1(right), whose star types and 2-types are the same as in  $\mathfrak{A}$ .  $\blacktriangleleft$



■ **Figure 1** Structure  $\mathfrak{A}$  before switching (left) and after switching (right).

The following lemma is the main lemma of this section. There we transform a model of  $\varphi_{tr}$  to another model, where  $\sim_k$  is interpreted as a bounded equivalence relation, and where the order is preserved. We decompose the model into substructures generated by elements connected by  $\text{succ}$  and sharing the same class type (thus any class type also decomposes into components). We show that elements within the same component in the model are necessarily connected by  $\sim_k$  predicate. Then we employ structure transformations defined above (*i.e.* switches) to show that elements of distinct components of the same class type can be pairwise connected by  $\sim_k$  to form equivalence classes.

► **Lemma 17.** *If the formula  $\varphi_{tr}$  is satisfiable in  $\mathcal{W}(\leq, \text{succ}, \pi_{bin})$  then the formula  $\varphi$  is satisfiable in  $\mathcal{W}(\leq, \text{succ}, \sim_k, \pi_{bin})$ .*

**Proof.** Let  $\mathfrak{A}$  be a finite model of  $\varphi_{tr}$  such that  $\mathfrak{A} \in \mathcal{W}(\leq, \text{succ}, \pi_{bin})$ . We will transform  $\mathfrak{A}$  to a  $\mathcal{W}(\leq, \text{succ}, \sim_k, \pi_{bin})$  structure while ensuring that every element of  $\mathfrak{A}$  retains its star-type and the set of realized 2-types is preserved. Since  $\mathfrak{A} \models \varphi$ , by Proposition 11, the obtained structure will still be a model of  $\varphi$ . Observe that  $\varphi_{tr}$  ensures reflexivity and symmetry of  $\sim_k$ . Thus to obtain a  $\mathcal{W}(\leq, \text{succ}, \sim_k, \pi_{bin})$  structure we only need to make  $\sim_k$  transitive. During the transformation the linear order (that is both  $\text{succ}^{\mathfrak{A}}$  and  $\leq^{\mathfrak{A}}$ ) remains fixed, while particular 2-types emitted and accepted by structure nodes may change.

Recall that we may identify each class type  $\mathfrak{c}$  with a relational structure (a clique). By *component* of  $\mathfrak{c}$  we mean any maximal subgraph  $\mathfrak{d}$  of  $\mathfrak{c}$  such that any node of  $\mathfrak{d}$  emits a  $\text{succ}$  edge to some other node of  $\mathfrak{d}$ . Thus graph  $\mathfrak{c}$  consists of (at most  $k$ ) linearly ordered components, each consisting of at most  $k$  elements. Let  $\sigma_1, \dots, \sigma_l$  be all nodes of  $\mathfrak{d}$  listed in order  $\text{succ}$  (all these star-types are distinct as all star-types in any class-type are distinct). Since  $\mathfrak{A} \models \psi^{\text{st}(\varphi \wedge \omega_\varphi \wedge \chi_\varphi \wedge \omega_\varphi)}$ , components of class-types correspond to substructures of  $\mathfrak{A}$  in the following way. If  $e_1 \in \mathfrak{A}$  is such that  $\text{ct}(e_1) = \mathfrak{c}$  and  $\text{st}^{\mathfrak{A}}(e_1) = \sigma_1$  then there exists  $l - 1$  nodes  $e_2, \dots, e_l \in \mathfrak{A}$  such that  $\text{ct}(e_i) = \mathfrak{c}$ ,  $\text{st}^{\mathfrak{A}}(e_i) = \sigma_i$  for  $i \in \{1, \dots, l\}$ , and  $\text{succ}^{\mathfrak{A}}(e_i, e_{i+1})$  for  $i \in \{1, \dots, l - 1\}$ . By definition of  $\mathfrak{d}$  we thus have  $e_i \sim_k^{\mathfrak{A}} e_{i+1}$  for  $i \in \{1, \dots, l - 1\}$ . We call the substructure  $\mathfrak{D}$  of  $\mathfrak{A}$  generated by  $e_1, \dots, e_l$  a *component* of  $\mathfrak{A}$  corresponding to  $\mathfrak{d}$ . We define  $\text{co}(\mathfrak{D}) = \mathfrak{d}$  (the *component-type* of  $\mathfrak{D}$ ) and  $\text{ct}(\mathfrak{D}) = \mathfrak{c}$  (the *class-type* of  $\mathfrak{D}$ ).

We will transform  $\mathfrak{A}$  so to form equivalence classes of  $\sim_k$ . These classes will be  $k$ -cliques composed of components. Thus, we need to ensure that two conditions hold:

- if two nodes belong to the same component then they are connected by  $\sim_k$  edge,
- for every component  $\mathfrak{D}_i$  of  $\mathfrak{A}$  such that all components of  $\text{ct}(\mathfrak{D}_i)$  listed in order are  $\mathfrak{d}_1, \dots, \mathfrak{d}_i, \dots, \mathfrak{d}_l$ , for some numbers  $i$  and  $l$ , we have the following. There exist  $l$  components  $\mathfrak{D}_1, \dots, \mathfrak{D}_i, \dots, \mathfrak{D}_l$  of  $\mathfrak{A}$  such that  $\text{co}(\mathfrak{D}_i) = \mathfrak{d}_i$  and if  $e_i \in \mathfrak{D}_i$  and  $e_j \in \mathfrak{D}_j$  then  $e_i \sim_k^{\mathfrak{A}} e_j$ , for some numbers  $i, j$ .

First, we will show that every two elements of a given component of  $\mathfrak{A}$  are related by  $\sim_k^{\mathfrak{A}}$ , *i.e.* that every component of  $\mathfrak{A}$  is a clique. We will consider components of  $\mathfrak{A}$  in the order defined by  $\leq^{\mathfrak{A}}$ , assuming that all components visited so far satisfy the required property. Let  $\mathfrak{D}$  be a component of  $\mathfrak{A}$  currently under inspection, let  $\mathfrak{c}$  be the class type of  $\mathfrak{D}$  and let  $\mathfrak{d}$  be the component type of  $\mathfrak{D}$ . Take any  $a, b \in \mathfrak{D}$  such that  $a \leq^{\mathfrak{A}} b$ . Let the star-types of  $a, b$  in  $\mathfrak{A}$  be resp.  $\sigma_a$  and  $\sigma_b$ . Ad absurdum, assume that  $a \not\sim_k^{\mathfrak{A}} b$  holds. Since  $a$  and  $b$  belong to the same component  $\mathfrak{D}$ , star-types  $\sigma_a$  and  $\sigma_b$  belong to the component  $\mathfrak{d}$ . Since  $\mathfrak{d}$  is a clique graph, there exists ray-type  $\tau$  such that  $x \leq y \in \tau$ ,  $x \sim_k y \in \tau$ ,  $\tau \in \sigma_a$ , and  $\tau^{-1} \in \sigma_b$ . Since the star-type of  $b$  is  $\sigma_b$ , there exists  $a' \in \mathfrak{A}'$  such that  $\text{tp}^{\mathfrak{A}}(a', b) = \tau$ . Since  $\mathfrak{A} \models \psi^{\text{st}(\varphi \wedge \omega_\varphi \wedge \chi_\varphi \wedge \omega_\varphi)}$ , class-type of  $a'$  is the same as the class-type of  $b$ , *i.e.*  $\text{ct}(a') = \mathfrak{c}$ . Since 1-types within class-types are unique, we have  $\text{st}^{\mathfrak{A}}(a') = \sigma_a$  and  $a'$  belongs to a component  $\mathfrak{D}'$  of  $\mathfrak{A}$  such that the component type of  $\mathfrak{D}'$  is  $\mathfrak{d}$ ,  $\mathfrak{D}' \neq \mathfrak{D}$  and  $\mathfrak{D}'$  occurs in  $\mathfrak{A}$  earlier (wrt.  $\leq^{\mathfrak{A}}$ ) than  $\mathfrak{D}$ . By the inductive assumption all elements of  $\mathfrak{D}'$  are connected by  $\sim_k$ . Since the component type of  $\mathfrak{D}'$  is  $\mathfrak{d}$ , there exists a  $b' \in \mathfrak{D}'$  such that  $\text{st}^{\mathfrak{A}}(b') = \sigma_b$ . Thus  $\text{tp}^{\mathfrak{A}}(a', b') = \tau$ . But, simultaneously  $\text{tp}^{\mathfrak{A}}(a', b) = \tau$ . Because of superchromaticity this can only be true if  $b' = b$ , but these nodes belong to disjoint substructures  $\mathfrak{D}'$  and  $\mathfrak{D}$  of  $\mathfrak{A}$ . Contradiction. Thus  $a \sim_k^{\mathfrak{A}} b$  holds implying, that any two elements of the same component of  $\mathfrak{A}$  are related by  $\sim_k^{\mathfrak{A}}$ .

Now we must switch edges of  $\mathfrak{A}$  so to ensure that elements of distinct components are connected by  $\sim_k$  edges to form equivalence classes of  $\sim_k^{\mathfrak{A}}$ . We traverse components of  $\mathfrak{A}$  in the order defined by  $\text{succ}^{\mathfrak{A}}$  restoring  $\sim_k$  relations between their nodes, when necessary, by employing Lemma 16.  $\blacktriangleleft$

Since the finite satisfiability for  $\text{C}^2[\leq, \text{succ}, \pi_{\text{bin}}]$  is VASS-complete [12], by Lemma 14 and Lemma 17 we immediately conclude:

► **Theorem 18.**  $\text{C}^2[\leq, \text{succ}, \sim_k, \pi_{\text{bin}}]$  is VASS-complete.

## 7 Conclusions

We considered counting extensions of the two-variable logic on finite data-words. While our main logic, namely  $\text{C}^2[\leq, \text{succ}, \sim, \pi_{\text{bin}}]$  turned out to be undecidable, we identified several decidable sub-logics, with complexities ranging from NEXPTIME to VASS, depending on the allowed binary relations in the vocabularies. We hope that the outcome of the paper might be interesting for the two-variable community and that the established decidability results can be later generalised to capture even more expressive forms of quantitative properties.

---

## References

- 1 Rajeev Alur, Pavol Cerný, and Scott Weinstein. Algorithmic analysis of array-accessing programs. *ACM Trans. Comput. Log.* 2012, 2012. doi:10.1145/2287718.2287727.
- 2 Henrik Björklund and Mikolaj Bojańczyk. Shuffle Expressions and Words with Nested Data. In *MFCS 2007*, 2007. A version with an appendix available at <https://www.mimuw.edu.pl/~bojan/upload/confmfcsBjorklundB07.pdf>. doi:10.1007/978-3-540-74456-6\_66.
- 3 Henrik Björklund and Thomas Schwentick. *Class-Memory Automata Revisited*, pages 201–215. Springer, 2017. doi:10.1007/978-3-319-48317-7\_12.
- 4 Mikolaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.* 2011, 2011. doi:10.1145/1970398.1970403.
- 5 Mikolaj Bojańczyk and Sławomir Lasota. An extension of data automata that captures XPath. *LMCS 2012*, 2012. doi:10.2168/LMCS-8(1:5)2012.



- 6 Benedikt Bollig. An Automaton over Data Words That Captures EMSO Logic. In *CONCUR 2011*, 2011. doi:10.1007/978-3-642-23217-6\_12.
- 7 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A robust class of data languages and an application to learning. *LMCS 2014*, 2014. doi:10.2168/LMCS-10(4:19)2014.
- 8 Patricia Bouyer. A logical characterization of data languages. *Inf. Process. Lett.* 2002, 2002. doi:10.1016/S0020-0190(02)00229-6.
- 9 Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.* 2003, 2003. doi:10.1016/S0890-5401(03)00038-5.
- 10 Witold Charatonik, Yegor Guskov, Ian Pratt-Hartmann, and Piotr Witkowski. Two-variable First-Order Logic with Counting in Forests. In *LPAR 2018*, 2018.
- 11 Witold Charatonik, Ian Pratt-Hartmann, and Piotr Witkowski. Two-Variable Logic with Counting and Data-Trees. Submitted. Available at <http://www.cs.man.ac.uk/~ipratt/papers/logic/c21d1e.pdf>.
- 12 Witold Charatonik and Piotr Witkowski. Two-variable Logic with Counting and a Linear Order. *LMCS 2016*, 2016. doi:10.2168/LMCS-12(2:8)2016.
- 13 Taolue Chen, Fu Song, and Zhilin Wu. Formal Reasoning on Infinite Data Values: An Ongoing Quest. In *SETSS 2016, Chongqing, China, March 28 - April 2, 2016*, 2016. doi:10.1007/978-3-319-56841-6\_6.
- 14 Conrad Cotton-Barratt. *Using Class Memory Automata in Algorithmic Game Semantics*. PhD thesis, University of Oxford, UK, 2016.
- 15 Conrad Cotton-Barratt, Andrzej S. Murawski, and C.-H. Luke Ong. Weak and Nested Class Memory Automata. In *LATA 2015*, 2015. doi:10.1007/978-3-319-15579-1\_14.
- 16 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *STOC 2019*, 2019. doi:10.1145/3313276.3316369.
- 17 Loris D’Antoni. In the maze of data languages. *CoRR*, 2012. URL: <http://arxiv.org/abs/1208.5980>.
- 18 Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about Data Repetitions with Counter Systems. *LMCS 2016*, 2016. doi:10.2168/LMCS-12(3:1)2016.
- 19 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.* 2009, 2009. doi:10.1145/1507244.1507246.
- 20 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-Order Logic with Two Variables and Unary Temporal Logic. *Inf. Comput.* 2002, 2002. doi:10.1006/inco.2001.2953.
- 21 Diego Figueira. A Decidable Two-Way Logic on Data Words. In *LICS 2011*, 2011. doi:10.1109/LICS.2011.18.
- 22 Daniel Genkin, Michael Kaminski, and Liat Peterfreund. A note on the emptiness problem for alternating finite-memory automata. *Theor. Comput. Sci.* 2014, 2014. doi:10.1016/j.tcs.2014.01.020.
- 23 Erich Grädel and Martin Otto. On Logics with Two Variables. *Theor. Comput. Sci.* 1999, 1999. doi:10.1016/S0304-3975(98)00308-9.
- 24 Ahmet Kara. *Logics on data words: Expressivity, satisfiability, model checking*. PhD thesis, Technical University of Dortmund, Germany, 2016. URL: <http://hdl.handle.net/2003/35216>.
- 25 Jérôme Leroux and Sylvain Schmitz. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *LICS 2019*, 2019. doi:10.1109/LICS.2019.8785796.
- 26 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.
- 27 Angelo Montanari, Marco Pazzaglia, and Pietro Sala. Metric propositional neighborhood logic with an equivalence relation. *Acta Inf.* 2016, 2016. doi:10.1007/s00236-016-0256-3.
- 28 Matthias Niewerth. *Data definition languages for XML repository management systems*. PhD thesis, Technical University of Dortmund, Germany, 2016.

## 17:14 A Note on C2 Interpreted over Finite Data-Words

- 29 Ian Pratt-Hartmann. The Two-Variable Fragment with Counting Revisited. In *WoLLIC 2010*, 2010. doi:10.1007/978-3-642-13824-9\_4.
- 30 Ian Pratt-Hartmann. The two-variable fragment with counting and equivalence. *Math. Log. Q.* 2015, 2015. doi:10.1002/malq.201400102.
- 31 Sebastian Rudolph. Presburger Concept Cardinality Constraints in Very Expressive Description Logics - Allegro sexagenarioso ma non ritardando. In *Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, 2019. doi:10.1007/978-3-030-22102-7\_25.
- 32 Fu Song and Zhilin Wu. On temporal logics with data variable quantifications: Decidability and complexity. *Inf. Comput.* 2016, 2016. doi:10.1016/j.ic.2016.08.002.



# Stab-Forests: Dynamic Data Structures for Efficient Temporal Query Processing

Jelle Hellings

Exploratory Systems Lab, Department of Computer Science,  
University of California, Davis, CA, USA  
jhellings@ucdavis.edu

Yuqing Wu

Computer Science Department, Pomona College, Claremont, CA, USA  
melanie.wu@pomona.edu

---

## Abstract

Many sources of data have temporal start and end attributes or are created in a time-ordered manner. Hence, it is only natural to consider joining datasets based on these temporal attributes. To do so efficiently, several internal-memory temporal join algorithms have recently been proposed. Unfortunately, these join algorithms are designed to join entire datasets and cannot efficiently join skewed datasets in which only few events participate in the join result.

To support high-performance internal-memory temporal joins of skewed datasets, we propose the *skip-join algorithm*, which operates on *stab-forests*. The *stab-forest* is a novel dynamic data structure for indexing temporal data that allows efficient updates when events are appended in a time-based order. Our *stab-forests* efficiently support not only traditional temporal *stab-queries*, but also more general *multi-stab-queries*. We conducted an experimental evaluation to compare the *skip-join* algorithm with state-of-the-art techniques using real-world datasets. We observed that the *skip-join* algorithm outperforms other techniques by an order of magnitude when joining skewed datasets and delivers comparable performance to other techniques on non-skewed datasets.

**2012 ACM Subject Classification** Information systems → Join algorithms; Information systems → Temporal data

**Keywords and phrases** Cache-friendly temporal joins, temporal data, skewed data, *stab-queries*, temporal indices

**Digital Object Identifier** 10.4230/LIPICs.TIME.2020.18

**Supplementary Material** Open-source code of the full implementation of the data structures, algorithms, and supporting tooling used can be found at <https://jhellings.nl/projects/skipjoin/>.

**Funding** This material is based upon work supported by the National Science Foundation under Grant No. NSF 1606557.

## 1 Introduction

In practice, most sources of data have temporal attributes. Examples include news events, air travel records, employment records, and event logs. Temporal attributes also play a role in data that does not have explicit temporal attributes: e.g., in versioned databases the time of creation and replacement of each data element is recorded such that the evolution of the database is maintained. Given that temporal data is so ubiquitous, applications naturally expect support from DBMSs for efficient operations based on these temporal attributes. Examples of such operations are *stab-queries* and the *temporal join*:

► **Example 1.1.** Consider complex systems in which events are logged by (start, end)-time intervals. We want to use the event log to diagnose failures in the complex system. More specifically, if a failure at time  $t$  needs to be diagnosed, one does not want to search through



© Jelle Hellings and Yuqing Wu;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 18; pp. 18:1–18:19

Leibniz International Proceedings in Informatics



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

the entire event log, but use more directed ways to look for causes. A first step would be to perform a *stab-query* at time  $t$  to find all events that are active when the failure happened. A next step would be to combine event logs of the failed system with event logs of another system via a (*windowed*) *temporal join* that yields pairs of events that were active at the same time (and could have influenced each other) in a 24 h-window around time  $t$ .

Efficient temporal join algorithms are at the basis of many other efficient temporal operations. E.g., selecting all events in a dataset that occur during a given set of windows in time is equivalent to a temporal join between the dataset and these windows. Unfortunately, state-of-the-art techniques fail to cope with skewed datasets or fail to deliver high performance:

**Temporal join algorithms.** Many temporal join algorithms proposed in the literature fine-tune the usage of traditional relational database storage and join techniques towards temporal joining [3, 7, 13, 15, 26]. These relational-oriented temporal join algorithms are not optimized for *high-performance internal-memory operations* and achieve only acceptable performance.

Separately from these relational-oriented approaches, a few dedicated internal-memory join algorithms have been proposed that operate on ordered arrays of events. These algorithms use merge-join style methods that employ either sweeping-based techniques or forward-scan techniques [3, 6, 7, 13, 15, 21, 26]. Based on these merge-join style algorithms, Piatov et al. [21] recently introduced the endpoint join algorithm, a cache-friendly internal-memory temporal join algorithm. Due to the timestamp-based representation of events used by the endpoint join algorithm, the algorithm needs complicated data structures to maintain active lists of events while joining them. Bourus et al. [6] showed that a traditional forward-scan algorithm [7] operating on a simple event list will attain similar performance without all the complexities of the endpoint join algorithm. Unfortunately, these merge-join style algorithms inspect the entire dataset, due to which their performance degrades when the join output is restricted to a small window in time and when the datasets only have *few overlapping events*, the latter limiting their usability on skewed datasets.

**Temporal data structures.** Besides relying on temporal joins, one can also consider index structures that support answering temporal operations. Unfortunately, existing index structures either do not support temporal operations efficiently, are statically built or complex to maintain, or fail to provide high-performance cache-friendly internal-memory operations. Indeed, to support temporal operations over interval data, traditional relational indices such as binary search trees, B-trees, and range-trees *cannot be effectively used*, as these structures lack the information to efficiently perform stab-queries and other basic temporal operations [5, 13, 22, 26]. Alternatively, one can use specialized static interval data structures developed for geometric applications [1, 4, 5, 16, 18, 23]. Examples include interval trees, segment trees, and priority search trees [11, 12, 17]. These statically built data structures all support efficient stab-queries, but *do not support any form of updates*.

Unfortunately, dynamic general-purpose versions of these statically-built interval data structures are highly complex, have expensive maintenance algorithms, and rely completely on pointer-based tree structures [4, 9, 20]. The usage of such complex pointer-based data structure prevents cache-friendly traversal and, hence, prevent them from supporting *high-performance internal-memory operations* [4, 14, 24]. A few external memory interval data structures have been proposed, but these either place many restrictions on the inserted data [19, 25] or are highly complex and have not yet proven themselves in practice [4].

**Our proposal: the skip-join algorithm.** To address the shortcomings of existing techniques, we propose the *skip-join algorithm* (Section 2). Our skip-join algorithm is an efficient temporal join algorithm that can deal with all datasets and, additionally, supports windowed temporal joins. To do so, the skip-join algorithm uses the *stab-forest*, a novel temporal index data structure that is designed to efficiently support *stab-queries* and, more importantly, *multi-stab-queries* that yield the combined results of multiple stab-queries in a highly efficient manner (Section 3 and Section 4). We also present efficient ways to maintain stab-forests when events are appended (Section 5).

To show the effectiveness of the skip-join algorithm, we evaluate the performance of our algorithm using real-world datasets (Section 6). Our evaluation shows that the skip-join algorithm outperforms state-of-the-art join algorithms by an order of magnitude when joining skewed datasets. On dense datasets, the performance of the skip-join algorithm is comparable to the state-of-the-art.

## 2 The Skip-Join Algorithm

Before we propose the skip-join algorithm, we first introduce some event-related terminology. A *timestamp* represents a single point in time. We assume that *timestamps* are non-negative integers. An *event* is a pair  $\langle v, w \rangle$  of timestamps that represents the interval  $[v, w]$  in time. If  $E = \langle v, w \rangle$  is an event, then  $v$  is the *start-time* and  $w$  is the *end-time* and we write  $E.start$  and  $E.end$  to denote  $v$  and  $w$ , respectively. If  $E.start \leq t \leq E.end$ , then we also say that  $E$  is *active* at  $t$ . If  $E_1$  and  $E_2$  are events, then the *intersection*  $E_1 \cap E_2$  is empty if  $E_1.end < E_2.start$  or  $E_2.end < E_1.start$ . We say that  $E_1$  and  $E_2$  *overlap* if  $E_1 \cap E_2 \neq \emptyset$ .

► **Definition 2.1.** Let  $R$  and  $S$  be sets of events. The temporal join of  $R$  and  $S$ , denoted by  $R \bowtie S$ , is defined by

$$R \bowtie S = \{(E_1, E_2) \in R \times S \mid E_1 \cap E_2 \neq \emptyset\}.$$

Our skip-join algorithm relies on the ability to efficiently perform sequences of stab-queries:

► **Definition 2.2.** Let  $S$  be a set of events, let  $t$  be a timestamp, and let  $\phi$  be a sorted sequence of timestamps. The *stab-query* of  $S$  by  $t$  is defined by

$$STAB(S, t) = \{E \in S \mid E.start \leq t \leq E.end\},$$

and the *multi-stab-query* of  $S$  by  $\phi$  is defined by

$$MULTISTAB(S, \phi) = \bigcup_{t \in \phi} STAB(S, t).$$

Several high-performance internal-memory temporal join algorithms have been proposed, most of which use a merge-join style method that employs either sweeping-based techniques or forward-scan techniques [3, 6, 7, 13, 15, 21, 26]. Unfortunately, these merge-join style algorithms cannot efficiently support windowed temporal joins or deal with skewed datasets.

To efficiently support windowed temporal joins and deal with skewed datasets, we will present our skip-join algorithm. To simplify presentation, we build skip-join on top of the forward-scan algorithm, the simplest among the merge-join style algorithms. Our techniques can, however, easily be translated to endpoint-based join algorithms, e.g., the algorithm of Piatov et al. [21].

■ **Algorithm 1** Algorithm FWDSKAN, outputs  $R \bowtie S$  ( $R$  and  $S$  sorted in ascending start-time order).

---

**Algorithm** FWDSKAN( $R, S$ ):

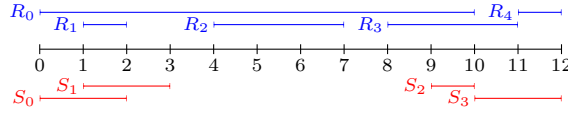
---

```

1:  $i, j := 0, 0$ 
2: while  $i < |R|$  and  $j < |S|$  do
3:   if  $R[i].\text{start} \leq S[j].\text{start}$  then
4:      $k := j$  #(Join  $R[i]$  with  $S[j \dots]$ ).
5:     while  $k < |S|$  and  $S[k].\text{start} \leq R[i].\text{end}$  do
6:       Output  $(R[i], S[k])$ 
7:        $k := k + 1$ 
8:      $i := i + 1$ 
9:   else analogous (swap roles of  $R$  and  $S$ )

```

---



■ **Figure 1** Two lists of events  $R$  and  $S$  visualized on an explicit timestamp scale.

**The forward-scan temporal join algorithm.** The forward-scan algorithm is a cache-friendly temporal join algorithm that can efficiently join non-skewed datasets represented by ordered lists of events (e.g., sorted arrays of events) [3, 6, 7, 13, 15, 21, 26]. The outline of such a forward-scan algorithm is shown in Algorithm 1.

► **Example 2.3.** Let  $R = [\langle 0, 10 \rangle, \langle 1, 2 \rangle, \langle 4, 7 \rangle, \langle 8, 11 \rangle, \langle 11, 12 \rangle]$  and  $S = [\langle 0, 2 \rangle, \langle 1, 3 \rangle, \langle 9, 10 \rangle, \langle 10, 12 \rangle]$  be the lists of events visualized in Figure 1. We compute  $R \bowtie S$  using Algorithm FWDSKAN. First, we join  $R_0$  with  $S[0 \dots]$ , and output  $(R_0, S_0)$ ,  $(R_0, S_1)$ ,  $(R_0, S_2)$ . Next, we join  $S_0$  with  $R[1 \dots]$ , and output  $(R_1, S_0)$ . Next, we join  $R_1$  with  $S[1 \dots]$ , and output  $(R_1, S_1)$ . Next we join  $S_1$  with  $R[2 \dots]$ , and output nothing. Next, we join  $R_2$  with  $S[2 \dots]$ , and output nothing. Next, we join  $R_3$  with  $S[2 \dots]$ , and output  $(R_3, S_2)$ ,  $(R_3, S_3)$ . Next, we join  $S_2$  with  $R[4 \dots]$ , and output nothing. Next, we join  $S_3$  with  $R[4 \dots]$ , and output  $(R_4, S_3)$ . Finally, we stop, as we have reached the end of  $S$ .

If the event-list is implemented as an array, then these forward-scan algorithms will have high performance when most events in  $R$  and  $S$  are part of the join result:

► **Proposition 2.4.** *Let  $R$  and  $S$  be lists of events sorted in ascending start-time order. The algorithm FWDSKAN( $R, S$ ) computes  $R \bowtie S$  in worst-case  $\mathcal{O}(|R| + |S| + |\text{output}|)$ .*

**Dealing with skew in temporal joins.** In practice, one can expect some skew in the data that causes standard forward-scan algorithm to waste time inspecting parts of  $R$  and/or  $S$  that are not part of the join result. To deal with this form of data skew, we need a way to detect and skip over parts of  $R$  and  $S$  that are irrelevant to the join result. To do so, we augment the forward-scan algorithm with the ability to use stab-queries to jump over irrelevant events: if, e.g., we are at an event  $R[i]$  that ends before the event  $S[j]$  starts, then we simply jump in  $R$  until we find the first event  $R[i']$  that starts after  $S[j]$ . By jumping over events in  $R$ , we might miss events in  $R[i \dots i']$  that end after  $S[j].\text{start}$ . To assure we do not miss such events, we jump over events in  $R$  via a stab-query and join the output of the stab-query with  $S[j \dots]$ . This approach results in the skip-join algorithm of Algorithm 2.

► **Example 2.5.** Consider the lists of events  $R$  and  $S$  of Example 2.3 and visualized in Figure 1. We compute  $R \bowtie S$  using the SKIPJOIN algorithm. First, we join  $R_0$  with  $S[0 \dots]$ ,

■ **Algorithm 2** Algorithm SKIPJOIN, outputs  $R \bowtie S$  ( $R$  and  $S$  sorted in ascending start-time order).

---

**Algorithm** SKIPJOIN( $R, S$ ):

---

```

1:  $i, j := 0, 0$ 
2: while  $i < |R|$  and  $j < |S|$  do
3:   if  $R[i].\text{start} \leq S[j].\text{start}$  then
4:     if  $S[j].\text{start} \leq R[i].\text{end}$  then
5:       Join  $R[i]$  with  $S[j \dots]$  (see Algorithm 1, Lines 4–8)
6:        $i := i + 1$ 
7:     else
8:        $(i, L) := \text{STAB}(R[i \dots], R[i].\text{start})$ 
9:       For each event  $E \in L$ , join  $E$  with  $S[j \dots]$  (see Algorithm 1, Lines 4–8)
10:    else analogous (swap roles of  $R$  and  $S$ )

```

---

and output  $(R_0, S_0)$ ,  $(R_0, S_1)$ ,  $(R_0, S_2)$ . Next, we join  $S_0$  with  $R[1 \dots]$ , and output  $(R_1, S_0)$ . Next, we join  $R_1$  with  $S[1 \dots]$ , and output  $(R_1, S_1)$ . Next we join  $S_1$  with  $R[2 \dots]$ , and output nothing. Next, when we process the event  $R_2 = \langle 4, 7 \rangle$ , we detect that the event  $S_2 = \langle 9, 10 \rangle$ , the first event in  $S[2 \dots]$ , starts after  $R_2.\text{end}$  as  $7 = R_2.\text{end} < S_3.\text{start} = 9$ . Hence, we perform  $\text{STAB}(R[2 \dots], 9)$ , which yields the list  $[R_3]$  and the index 4 in  $R$ . We output  $(R_3, S_2)$  and continue with joining  $R[4 \dots]$  and  $S[2 \dots]$ , which only yields  $(R_4, S_3)$ .

The SKIPJOIN algorithm will only be efficient if the sequence of stab-queries can be performed efficiently. Obviously, such an ordered sequence of stab-queries can be seen as a single multi-stab-query (whose evaluation is interleaved with running the join algorithm). In Section 3, we introduce the stab-forest data structure which we will use to answer such multi-stab-queries efficiently, and in Section 4, we show how to efficiently query stab-forests.

► **Theorem 2.6.** *Let  $R$  and  $S$  be lists of events sorted in ascending start-time order. The algorithm SKIPJOIN( $R, S$ ) computes  $R \bowtie S$  in worst-case  $\mathcal{O}(M(R, S) + M(R, S) + |\text{output}|)$ , in which  $M(A, B)$  denotes the cost of either a multi-stab-query with  $|A|$  timestamps on  $B$  or of fully traversing  $B$ , whichever is smaller.*

We notice that the focus of the skip-join algorithm is on supporting temporal joins of skewed datasets. The skip-join algorithm can easily be tuned to also support *windowed* temporal joins that only output events restricted to some window  $\langle v, w \rangle$  in time, however: one simply starts with stab-queries to determine which events are active at  $v$  and stops whenever encountering events that start after  $w$ .

### 3 The Stab-Forest Data Structure

In the previous section, we introduced the SKIPJOIN algorithm. This algorithm requires an efficient manner to perform multi-stab-queries. To provide this, we introduce a novel index structure, the *stab-forest*. The stab-forest is a triple  $\mathcal{S} = (\mathcal{E}, \mathcal{I}, i_{\text{tail}})$  with  $\mathcal{E}$  an *event-list* ordered lexicographically on (start, end)-times,  $\mathcal{I}$  an *index* over the head of the event-list, and  $i_{\text{tail}}$  the *tail pointer* that holds the offset of the first event in  $\mathcal{E}$  not yet part of  $\mathcal{I}$ . We call the part of the event-list starting at  $i_{\text{tail}}$  the *tail*. We define  $|\mathcal{S}| = |\mathcal{E}|$ . Next, we introduce stab-trees, which are at the basis of index  $\mathcal{I}$ . Then, we introduce the forest structure of  $\mathcal{I}$ , which stitches together the stab-trees used to index the event-list. Finally, we discuss the relevant parts of the physical layout we use for the index.

**The stab-tree.** A stab-tree is a binary tree that shares similarities with binary search trees and interval trees. First, we introduce the standard binary tree terminology and

notation. Let  $\mathcal{S} = (\mathcal{E}, \mathcal{I}, i_{\text{tail}})$  be a stab-forest and let  $T$  be a stab-tree indexing a portion of  $\mathcal{E}$ . We write  $\text{root}(T)$  to denote the *root node* of  $T$ . Let  $n$  be a node in  $T$ . By  $\text{left}(n)$  and  $\text{right}(n)$ , we denote the *left* and *right* child of  $n$ . We call  $n$  a *leaf* if  $n$  does not have children ( $\text{left}(n) = \text{right}(n) = \perp$ ). By  $\text{height}(T)$ , we denote the *height* of the tree  $T$ , which we define as the number of nodes on the longest downward path from the root of the tree to a leaf node (the height of a tree without nodes is 0 and the height of a tree with a single node is 1).

Each node  $n$  has a *navigation key* and a *data key*, denoted by  $\text{nkey}(n)$  and  $\text{dkey}(n)$ , respectively. The key  $\text{dkey}(n)$  is a timestamp present as the start-time of an event in the event-list. The *data pointer*  $i_{\text{data}}(n)$  holds the offset of the first event in  $\mathcal{E}$  with start-time  $\text{dkey}(n)$ . The key  $\text{nkey}(n)$  is the smallest timestamp such that no event in the event-list has a start-time in the range  $[\text{nkey}(n), \text{dkey}(n))$ .

Each node of a stab-tree represents events in  $\mathcal{E}$  via the data key and data pointer: the node  $n$  represents those events  $E \in \mathcal{E}$  with  $E.\text{start} = \text{dkey}(n)$ . The navigation key  $\text{nkey}(n)$  is derived from the event preceding  $\mathcal{E}[i_{\text{data}}(n)]$ . Based on the definition of  $\text{nkey}(n)$ , the only timestamp in  $[\text{nkey}(n), \text{dkey}(n)]$  that has events  $E \in \mathcal{E}$  starting at it is  $\text{dkey}(n)$  – these are exactly the events represented by  $n$ . In Section 4, we will explain how the data and navigation keys are used while querying stab-forests. We define

$$\begin{aligned} \min(n) &= \min\{\text{nkey}(n') \mid n' \text{ in the subtree rooted at } n\}; \\ \max(n) &= \max\{\text{dkey}(n') \mid n' \text{ in the subtree rooted at } n\}. \end{aligned}$$

The *scope* of  $n$  is defined by  $\text{scope}(n) = [\text{nkey}(n), \text{dkey}(n)]$  and the *cover* by  $\text{cover}(n) = [\min(n), \max(n)]$ . We say that timestamp  $t$  is in the *scope* of  $n$  if  $t \in \text{scope}(n)$  and is *covered* by  $n$  if  $t \in \text{cover}(n)$ . For answering stab-queries, each node  $n$  is augmented with a *left-list*

$$\text{LEFT}(n) = \{\langle v, w \rangle \in \mathcal{E} \mid \min(n) \leq v \leq \text{dkey}(n) \wedge \text{nkey}(n) \leq w \leq \max(n)\}.$$

Intuitively, the left-list  $\text{LEFT}(n)$  contains all events that are active in the *scope* of  $n$ , while starting and ending in the cover of  $n$ .

► **Example 3.1.** Consider the list of events  $[\langle 0, 3 \rangle, \langle 0, 11 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 4, 5 \rangle, \langle 5, 5 \rangle, \langle 5, 6 \rangle, \langle 6, 8 \rangle, \langle 7, 7 \rangle, \langle 7, 9 \rangle]$ . This list is indexed by the stab-tree  $T$  visualized in Figure 2, *left*. We have  $\text{height}(T) = 3$ . For the root node  $r = \text{root}(T)$ , we have  $\text{nkey}(r) = 3$ ,  $\text{dkey}(r) = 4$ ,  $\min(r) = 0$ ,  $\max(r) = 7$ ,  $\text{scope}(r) = \langle 3, 4 \rangle$ ,  $\text{cover}(r) = \langle 0, 7 \rangle$ , and  $\text{LEFT}(r) = [\langle 0, 3 \rangle, \langle 2, 3 \rangle, \langle 4, 5 \rangle]$ .

Every node  $n$  in a stab-tree in  $\mathcal{I}$  must satisfy the following four structural invariants:

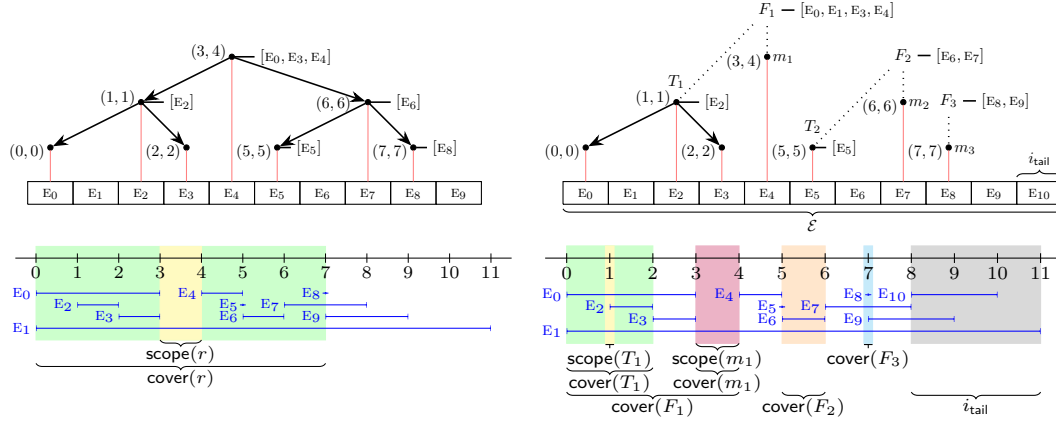
- (i)  $\text{nkey}(n) \leq \text{dkey}(n)$ ;
- (ii) if  $E \in \mathcal{E}$  with  $E.\text{start} \in \text{scope}(n)$ , then  $E.\text{start} = \text{dkey}(n)$ ;
- (iii) if  $\text{left}(n) \neq \perp$ , then  $\text{nkey}(n) = 1 + \max(\text{left}(n))$ ; and
- (iv) if  $\text{right}(n) \neq \perp$ , then  $\text{dkey}(n) = \min(\text{right}(n)) - 1$ .

To use the stab-trees for answering stab-queries efficiently, we also need to provide strong upper-bounds on the height of stab-trees. To do so, we put the following structural invariant on each stab-tree  $T$  used in  $\mathcal{I}$ :

- (v)  $T$  has exactly  $2^{\text{height}(T)} - 1$  nodes.

Invariants i-iv imply the binary-search-tree property and Invariant v implies that each stab-tree is balanced and complete.

Let  $t$  be a timestamp. We say that a stab-tree  $T$  *covers*  $t$  if  $\text{root}(T)$  covers  $t$ . We say that an event  $E \in \mathcal{E}$  is *covered* by a stab-tree node or stab-tree if  $E.\text{start}$  is covered by it. Given a stab-tree  $T$  and a timestamp  $t$  covered by  $T$ , Invariants i-iv guarantee that there exists exactly one node  $n$  in  $T$  that has  $t$  in its scope. Likewise, if  $E \in \mathcal{E}$  is covered by  $T$ , then there



**Figure 2** Examples of stab-trees and stab-forests. *Left*, a stab-tree indexing ten events. For each node  $n$ , the keys are visualized as  $(nkey(n), dkey(n))$  and the left-list is only included if  $LEFT(n) \neq \emptyset$ . *Right*, three forest-points over the same dataset, each with its own stab-tree, dummy stab-tree node, and max-list. Observe that the last forest-point’s stab-tree is empty.

exists exactly one node  $n$  in  $T$  with  $dkey(n) = E.start$ . Given this node  $n$ ,  $i_{data}(n)$  holds the offset of the first event in  $\mathcal{E}$  with start-time  $E.start$ . In this case, either  $E$  is part of exactly one left-list of a node  $n'$ , ancestor of  $n$ , in  $T$  or  $E.end > \max(\text{root}(T))$ .

► **Example 3.2.** Consider the stab-tree  $T$  in Example 3.1, visualized in Figure 2, *left*. The timestamps  $0, \dots, 7$  are covered by  $T$ . More specifically, the timestamp 3 is covered by  $\text{root}(T)$ , even though no event starts at 3. the timestamp 5 is covered by the leaf node with data key 5. No timestamp at-or-after 8 is covered by  $T$ , even though some events covered by  $T$  end at-or-after timestamp 8.

**The stab-forest index.** We require that all stab-trees are balanced and complete. Consequently, it is in most cases impossible to cover all events by a single stab-tree. Alternatively, we can cover consecutive parts of the event-list by a forest of stab-trees of decreasing heights. To use this forest of stab-trees for query answering, we need to maintain some metadata per stab-tree. This metadata is stored in *forest-points*.

Let  $\mathcal{S} = (\mathcal{E}, \mathcal{I}, i_{tail})$  be a stab-forest. A *forest-point* in  $\mathcal{I}$  is a pair  $F = (T, m)$ , with  $T$  a stab-tree and  $m$  a dummy stab-tree node without left-list augmentation. We define  $\text{height}(F) = 1 + \text{height}(T)$ ,

$$\min(F) = \begin{cases} nkey(m) & \text{if } \text{height}(T) = 0; \\ \min(\text{root}(T)) & \text{if } \text{height}(T) \neq 0, \end{cases}$$

and  $\max(F) = dkey(m)$ . Each forest-point  $F$  is augmented with a *max-list*

$$\text{MAX}(F) = \{\langle v, w \rangle \in \mathcal{E} \mid \min(m) \leq v \leq dkey(m) \wedge nkey(m) \leq w\}.$$

If we interpret  $\text{root}(T)$  as the left child of  $m$ , then, intuitively, the max-list  $\text{MAX}(F)$  contains all events that are active in the scope of  $m$  while starting in the cover of  $m$  (but not necessary ending in the cover of  $m$ ). Hence, conceptually, forest-points and their max-lists can be seen as open-ended versions of stab-tree nodes and their left-lists:  $\text{MAX}(m)$  is a superset of a conceptual left-list of  $m$  with left child  $\text{root}(T)$  and with a yet undetermined right child. If a



sufficient number of events is appended to the event-list, then one is capable of constructing an appropriate right child for  $m$ , after which  $\text{MAX}(F)$  provides all the candidate events that might be part of  $\text{LEFT}(m)$ .

► **Example 3.3.** Consider the event-list  $\mathcal{E} = [\langle 0, 3 \rangle, \langle 0, 11 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 4, 5 \rangle, \langle 5, 5 \rangle, \langle 5, 6 \rangle, \langle 6, 8 \rangle, \langle 7, 7 \rangle, \langle 7, 9 \rangle, \langle 8, 10 \rangle]$ . This list is indexed by the stab-forest  $\mathcal{S}$  visualized in Figure 2, *right*. The stab-forest  $\mathcal{S}$  has three forest-points,  $F_1 = (T_1, m_1)$ ,  $F_2 = (T_2, m_2)$ , and  $F_3 = (T_3, m_3)$ . Let  $r_1 = \text{root}(T_1)$ . For the first forest-point, we have  $\text{nkey}(r_1) = \text{dkey}(r_1) = 1$ ,  $\text{nkey}(m_1) = 3$ ,  $\text{dkey}(m_1) = 4$ , and  $\text{MAX}(F_1) = [\langle 0, 3 \rangle, \langle 0, 11 \rangle, \langle 2, 3 \rangle, \langle 4, 5 \rangle]$ . For the third forest-point, we observe that stab-tree  $T_3$  is empty. Not all events are part of the index, as the tail-pointer points to the last event  $E_{10} = \langle 8, 10 \rangle$ . We observe that these forest-points cover the same set of events as the single stab-tree of Example 3.1. Due to the structural invariants we will place on stab-forests, the provided set of events does not yet contain sufficient information to merge these three forest-points to the stab-tree of Example 3.1, however.

We say that a forest-point *covers* timestamp  $t$  if either  $T$  or  $m$  covers  $t$ , we say that index  $\mathcal{I}$  *covers* timestamp  $t$  if a forest-point  $F \in \mathcal{I}$  covers  $t$ , and we say that the tail *covers*  $t$  if  $t$  is larger than any timestamp covered by  $\mathcal{I}$ . We say that an event  $E \in \mathcal{E}$  is covered by a forest-point, index, or tail if  $E.\text{start}$  is covered by it. We write  $\text{events}(F)$ ,  $\text{events}(T)$ ,  $\text{events}(n)$ , and  $\text{events}(i_{\text{tail}})$  to denote the set of events in the  $\mathcal{E}$  covered by forest-point  $F$ , stab-tree  $T$ , stab-tree node  $n$ , or the tail, respectively.

To guarantee that  $\mathcal{I}$  covers all events in  $\mathcal{E}$  up to  $i_{\text{tail}}$  and that the index structure has strong upper-bounds on its size, we put the following structural invariants on the index:

- (vi) the first forest-point in  $\mathcal{I}$  covers the first event in  $\mathcal{E}$ ;
- (vii) all events in  $\mathcal{E}$  at-or-after  $i_{\text{tail}}$  have the same start-time;
- (viii) if  $\mathcal{E} \neq \emptyset$ , then  $i_{\text{tail}}$  is the offset of the first event  $E \in \mathcal{E}$  not covered by  $\mathcal{I}$ ;
- (ix) if  $(T, m) \in \mathcal{I}$  with  $\text{height}(T) \neq 0$ , then  $\text{nkey}(m) = 1 + \max(\text{root}(T))$ ; and
- (x) if  $F_2 \in \mathcal{I}$  directly follows  $F_1 \in \mathcal{I}$ , then  $\text{height}(F_1) > \text{height}(F_2)$  and  $\min(F_2) = 1 + \max(F_1)$ .

We observe that the Invariants vi–x combined with Invariants i–iv guarantee that, for every event  $E \in \mathcal{E}$  before offset  $i_{\text{tail}}$ , there exists exactly one forest-point  $F \in \mathcal{I}$  that covers  $E$ . If  $E \in \mathcal{E}$  and  $E$  is covered by  $F = (T, m)$ , then either  $E$  is part of exactly one left-list of a node in  $T$  or  $E \in \text{MAX}(F)$ . Combining Invariant v with Invariant x allows us to upper bound the number and height of forest-points: if  $\mathcal{E}$  has  $N$  distinct start-times and  $F \in \mathcal{I}$  is the  $i$ -th forest-point in  $\mathcal{I}$ ,  $0 \leq i < |\mathcal{I}|$ , then  $|\mathcal{I}| \leq \lceil \log N \rceil$  and  $\text{height}(F) \leq \lceil \log N \rceil - (i + 1)$ .

**Physical representation.** The index structure will be used to support multi-stab-queries. To do so efficiently, we use specialized materializations of the left-lists and max-lists. Let  $n$  be a stab-tree node and let  $F = (T, m)$  be a forest-point. The lists  $\text{LEFT}(n)$  and  $\text{MAX}(F)$  are each stored in two parts:

$$\begin{aligned} \text{LEFT}_{n,\downarrow}(n) &= \{\langle v, w \rangle \in \text{LEFT}(n) \mid v < \text{nkey}(n)\}; & \text{LEFT}_{d,\downarrow}(n) &= \text{LEFT}(n) \setminus \text{LEFT}_{n,\downarrow}(n); \\ \text{MAX}_{n,\downarrow}(F) &= \{\langle v, w \rangle \in \text{MAX}(F) \mid v < \text{nkey}(m)\}; & \text{MAX}_{d,\downarrow}(F) &= \text{MAX}(F) \setminus \text{MAX}_{n,\downarrow}(F). \end{aligned}$$

In the above, each part is sorted on descending end-time order. We also maintain copies  $\text{LEFT}_{n,\uparrow}(n)$  and  $\text{MAX}_{n,\uparrow}(F)$  of  $\text{LEFT}_{n,\downarrow}(n)$  and  $\text{MAX}_{n,\downarrow}(F)$  that are sorted on ascending start-time order.

► **Proposition 3.4.** *Let  $L$  be a list of events. The stab-forest  $\mathcal{S}$  indexing  $L$  can be stored in worst-case  $\mathcal{O}(|L|)$  space.*



## 4 Query Evaluation on Stab-Forests

Previously, we discussed the structure of the stab-forest. Next, we show how the stab-forest supports answering multi-stab-queries efficiently. The definition of multi-stab-queries suggests a straightforward way to answer them: by simply executing multiple stab-queries and combining the results. This approach can become unnecessary inefficient if the dataset has events that appear in several of these stab-queries, as we have to explicitly eliminate duplicates.

► **Example 4.1.** Consider the stab-forest  $\mathcal{S}$  of Example 3.3. We consider the multi-stab-query  $\text{MULTISTAB}(\mathcal{S}, [0, 2, 5])$ . We have  $\text{STAB}(\mathcal{S}, 0) = \{\langle 0, 3 \rangle, \langle 0, 11 \rangle\}$ ,  $\text{STAB}(\mathcal{S}, 2) = \{\langle 0, 3 \rangle, \langle 0, 11 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle\}$ , and  $\text{STAB}(\mathcal{S}, 5) = \{\langle 0, 11 \rangle, \langle 4, 5 \rangle, \langle 5, 5 \rangle\}$ . By combining the results, we obtain  $\text{MULTISTAB}(\mathcal{S}, [0, 2, 5]) = \{\langle 0, 3 \rangle, \langle 0, 11 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 4, 5 \rangle, \langle 5, 5 \rangle\}$ . Observe that  $\langle 0, 3 \rangle$  appears in two stab-query results and  $\langle 0, 11 \rangle$  appears in all stab-query results.

To improve on this situation, we will present a direct multi-stab-query procedure that circumvents the need of deduplication. Let  $\phi = [t_1, \dots, t_{|\phi|}]$  be a sorted sequence of timestamps, let  $R_i = \text{STAB}(\mathcal{S}, t_i)$ ,  $1 \leq i \leq |\phi|$ , let  $S_1 = R_1$ , and let  $S_j = R_j \setminus R_{j-1}$ ,  $2 \leq j \leq |\phi|$ . Notice that  $\text{MULTISTAB}(\mathcal{S}, \phi) = \bigcup_{1 \leq i \leq |\phi|} R_i = \bigcup_{1 \leq i \leq |\phi|} S_i$ . By definition, the sets  $S_1, \dots, S_{|\phi|}$  are all pair-wise disjoint. Hence, we can answer multi-stab-queries efficiently if we can compute the sets  $S_i$ ,  $1 \leq i \leq |\phi|$ , efficiently. Observe that an event is in  $S_j$  if and only if it is active at  $t_j$  but not at  $t_{j-1}$  (or any other timestamp in  $[t_1, \dots, t_{j-1}]$ ). First, we describe how to find parts of  $S_j$  stored in forest-points and the tail. Then, we describe how to find parts of  $S_j$  stored in a stab-tree. Finally, we provide necessary implementation details and analyze the complexity of the described multi-stab-query procedure. All details necessary to answer stab-queries efficiently can be derived from this multi-stab-query procedure.

**Searching in forest-points and the tail.** Let  $\mathcal{S} = (\mathcal{E}, \mathcal{I}, i_{\text{tail}})$ . To simplify presentation, we assume that  $\mathcal{E} \neq \emptyset$  and  $t_j$  is at-or-after the start of the first event in  $\mathcal{E}$ . (If these assumptions do not hold, we have  $S_j = \emptyset$ ). We also assume that  $t_{j-1} = -\infty$  if  $t_j = t_1$ . Under these assumptions, we need to search in the stab-forest to find all events in  $S_j$ . The first step is to identify if there exists a forest-point that covers  $t_j$ . If  $t_{j-1}$  is smaller than the start-time of any event in  $\mathcal{E}$ , then we start at the first forest-point in  $\mathcal{I}$ . Otherwise, we start at the forest-point that covers  $t_{j-1}$ . When visiting a forest-point  $F = (T, m)$ , we have one of the following four cases:

1. *F only covers events that start before  $t_j$ .* In this case,  $\max(F) < t_j$  and events in  $\text{events}(T)$  start before-or-at  $\max(F)$ . Hence,  $\text{events}(F) \cap S_j = \text{MAX}(F) \cap S_j$ . We have  $\text{events}(F) \cap S_j \neq \emptyset$  only when  $t_{j-1} < \max(F)$ . In this case, we compute  $\text{events}(F) \cap S_j$  by traversing both  $\text{MAX}_{n,\downarrow}(F)$  and  $\text{MAX}_{d,\downarrow}(F)$  and stop when we find the first event that stops before  $t_j$ . During this traversal, we may encounter events already active at  $t_{j-1}$ ; we skip over these events by not outputting them again. As an optimization, we notice that  $\text{MAX}(F) \cap S_j \subseteq \text{MAX}_{d,\downarrow}(F)$  if  $n\text{key}(m) \leq t_{j-1} < d\text{key}(m)$ . In this case, we can skip traversing  $\text{MAX}_{n,\downarrow}(F)$ . After processing this forest-point, proceed to the next forest-point.
2. *F represents events that start at  $t_j$  and  $t_j$  is covered by  $T$ .* In this case,  $\min(F) \leq t_j < n\text{key}(m)$  and  $\text{events}(F) \cap S_j = (\text{MAX}(F) \cup \text{events}(T)) \cap S_j$ . Due to  $t_j < d\text{key}(m)$ , we have  $\text{MAX}(F) \cap S_j = \text{MAX}_{n,\uparrow}(F) \cap S_j$ . We compute  $\text{MAX}_{n,\uparrow}(F) \cap S_j$  by traversing  $\text{MAX}_{n,\uparrow}(F)$  and stop when we find the first event that starts after  $t_j$ . Traversing  $\text{MAX}_{n,\uparrow}(F)$ , we encounter events in  $\text{MAX}(F)$  that start before-or-at  $t_{j-1}$ , followed by those that start after  $t_{j-1}$  and before-or-at  $t_j$ , followed by those that start after  $t_j$ . Hence, to avoid

unnecessary deduplication, we traverse  $\text{MAX}_{n,\uparrow}(F)$  starting at the first event that starts after  $t_{j-1}$  (we detail how to do so later). Next, we search for all events in  $\text{events}(T) \cap S_j$ . After searching in  $T$ , we have completed the computation of  $S_j$ .

3.  $F$  represents events that start at  $t_j$  and  $t_j$  is not covered by  $T$ . In this case,  $n\text{key}(m) \leq t_j \leq d\text{key}(m)$ . Events in  $\text{events}(T)$  start before  $n\text{key}(m)$ . Hence,  $\text{events}(F) \cap S_j = \text{MAX}(F) \cap S_j$ . We compute  $\text{events}(F) \cap S_j$  by traversing  $\text{MAX}_{n,\downarrow}(F)$  as in Case 1. If  $t_j = d\text{key}(m)$ , we also include  $\text{MAX}_{d,\downarrow}(F)$  entirely. We completed the computation of  $S_j$ .
4.  $F$  only covers events that start after  $t_j$ . In this case,  $t_j < \min(F)$ . We have  $\text{events}(F) \cap S_j = \emptyset$ , and, as we process forest-points ordered on the events they cover, this forest-point will not be reached.

We have  $\text{events}(i_{\text{tail}}) \cap S_j \neq \emptyset$  only if  $t_j$  is greater than any timestamp covered by  $\mathcal{I}$ . Let  $E$  be the event pointed at by  $i_{\text{tail}}$ . We have  $\text{events}(i_{\text{tail}}) \cap S_j \neq \emptyset$  only if  $t_{j-1} < E.\text{start} \leq t_j$ . In this case, we find all events in the tail that are active at  $t_j$  by traversing  $\mathcal{E}$  backwards starting at the end and stopping at either  $i_{\text{tail}}$  or at the first event that stops before  $t_j$ , whichever comes first. We notice that this traversal is a traversal on descending end-time order.

**Searching in a stab-tree.** The above only details how to process the max-lists of forest-points and the tail. To handle Case 2 above, we also need to describe how to compute  $\text{events}(T) \cap S_j$ . Assume that  $t_j \in \text{cover}(F)$  and  $t_j < n\text{key}(m)$ . We perform a binary-search-tree search on  $T$  until we find the node  $n$  with  $t \in \text{scope}(n)$ . For each node  $n'$  visited during this search, we have one of the following three cases:

5. If  $t < n\text{key}(n')$ , then we need to continue the search in  $\text{left}(n')$ . We have  $\text{events}(n') \cap S_j = (\text{events}(\text{left}(n')) \cup \text{LEFT}(n')) \cap S_j$ . We compute  $\text{LEFT}(n') \cap S_j$  by traversing  $\text{LEFT}_{n,\uparrow}(n')$  and stop when we find the first event that starts after  $t_j$ . Traversing  $\text{LEFT}_{n,\uparrow}(n')$ , we encounter events in  $\text{LEFT}(n')$  that start before-or-at  $t_{j-1}$ , followed by those that start after  $t_{j-1}$  and before-or-at  $t_j$ , followed by those that start after  $t_j$ . Hence, to avoid unnecessary deduplication, we traverse  $\text{LEFT}_{n,\uparrow}(n')$  starting at the first event that starts after  $t_{j-1}$  (we detail how to do so later).
6. If  $n\text{key}(n') \leq t \leq d\text{key}(n')$ , then we have found node  $n$ . We have  $\text{events}(n') \cap S_j = \text{LEFT}(n') \cap S_j$ . We compute  $\text{events}(n') \cap S_j$  by traversing  $\text{LEFT}_{n,\downarrow}(n')$  and stop when we find the first event that stops before  $t_j$ . During this traversal, we may encounter events already active at  $t_{j-1}$ ; we skip over these events by not outputting them again. If  $t_j = d\text{key}(n')$ , we also include  $\text{LEFT}_{d,\downarrow}(n')$  entirely. We completed the search in  $T$ .
7. If  $d\text{key}(n') < t$ , then we need to continue the search in  $\text{right}(n')$ . We have  $\text{events}(n') \cap S_j = (\text{events}(\text{right}(n')) \cup \text{LEFT}(n')) \cap S_j$ . We have  $\text{LEFT}(n') \cap S_j \neq \emptyset$  only when  $t_{j-1} < d\text{key}(n')$ . In this case, we compute  $\text{LEFT}(n') \cap S_j$  by traversing both  $\text{LEFT}_{n,\downarrow}(n')$  and  $\text{LEFT}_{d,\downarrow}(n')$  and stop when we find the first event that stops before  $t_j$ . During this traversal, we may encounter events already active at  $t_{j-1}$ ; we skip over these events by not outputting them again. As an optimization, we notice that  $\text{LEFT}(n') \cap S_j \subseteq \text{LEFT}_{d,\downarrow}(n')$  if  $n\text{key}(n') \leq t_{j-1} < d\text{key}(n')$ . In this case, we can skip traversing  $\text{LEFT}_{n,\downarrow}(n')$ .

**Analysis of multi-stab-queries.** To implement Cases 2 and 5 efficiently, we need to do some bookkeeping. For the relevant nodes  $n'$  and forest-point  $F$  on which Cases 2 and 5 applied while computing  $S_j$ , we need to keep track of the position of the first events in  $\text{LEFT}_{n,\uparrow}(n')$  and  $\text{MAX}_{n,\uparrow}(F)$  that start after  $t_j$ . In total, we need to keep track of at most  $\lceil \log |\mathcal{S}| \rceil$  different positions.

► **Example 4.2.** We repeat the query  $\text{MULTISTAB}(\mathcal{S}, [0, 2, 5])$  of Example 4.1 on the stab-forest  $\mathcal{S}$  of Example 3.3. When stabbing with 0, we traverse  $\text{MAX}_{n,\uparrow}(F_1)$  and output the first two events  $\langle 0, 3 \rangle$  and  $\langle 0, 11 \rangle$ . While searching in the stab-tree  $T_1$ , we do not find any further

events. Next, we stab with 2. When traversing  $\text{MAX}_{n,\uparrow}(F_1)$  we start at the third event,  $\langle 2, 3 \rangle$ , which we output. Next, we search in the stab-tree  $T_1$ , where we find  $\langle 1, 2 \rangle$ . During the stab with 5, we recognize that 5 is not covered by  $F_1$ . Hence, we traverse  $\text{MAX}_{n,\downarrow}(F_1)$  and  $\text{MAX}_{d,\downarrow}(F_1)$  to find any events that are still active at 5. The first event in  $\text{MAX}_{d,\downarrow}(F_1)$  is  $\langle 4, 5 \rangle$ , which we output. The first event in  $\text{MAX}_{n,\downarrow}(F_1)$  is  $\langle 0, 11 \rangle$ , which we skip over. We then search in  $F_2$  to find and output  $\langle 5, 6 \rangle$  and  $\langle 5, 5 \rangle$ .

We observe that the above multi-stab-query procedure will, in the worst case, read every event in the output of the multi-stab-query twice; once in a max-list or left-list that is sorted on ascending start-time order and once in a max-list or left-list that is sorted on descending end-time order. If the index  $\mathcal{I}$  has  $N$  stab-tree nodes, then the approach to compute  $S_j$  will navigate through up to  $\lceil \log N \rceil$  forest-points and stab-tree nodes. The multi-stab-query approach for computing  $S_j$  described above can easily be extended to also yield a pointer  $p_j$  to the first event in the event-list that starts after  $t_j$ , as used by the SKIPJOIN algorithm.

We can also compute  $S_j$  by traversing the event-list from the first event starting after  $t_{j-1}$  until the first event that starts after  $t_j$ . As long as this traversal of  $\mathcal{E}$  performs at most  $\lceil \log N \rceil$  memory operations, traversing  $\mathcal{E}$  will be faster. To choose between these two methods to compute  $S_j$ , we can use a simple test. Let  $q$  be the position of the first event starting after  $t_{j-1}$ . Let  $c$  be the *threshold constant* representing the number of events one can read from the event-list in a single memory operation. To choose between the two approaches to compute  $S_j$ , we check if the event at position  $q + c \lceil \log N \rceil$  does not exist or, otherwise, starts at-or-after  $t_{j+1}$ . With this approach, we need to change the processing of left-lists and max-lists to, additionally, skip over any events we found by traversing the event-list. Hence, with this change, the above process will read every event in the output at-most thrice.

► **Theorem 4.3.** *Let  $\mathcal{S}$  be a stab-forest and let  $\phi$  be a sorted sequence of timestamps.  $\text{MULTISTAB}(\mathcal{S}, \phi)$  can be answered in  $\mathcal{O}(\min(|\phi| \log |\mathcal{S}|, |\phi| + |\mathcal{S}|) + |\text{output}|)$ .*

## 5 Stab-Forest Maintenance

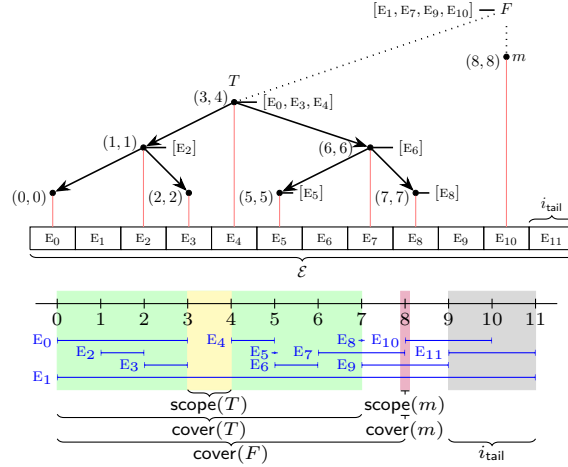
The stab-forest is designed to be a dynamic data structure to which events can be appended efficiently. Here, we show how to append events using the assumption that events are appended in lexicographical order on (start, end)-time. In Appendix A, we generalize the principles of the stab-forest to support less-restrictive semantics in equally efficient ways.

To support appending events that are ordered lexicographically on (start, end)-times, we start by describing an algorithm to put appended events in newly created forest-points. When appending a new event  $E'$  to stab-forest  $\mathcal{S} = (\mathcal{E}, \mathcal{I}, i_{\text{tail}})$ , we distinguish the following cases:

1. If  $|\mathcal{E}| = 0$ , then append event  $E'$  to the end of  $\mathcal{E}$  and set  $i_{\text{tail}} := 0$ , the offset of  $E'$  in  $\mathcal{E}$ .
2. Else, if  $\mathcal{E}[i_{\text{tail}}].\text{start} = E'.\text{start}$ , then append event  $E'$  to the end of  $\mathcal{E}$ .
3. In all other cases,  $\mathcal{E}[i_{\text{tail}}].\text{start} < E'.\text{start}$ . Let  $L$  be the list of events in the event-list starting at  $i_{\text{tail}}$ . Create a fresh leaf node  $l$  and a fresh forest-point  $F = (T, l)$  in  $\mathcal{I}$  with  $T$  an empty tree. We set

$$\begin{aligned} \text{dkey}(l) &= E'.\text{start}; & i_{\text{data}}(l) &= i_{\text{tail}}; \\ \text{left}(l) &= \perp; & \text{right}(l) &= \perp; \\ \text{LEFT}(l) &= \emptyset; & \text{MAX}(F) &= L. \end{aligned}$$

If  $|\mathcal{I}| = \emptyset$ , then set  $\text{nkey}(l) = \text{dkey}(l)$ . Else, set  $\text{nkey}(l) = \max(F') + 1$ , with  $F'$  the last forest-point in  $\mathcal{I}$ . After constructing  $F$ , append  $F$  to the end of  $\mathcal{I}$ . Finally, append  $E'$  to  $\mathcal{E}$  and set  $i_{\text{tail}} := |\mathcal{E}| - 1$ , the offset of  $E'$  in  $\mathcal{E}$ .



■ **Figure 3** The stab-forest obtained after adding event  $\langle 9, 11 \rangle$  to the stab-forest of Figure 2, *right*.

Remember that the event-list is ordered lexicographically on  $(\text{start}, \text{end})$ -times and that the current tail  $L$  only has events with a start-time  $E.\text{start}$ . Hence, by reversing  $L$ , we can directly construct  $\text{MAX}_{d,\downarrow}(F)$ , and, in this case, we have  $\text{MAX}_{d,\downarrow}(F) = \text{MAX}(F)$ .

We observe that the above algorithm might invalidate Invariant  $x$  (Section 3), as a newly added forest-point can have the same height as the previous last forest-point in  $\mathcal{I}$ . To restore Invariant  $x$ , we will repeatedly merge the last two forest-points in  $\mathcal{I}$  until they no longer have the same height. Let  $F_1 = (T_1, m_1)$  and  $F_2 = (T_2, m_2)$  be adjacent forest-points with  $h = \text{height}(F_1) = \text{height}(F_2)$  and  $\min(F_2) = \max(F_1) + 1$ . We merge these forest-points into a single forest-point  $F = (T, m_2)$  with

$$\begin{aligned} \text{root}(T) &= m_1; \\ \text{left}(m_1) &= \text{root}(T_1); \\ \text{right}(m_1) &= \text{root}(T_2); \\ \text{LEFT}(m_1) &= \{E \in \text{MAX}(F_1) \mid E.\text{end} \leq \max(T_2)\}; \\ \text{MAX}(F) &= (\text{MAX}(F_1) \setminus \text{LEFT}(m_1)) \cup \text{MAX}(F_2). \end{aligned}$$

In the above, the necessary parts of  $\text{LEFT}(m_1)$  and  $\text{MAX}(F)$  can be constructed via straightforward merge-procedures on the parts of  $\text{MAX}(F_1)$  and  $\text{MAX}(F_2)$ .

► **Example 5.1.** Consider the stab-forest of Example 3.3. We add the event  $\langle 9, 11 \rangle$ , resulting in the stab-forest visualized in Figure 3.

One can show that the above forest-point merge maintains the Invariants i–iv, v, and ix (Section 3). Using this result, it is straightforward to prove that the described append method is sound. We conclude:

► **Theorem 5.2.** *Let  $L$  be a list of events ordered lexicographically on  $(\text{start}, \text{end})$ -times. The structure obtained from starting with an empty stab-forest and appending each of the events in  $L$  in order is a stab-forest. This stab-forest is constructed in  $\mathcal{O}(|L| \log |L|)$  and will use  $\mathcal{O}(|L|)$  storage. Additional events can be added to the stab-forest in amortized  $\mathcal{O}(\log |L|)$ .*

Notice that the maintenance algorithm only operates on forest-points and does not change the stab-forests stored within them. Indeed, the constructed stab-trees are *static*, while merging of forest-points can be implemented via efficient array-merge-operations on their max-lists.

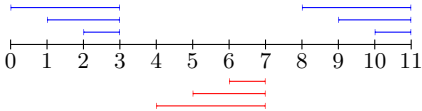
## 6 Empirical Evaluation

Finally, we provide an empirical study to showcase the practical performance of the stab-forest and the skip-join algorithm. We have implemented the stab-forest, the temporal query operations, and the temporal join algorithms presented in this paper in C++. Open-source code of the full implementation of the data structures, algorithms, and supporting tooling used can be found at <https://jhellings.nl/projects/skipjoin/>. In our implementation, we used 32bit unsigned integers to represent timestamps. The programs were compiled with the Microsoft C/C++ Compiler Version 19.13.26132 for x64, part of Visual Studio 2017, and run on a workstation with an Intel Core i5-4670 processor and 16GB of internal memory. In each experiment, the algorithms used write out their query results to a dynamic array (implemented by the standard `vector` data structure).

As a baseline for comparison, we implemented the forward-scan algorithm `FWDSKAN`, which is reported to be among the fastest internal-memory temporal join algorithms [6]. As our `SKIPJOIN` algorithm is based on `FWDSKAN`, our experiments not only showcase how `SKIPJOIN` performs compared to the state-of-the-art, but also allows for a detailed look at the benefits and costs of `SKIPJOIN`. To further examine the behavior of `SKIPJOIN` in detail, we tested with three variants; namely `SKIPJOIN- $\mathcal{E}$`  that uses the event-list exclusively for answering stab-queries, `SKIPJOIN- $\mathcal{I}$`  that uses the stab-forest index exclusively for answering stab-queries, and normal `SKIPJOIN` that uses a *threshold constant*  $c = 16$  to choose between using the event-list and the stab-forest index.

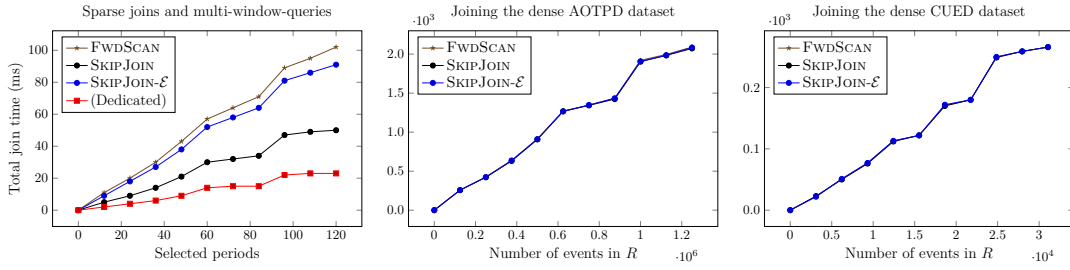
In our experiments, we used two real-world datasets. The first real-world dataset we used is the *Airline On-Time Performance Data* (AOTPD) dataset [8], which contains flight-events (takeoff and duration) over a ten-year period. The second real-world dataset we used is the *Civil Unrest Event Data* (CUED) dataset [10], a set of civil unrest events in recent human history. The details of both datasets can be found in Figure 4, *left*. We also used a synthetically generated *gap dataset*. This dataset consists of two lists  $\mathcal{R}$  and  $\mathcal{S}$  that contain consecutive non-overlapping groups of  $G$  events (the *gap size*) that are placed alternatingly in either  $\mathcal{R}$  or  $\mathcal{S}$ . Figure 4, *right*, visualizes such a dataset with twelve events grouped in groups of  $G = 3$  events.

	AOTPD [8]	CUED [10]
Number of Events	61, 100, 539	62, 141
Start date	July, 2007	February, 1946
End date	June, 2017	November, 2005
Minimal duration	0 minutes	0 days
Maximum duration	1, 350 minutes	18, 407 days

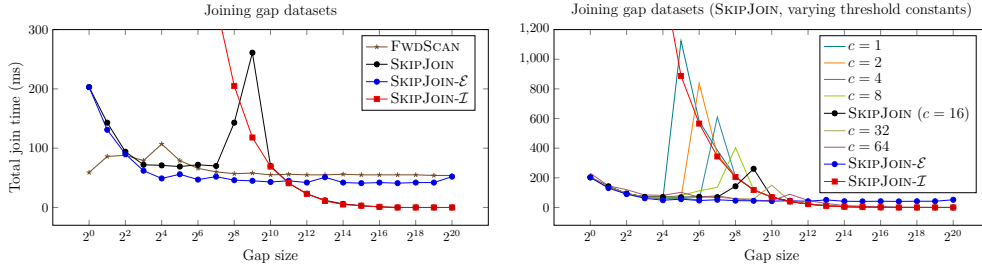


■ **Figure 4** The datasets used in our evaluation. *Left*, statistics on the real-life datasets used. *Right*, gap datasets  $\mathcal{R}$  and  $\mathcal{S}$  with gap size 3.

**Temporal joins on sparse datasets.** First, we investigated the performance of temporal join algorithms in cases where only a few events are part of the join result, the situation for which our `SKIPJOIN` algorithm is designed. We used the temporal join algorithms to select a set of days from the AOTPD dataset. The temporal join algorithms select these specified days by joining the AOTPD dataset with a filter dataset that contains the to-be-selected days. As an additional point of reference, we compared the temporal join algorithms with a dedicated multi-window-query implementation that selects the same days. In this experiment, we selected the 7-th day from each of the first  $n$  out of 120 months. The results of our



■ **Figure 5** Temporal joins on sparse and dense datasets. On the *left*, we use temporal joins to select events from specific days in the AOTPD dataset. We compare these sparse temporal joins with a dedicated multi-window-query implementation to select the specific days. In the *middle* and on the *right*, we present the join performance on dense datasets, joining parts of the AOTPD dataset (*middle*) and parts of the CUED dataset (*right*). In these two cases, all three algorithms perform approximately the same.



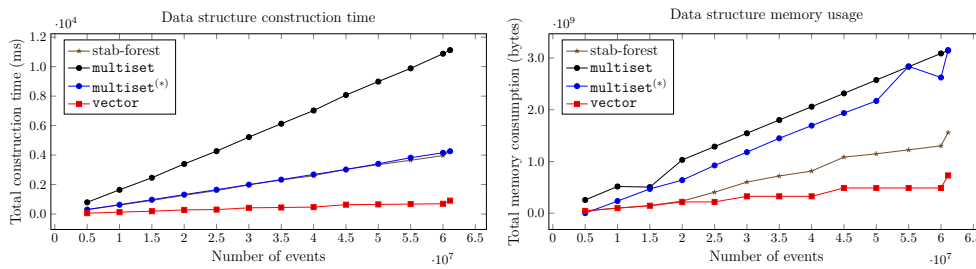
■ **Figure 6** The behavior of SKIPJOIN: temporal join performance on sparse gap datasets, in which the gap-size determines the amount of data SKIPJOIN can skip over.

measurements can be found in Figure 5, *left*. As expected, SKIPJOIN benefits heavily from skipping over the non-relevant portions of the event-list. We also observe that the gain in performance comes from the usage of the stab-forest index, as SKIPJOIN- $\mathcal{E}$  only performs slightly better than FWDSKAN. Finally, we observe that the performance of SKIPJOIN comes close to our dedicated algorithm; showing that the performance of SKIPJOIN is even acceptable for implementing more specialized operators.

**Temporal joins on dense datasets.** Next, we investigated the performance of temporal join algorithms in cases where most events are part of the join result, e.g., in which almost every event in each dataset joins with an event in the other dataset. This is the situation for which the traditional FWDSKAN algorithm is designed. In this experiment, we used two datasets, namely the CUED dataset and a randomly selected fragment of 2,500,000 events from the AOTPD dataset. For this experiment, we took a dataset, split that dataset into two halves  $R$  and  $S$ , shuffled  $R$ , and joined the first 0%, 10%, ..., 100% of the shuffled  $R$  with the entirety of  $S$ . The results of our measurements can be found in Figure 5, *middle* and *right*. We observe that in the setting of joining densely correlated datasets, there is no real difference between the SKIPJOIN-family of algorithms and the FWDSKAN algorithm, even though the SKIPJOIN-family of algorithms have higher complexity and overhead.

**The behavior of SKIPJOIN.** Third, we investigated the exact behavior of SKIPJOIN in situations where the algorithm is triggered to skip over data. To have full control over the amount of skipping possible, we used gap datasets with  $64 \cdot 2^{20}$  events and a gap





■ **Figure 7** The costs of the stab-forest: construction time (*left*) and memory usage (*right*) of the stab-forest in comparison to various other data structures.

size of  $G = 2^0, 2^1, \dots, 2^{20}$ . We ran the SKIPJOIN algorithm with threshold constants  $c \in \{1, 2, 4, 8, 16, 32, 64\}$ , and compared the SKIPJOIN algorithm with FWDSCAN, SKIPJOIN- $\mathcal{E}$ , and SKIPJOIN- $\mathcal{I}$ . The results of our measurements can be found in Figure 6. From the measurements, we conclude that the SKIPJOIN-family of algorithms is favorable as soon as they can skip over at least 4 events (the performance gain of skipping over a single event does not justify the overhead introduced by skipping). Skipping over events via the event-list, as SKIPJOIN- $\mathcal{E}$  does, provides a small improvement over FWDSCAN. Skipping over events via the index, as SKIPJOIN- $\mathcal{I}$  does, can provide order-of-magnitudes improvements over FWDSCAN, but only if sufficient events are skipped over. Due to this, it is important to use a threshold constant that is well-suited to the details of the underlying hardware. For our setting, the threshold constant  $c = 16$ , as used in SKIPJOIN, provides acceptable performance in all cases as it usually provides performance that is close to the fastest algorithm.

**The costs of the stab-forest.** In our final experiment, we looked at the costs of stab-forest maintenance. More specifically, we investigated the construction cost (by appending events one-by-one) and the memory consumption of a fully constructed stab-forest. We compared the construction of the stab-forest with the construction of three standard C++ data structures:

1. **vector**. We use a **vector**, a bare bones dynamic array implementation, as a lower bound for representing the underlying event data without any indices.
2. **multiset**. We use a **multiset**, which is implemented as a self-balancing binary search tree. The **multiset** provides a lower bound on the cost of constructing and maintaining dynamic general-purpose interval data structures, as all dynamic general-purpose interval data structures are built using self-balancing binary search trees at their core [4, 9, 20].
3. **multiset<sup>(\*)</sup>**. Finally, we use a **multiset** in which each insert operation uses placement hints to allow the data structure to optimize for the append-only workload we provided. This **multiset** implementation provides a lower bound on the cost of constructing and maintaining dynamic general-purpose interval data structures that provide optimized append operations. We denote this usage of the **multiset** by **multiset<sup>(\*)</sup>**.

In this experiment, we used the AOTPD dataset. For each of the data structures, we measured the time it took to append the first  $N$  events from this dataset to the data structure ( $N = 5 \cdot 10^6, 10 \cdot 10^6, \dots, 60 \cdot 10^6$ ). The results of our measurements can be found in Figure 7. Unsurprisingly, appending data to the stab-forest is slower than appending to a **vector**, as the **vector** is the underlying representation of the event-list. The cost of appending to a stab-forest is on-par with the cost of appending to a **multiset<sup>(\*)</sup>** binary search tree, showing that the stab-forest construction only incurs minimal overhead. Finally, appending to a fully dynamic **multiset** binary search tree, which provides the lower bound for dynamic general-purpose interval data structures, is much more costly than appending to

stab-forests. This supports our choice for designing stab-forests with append-only semantics. With respect to memory usage, we see that the stab-forest is much more compact than a binary search tree (even with all time-based augmentations), as it only requires a single stab-tree node per start-time, whereas the `multiset` uses a single search tree node per event.

## 7 Conclusion

We set out to develop high-performance internal-memory temporal join algorithms for dynamically generated heavily skewed data. Towards this goal, we proposed the stab-forest, the multi-stab-query, and the skip-join temporal join algorithm. In our evaluation, we showed that the skip-join algorithm is capable of significantly speeding up temporal joins of heavily skewed data. Our experiments also showed that the overhead of the skip-join algorithm when joining non-skewed data is insignificant, making our algorithm highly performant in all cases.

---

## References

- 1 Pankaj K Agarwal and Jeff Erickson. *Geometric range searching and its relatives*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, 1999.
- 2 Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015. doi:10.14778/2824032.2824076.
- 3 Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, and Jeffrey Scott Vitter. Scalable sweeping-based spatial join. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 570–581. Morgan Kaufmann Publishers Inc., 1998.
- 4 Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM Journal on Computing*, 32(6):1488–1508, 2003. doi:10.1137/S009753970240481X.
- 5 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.
- 6 Panagiotis Bouros and Nikos Mamoulis. A forward scan based plane sweep algorithm for parallel interval joins. *Proceedings of the VLDB Endowment*, 10(11):1346–1357, 2017. doi:10.14778/3137628.3137644.
- 7 Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. *SIGMOD Record*, 22(2):237–246, 1993. doi:10.1145/170036.170075.
- 8 Bureau of Transportation Statistics, United States Department of Transportation. Airline on-time performance data, 2017. URL: [https://www.transtats.bts.gov/Tables.asp?DB\\_ID=120](https://www.transtats.bts.gov/Tables.asp?DB_ID=120).
- 9 Yi-Jen Chiang and Roberto Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, 1992. doi:10.1109/5.163409.
- 10 Cline Center for Democracy. Speed project - civil unrest event data, 2012. URL: <https://clinecenter.illinois.edu/project/human-loop-event-data-projects/SPEED>.
- 11 Herbert Edelsbrunner. A new approach to rectangle intersections part I. *International Journal of Computer Mathematics*, 13(3–4):209–219, 1983. doi:10.1080/00207168308803364.
- 12 Herbert Edelsbrunner. A new approach to rectangle intersections part II. *International Journal of Computer Mathematics*, 13(3–4):221–229, 1983. doi:10.1080/00207168308803365.
- 13 Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Join operations in temporal databases. *The VLDB Journal*, 14(1):2–29, 2005. doi:10.1007/s00778-003-0111-3.
- 14 Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter Michael Fischer, Donald Kossmann, Franz Färber, and Norman May. Timeline index: A unified data structure for processing queries on temporal data in SAP HANA. In *Proceedings of the 2013 ACM*



- SIGMOD International Conference on Management of Data*, pages 1173–1184. ACM, 2013. doi:10.1145/2463676.2465293.
- 15 Hans-Peter Kriegel, Marco Pötke, and Thomas Seidl. Managing intervals efficiently in object-relational databases. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 407–418. Morgan Kaufmann Publishers Inc., 2000.
  - 16 Jiří Matoušek. Geometric range searching. *ACM Computing Surveys*, 26(4):422–461, 1994. doi:10.1145/197405.197408.
  - 17 Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985. doi:10.1137/0214021.
  - 18 Dinesh P. Mehta and Sartaj Sahni. *Handbook Of Data Structures And Applications, Second Edition*. Chapman & Hall/CRC, 2017.
  - 19 A. Montplaisir-Gonçalves, N. Ezzati-Jivan, F. Wininger, and M. R. Dagenais. State history tree: An incremental disk-based data structure for very large interval data. In *2013 International Conference on Social Computing*, pages 716–724. IEEE, 2013. doi:10.1109/SocialCom.2013.107.
  - 20 Mark H. Overmars. *The Design of Dynamic Data Structures*. Springer, 1983.
  - 21 Danila Piatov, Sven Helmer, and Anton Dignös. An interval join optimized for modern hardware. In *2016 IEEE 32nd International Conference on Data Engineering*, pages 1098–1109. IEEE, 2016. doi:10.1109/ICDE.2016.7498316.
  - 22 Betty Salzberg and Vassilis J. Tsotras. Comparison of access methods for time-evolving data. *ACM Computing Surveys*, 31(2):158–221, 1999. doi:10.1145/319806.319816.
  - 23 Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., 1990.
  - 24 Peter Sanders. *Memory Hierarchies — Models and Lower Bounds*, pages 1–13. Springer, 2003. doi:10.1007/3-540-36574-5\_1.
  - 25 Arie Segev and Himawan Gunadhi. Event-join optimization in temporal relational databases. In *Proceedings of the 15th international conference on Very large data bases*, pages 205–215. Morgan Kaufmann Publishers Inc., 1989.
  - 26 Donghui Zhang, Vassilis J. Tsotras, and Bernhard Seeger. Efficient temporal join processing using indices. In *Proceedings 18th International Conference on Data Engineering*, pages 103–113. IEEE, 2002. doi:10.1109/ICDE.2002.994701.

## **A** Variants of Stab-Forests and their Maintenance

The stab-forest is designed to be a dynamic data structure to which events can be appended efficiently. In the main text, we showed how stab-forests support appending events using the assumption that events are appended in lexicographical order on (start, end)-time. Here, we the principles of the stab-forest to support two less-restrictive semantics in equally efficient ways.

**Increasing start-time order semantics.** The ordering on end-times is only used to assure that the tail is always lexicographically ordered on (start, end)-times: we only need to keep the tail ordered on end-times as all events in the tail have the same start-time. Hence, we can support appending events only ordered on start-time if we store the tail in a search tree. We then simply copy over the tail to the event-list whenever appending an event triggers the construction of a fresh forest-point.

One can also opt to not keep the tail sorted on end-times, but only enforce this ordering when creating a fresh forest-point. In such a design, queries can only access a *history* of the data that does not include the *current events* in the tail. This approach can also be used to support streams of events for which *watermarks* provide an after-the-fact guarantee on the ordering of past events [2].

**Timestamp-based semantics.** In streaming data processing and in versioned databases, the start-time and end-time of events are usually known when the event starts and ends, respectively [22]. For these applications, it is natural to append the start- and end-times when they happen, as separate operations. The stab-forest can be generalized to support these applications. In specific, we show how a stab-forest can support the following *timestamp-based semantics*. When an event starts, it is *appended* to the stab-forest by registering its *start-time*. On successful registration, the stab-forest returns an *event-handle* that can be used to update the event. When the event ends, one uses the event-handle to update the *end-time* of the event. We assume that all start- and end-times are appended and updated on increasing timestamps.

► **Example A.1.** Consider a versioned database. At  $t_1$  record  $r$  gets created, at  $t_2$  record  $r$  gets updated, and, finally, at  $t_3$  record  $r$  gets deleted. At  $t_1$ , we append an event  $E_1$  representing record  $r$  with start-time  $t_1$  (and no end-time). At  $t_2$ , we update  $E_1$  by setting the end-time  $t_2$ . We create a new event  $E_2$  representing the updated record  $r$  with start-time  $t_2$  (and no end-time). Finally, at  $t_3$ , we update  $E_2$  by setting the end-time  $t_3$ .

Stab-forests with timestamp-based semantics are an obvious choice when adding skip-join style techniques to endpoint-based join algorithms, e.g., the algorithm of Piatov et al. [21].

To maintain all the invariants under the timestamp-based semantics, we need to make a few changes to the stab-forest structures. We represent open-ended events with start-time  $v$  and without an end-time by  $E = \langle v, \infty \rangle$ . Each copy of such an open-ended event  $E$  in the event-list and in max-lists keeps a reference to the *event-handle*, which we describe in detail later. Each stab-tree node  $n$ , which represents events with start-time  $\text{dkey}(n)$ , is augmented with an  $\infty$ -pointer  $i_\infty(n)$  that holds the offset of the first open-ended event in  $\mathcal{E}$  with start-time  $\text{dkey}(n)$  (if such an event exists). Each forest-point  $F$  is augmented with  $\infty$ -pointers  $i_\infty(\text{MAX}_{n,\downarrow}(F))$  and  $i_\infty(\text{MAX}_{d,\downarrow}(F))$  that hold the offsets of the last open-ended events in  $\text{MAX}_{n,\downarrow}(F)$  and  $\text{MAX}_{d,\downarrow}(F)$  (if such events exist). Finally, we use an  $\infty$ -pointer  $i_{\infty\text{-tail}}$  to hold the offset of the first open-ended event in the tail.

For every open-ended event  $E = \langle v, \infty \rangle$ , we maintain an *event-handle*

$$\text{handle}(E) = (i_{\mathcal{E}}, n, F, i_{\text{MAX}_{n,\downarrow}(F)}, i_{\text{MAX}_{d,\downarrow}(F)}, i_{\text{MAX}_{n,\uparrow}(F)}),$$

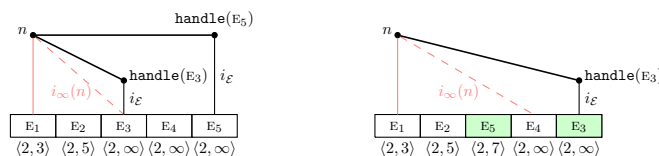
in which  $i_{\mathcal{E}}$  is the offset of the copy of  $E$  in  $\mathcal{E}$ ,  $n$  is a reference to the stab-tree node with  $\text{dkey}(n) = v$  (if  $E$  is not in the tail),  $F$  is the reference to the forest-point that has  $E$  in its max-list (if  $E$  is not in the tail), and  $i_{\text{MAX}_{n,\downarrow}(F)}$ ,  $i_{\text{MAX}_{d,\downarrow}(F)}$ , and  $i_{\text{MAX}_{n,\uparrow}(F)}$  are offsets of the copies of  $E$  in these max-lists (if such copies exist).

When a start-time  $v$  is appended to the stab-forest, the event  $E = \langle v, \infty \rangle$  and event-handle  $\text{handle}(E)$  are constructed and  $E$  is appended to the tail. Appending an event to the tail can cause the construction and merging of forest-points. During this forest-point maintenance, the event-handles of open-ended events need to be kept up-to-date, which can be done in constant time per involved event. Finally, a reference to  $\text{handle}(E)$  is returned.

When an end-time  $w$  for open-ended event  $E$  is updated in the stab-forest, one uses the reference to event-handle  $\text{handle}(E) = (i_{\mathcal{E}}, n, F, i_{\text{MAX}_{n,\downarrow}(F)}, i_{\text{MAX}_{d,\downarrow}(F)}, i_{\text{MAX}_{n,\uparrow}(F)})$ . First, we consider the steps necessary to update the end-time when  $E$  is not in the tail:

1. The copy of  $E$  in  $\mathcal{E}$  is updated by setting  $\mathcal{E}[i_{\mathcal{E}}].\text{end} := w$ . Then, to restore the lexicographic order on (start, end)-times in  $\mathcal{E}$ , the event at  $\mathcal{E}[i_{\mathcal{E}}]$  is swapped with the event at  $\mathcal{E}[i_\infty(n)]$ . Next, the offsets  $i_{\mathcal{E}}$  in the handles of the swapped events are updated. Finally,  $i_\infty(n)$  is incremented by setting  $i_\infty(n) := i_\infty(n) + 1$ .

This sequence of steps will update  $\mathcal{E}$  and restore the lexicographic order in  $\mathcal{E}$ . Observe that the end-times arrive in order. Hence, the end-time  $w$  of  $E$  comes after all earlier-updated



■ **Figure 8** Updating an event by setting the end-time. On the *left*, the stab-forest before the update. On the *right*, the situation after setting  $E_5.\text{end} := 7$ . In this sketch, only details relevant to the update are included.

end-times and swapping  $E$  to offset  $i_\infty(n)$  puts  $E$  directly after all other events with the same start-time and with a smaller end-time. All other open-ended events with the same start-time (including the event that got swapped with  $E$ ) follow  $E$  and, hence, are still in a valid order. Consequently, incrementing  $i_\infty(n)$  will assure that  $i_\infty(n)$  once again points to the first open-ended event with start-time  $\text{dkey}(n)$  (if such an event exists).

2. If a copy of  $E$  is in a max-list  $\text{MAX}_{n,\uparrow}(F)$ , then this copy of  $E$  is updated by setting  $\text{MAX}_{n,\uparrow}(F)[i_{\text{MAX}_{n,\uparrow}(F)}] := w$ . This update does not affect the start-time ordering of events in  $\text{MAX}_{n,\uparrow}(F)$ , hence, no further change to  $\text{MAX}_{n,\uparrow}(F)$  is necessary.
3. If a copy of  $E$  is in a max-list  $\text{MAX}_{n,\downarrow}(F)$ , then this copy of  $E$  is updated by setting  $\text{MAX}_{n,\downarrow}(F)[i_{\text{MAX}_{n,\downarrow}(F)}] := w$ . This update does affect the end-time ordering of events  $\text{MAX}_{n,\downarrow}(F)$ . Similar to how the ordering of  $\mathcal{E}$  is restored by a swap, the ordering in  $\text{MAX}_{n,\downarrow}(F)$  is restored by swapping value  $\text{MAX}_{n,\downarrow}(F)[i_{\text{MAX}_{n,\downarrow}(F)}]$  and value  $\text{MAX}_{n,\downarrow}(F)[i_\infty(\text{MAX}_{n,\downarrow}(F))]$ , updating the relevant handles, and, finally, decrementing  $i_\infty(\text{MAX}_{n,\downarrow}(F))$  by setting  $i_\infty(\text{MAX}_{n,\downarrow}(F)) := i_\infty(\text{MAX}_{n,\downarrow}(F)) - 1$ .
4. Finally, if a copy of  $E$  is in a max-list  $\text{MAX}_{d,\downarrow}(F)$ , then this copy is updated analogous to the previous case.

When  $E$  is in the tail, a swap of  $\mathcal{E}[i_\mathcal{E}]$  and  $\mathcal{E}[i_{\infty\text{-tail}}]$ , followed by incrementing  $i_{\infty\text{-tail}}$  suffices (once again similar to how the ordering of  $\mathcal{E}$  is restored by a swap). After updating the end-time  $w$  for event  $E$ , the handle  $\text{handle}(E)$  can be destroyed.

► **Example A.2.** Consider a stab-tree node  $n$  with  $\text{dkey}(n) = 2$ , pointing to an event-list with events  $\langle 2, 3 \rangle$ ,  $\langle 2, 5 \rangle$ ,  $\langle 2, \infty \rangle$ ,  $\langle 2, \infty \rangle$ , and  $\langle 2, \infty \rangle$ . We wish to update the last event,  $E_5 = \langle 2, \infty \rangle$ , by setting its end-time to 7. In Figure 8, *left* and *right*, we sketched this setting before and after updating end-time 7.



# On the Decidability of a Fragment of preferential LTL

**Anasse Chafik**

CRIL, University of Artois & CNRS, Arras, France  
chafik@cril.fr

**Fahima Cheikh-Alili**

CRIL, University of Artois & CNRS, Arras, France  
cheikh@cril.fr

**Jean-François Condotta**

CRIL, University of Artois & CNRS, Arras, France  
condotta@cril.fr

**Ivan Varzinczak**

CRIL, University of Artois & CNRS, Arras, France  
varzinczak@cril.fr

---

## Abstract

Linear Temporal Logic (LTL) has found extensive applications in Computer Science and Artificial Intelligence, notably as a formal framework for representing and verifying computer systems that vary over time. Non-monotonic reasoning, on the other hand, allows us to formalize and reason with exceptions and the dynamics of information. The goal of this paper is therefore to enrich temporal formalisms with non-monotonic reasoning features. We do so by investigating a preferential semantics for defeasible LTL along the lines of that extensively studied by Kraus et al. in the propositional case and recently extended to modal and description logics. The main contribution of the paper is a decidability result for a meaningful fragment of preferential LTL that can serve as the basis for further exploration of defeasibility in temporal formalisms.

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

**Keywords and phrases** Knowledge Representation, non-monotonic reasoning, temporal logic

**Digital Object Identifier** 10.4230/LIPIcs.TIME.2020.19

**Related Version** [https://github.com/calleann/Preferential\\_LTL](https://github.com/calleann/Preferential_LTL).

## 1 Introduction

Specification and verification of dynamic computer systems is an important task, given the increasing number of new computer technologies being developed. Recent examples include blockchain technology and various existing tools for home automation of the different production chains provided by Industry 4.0. Therefore, it is fundamental to ensure that systems based on them have the desired behavior but, above all, satisfy safety standards. This becomes even more critical with the increasing deployment of artificial intelligence techniques as well as the need to explain their behaviors.

Several approaches for qualitative analysis of computer systems have been developed. Among the most fruitful are the different families of temporal logic. The success of these is due mainly to their simplified syntax compared to that of first-order logic, their intuitive syntax, semantics and their good computational properties. One of the members of this family is Linear Temporal Logic [15, 19], known as *LTL*, is widely used in formal verification and specification of computer programs.



© Anasse Chafik, Fahima Cheikh-Alili, Jean-François Condotta, and Ivan Varzinczak;  
licensed under Creative Commons License CC-BY

27th International Symposium on Temporal Representation and Reasoning (TIME 2020).

Editors: Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald; Article No. 19; pp. 19:1–19:19

Leibniz International Proceedings in Informatics



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

Despite the success and wide use of linear temporal logic, it remains limited for modeling and reasoning about the real aspects of computer systems or those that depend on them. In fact, computer systems are not either 100% secure or 100% defective, and the properties we wish to check may have innocuous and tolerable exceptions, or conversely, exceptions that must be carefully addressed in order to guarantee the overall reliability of the system. Similarly, the expected behavior of a system may be correct not for all possible execution, but rather for its most “normal” or expected executions.

It turns out that *LTL*, because it is a logical formalism of the so-called classical type, whose underlying reasoning is that of mathematics and not that of common sense, does not allow at all to formalize the different nuances of the exceptions and even less to treat them. First of all, at the level of the object language (that of the logical symbols), it has operators behaving monotonically, and at the level of reasoning, possesses a notion of logical consequence which is monotonic too, and consequently, it is not adapted to the evolution of defeasible facts.

Non-monotonic reasoning (NMR), on the other hand, allows to formalize and reason with exceptions, it has been widely studied by the AI community for over 40 years now. Such is the case of Kraus et al. [12], known as the KLM approach.

However, the major contributions in this area are limited to the propositional framework. It is only recently that some approaches to non-monotonic reasoning, such as belief revision, default rules and preferential approaches, have been studied for more expressive logics than propositional logic, including modal [3, 5] and description logics [4, 9]. The objective of our study is to establish a bridge between temporal formalisms for the specification and verification of computer systems and approaches to non-monotonic reasoning, in particular the preferential one, which satisfactorily solves the limitations raised above.

In this paper, we define a logical framework for reasoning about defeasible properties of program executions, we investigate the integration of preferential semantics in the case of *LTL*, hereby introducing preferential linear temporal logic  $LTL^\sim$ . The remainder of the present paper is structured as follows: In Section 3 we set up the notation and appropriate semantics of our language. In Sections 4, 5 and 6, we investigate the satisfiability problem of this formalism. The appendix contains proofs of results in this paper. The remaining proofs can be viewed in [https://github.com/calleann/Preferential\\_LTL](https://github.com/calleann/Preferential_LTL).

## 2 Preliminaries: LTL and the KLM approach to NMR

Let  $\mathcal{P}$  be a finite set of *propositional atoms*. The set of operators in the *Linear Temporal Logic* can be split into two parts: the set of *Boolean connectives* ( $\neg, \wedge$ ), and that of *temporal operators* ( $\square, \diamond, \circ, \mathcal{U}$ ), where  $\square$  reads as *always*,  $\diamond$  as *eventually*,  $\circ$  as *next* and  $\mathcal{U}$  as *until*. The set of well-formed sentences expressed in *LTL* is denoted by  $\mathcal{L}$ . Sentences of  $\mathcal{L}$  are built up according to the following grammar:  $\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \square\alpha \mid \diamond\alpha \mid \circ\alpha \mid \alpha\mathcal{U}\alpha$ .

Let the set of natural numbers  $\mathbb{N}$  denote time points. A *temporal interpretation*  $I$  is a mapping function  $V : \mathbb{N} \rightarrow 2^{\mathcal{P}}$  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 conditions of LTL sentences are defined as follows, where  $I$  is a temporal interpretation and  $t$  a time point in  $I$ :

- $I, t \models p$  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  $I, t + 1 \models \alpha$ ;
- $I, t \models \alpha U \alpha'$  if  $I, t' \models \alpha'$  for some  $t' \geq t$  and for all  $t \leq t'' < t'$  we have  $I, t'' \models \alpha$ .

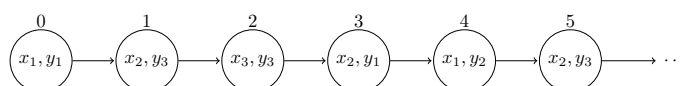
We say  $\alpha \in \mathcal{L}$  is *satisfiable* if there are  $I$  and  $t \in \mathbb{N}$  such that  $I, t \models \alpha$ .

We now give a brief outline to Kraus et al.'s [12] approach to non-monotonic reasoning. A propositional *defeasible consequence relation*  $\sim$  [12] is defined as a binary relation on sentences of an underlying propositional logic. The semantics of preferential consequence relation is in terms of *preferential models*: A preferential model on a set of atomic propositions  $\mathcal{P}$  is a tuple  $\mathcal{P} \stackrel{\text{def}}{=} (S, l, <)$  where  $S$  is a set of elements called states,  $l : S \rightarrow 2^{\mathcal{P}}$  is a mapping which assigns to each state  $s$  a single world  $m \in 2^{\mathcal{P}}$  and  $<$  is a *strict partial* order on  $S$  satisfying smoothness condition. Intuitively, the states that are lower down in the ordering are more plausible, normal or in a general case preferred, than those that are higher up. A statement of the form  $\alpha \sim \beta$  holds in a preferential model iff the minimal  $\alpha$ -states are also  $\beta$ -states.

### 3 Preferential LTL

In this paper, we introduce a new formalism for reasoning about time that is able to distinguish between normal and exceptional points of time. We do so by investigating a defeasible extension of *LTL* with a preferential semantics. The following example introduces a case scenario we shall be using in the remainder of this section, with the purpose of giving a motivation for this formalism and better illustrating the definitions in what follows.

► **Example 1.** We have a computer program in which the values of its variables change with time. In particular, the agent wants to check two parameters, say  $x$  and  $y$ . These two variables take one and only one value between 1 and 3 on each iteration of the program. We represent the set of atomic propositions by  $\mathcal{P} = \{x_1, x_2, x_3, y_1, y_2, y_3\}$  where  $x_i$  (resp.  $y_i$ ) for all  $i \in \{1, 2, 3\}$  is true iff the variable  $x$  (resp.  $y$ ) has the value  $i$  in a current iteration. Figure 1 depicts a temporal interpretation corresponding to a possible behaviour of such a program:



■ **Figure 1** LTL interpretation  $V$  (for  $t > 5$ ,  $V(t) = V(5) = \{x_2, y_3\}$ ).

Under normal circumstances, the program assigns the value 3 to  $y$  whenever  $x = 2$ . We can express this fact using classical LTL as follows:  $\Box(x_2 \rightarrow y_3)$ , with  $x_2 \rightarrow y_3$  is defined by  $\neg x_2 \vee y_3$ . Nevertheless, the agent notices that there is one exceptional iteration (Iteration 3) where the program assigns the value 1 to  $y$  when  $x = 2$ .

Some might consider that the current program is defective at some points of time. In LTL, the statement  $\Box(x_2 \rightarrow y_3) \wedge \Diamond(x_2 \wedge y_1)$  will always be false, since  $y$  cannot have two different values in an iteration where  $x = 2$ . Nonetheless we want to propose a logical framework that is exception tolerant for reasoning about a system's behaviour. In order to express this general tendency ( $x_2 \rightarrow y_3$ ) while taking into account that there might be some exceptional iterations that are expected.



### 3.1 Introducing defeasible temporal operators

Britz & Varzinczak [5] introduced new modal operators called defeasible modalities. In their setting, defeasible operators, unlike their classical counterparts, are able to single out normal worlds from those that are less normal or exceptional in the reasoner's mind. Here we extend the vocabulary of classical LTL with the *defeasible temporal operators*  $\boxminus$  and  $\diamond$ . Sentences of the resulting logic  $LTL^\sim$  are built up according to the following grammar:

$$\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \Box\alpha \mid \diamond\alpha \mid \bigcirc\alpha \mid \alpha\mathcal{U}\alpha \mid \boxminus\alpha \mid \diamond\alpha$$

The intuition behind these new operators is the following:  $\boxminus$  reads as *defeasible always* and  $\diamond$  reads as *defeasible eventually*.

► **Example 2.** Going back to our example 1, we can describe the normal behaviour of the program using the statement  $\boxminus(x_2 \rightarrow y_3) \wedge \diamond(x_2 \wedge y_1)$ . In all normal future time points, the program assigns the value 3 to  $y$  when  $x = 2$ . Although unlikely, there are some exceptional time points in the future where  $x = 2$  and  $y = 1$ . But those are 'ignored' by the defeasible always operator.

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  $\alpha \in \mathcal{L}$  is a sentence of  $\mathcal{L}^\sim$ . We denote a subset of our language that contains only Boolean connectives, the two defeasible operators  $\boxminus$ ,  $\diamond$  and their classical counterparts by  $\mathcal{L}^*$ . Next we shall discuss how to interpret statements that have this defeasible aspect and how to determine the truth values of each well-formed sentence in  $\mathcal{L}^\sim$ .

### 3.2 Preferential semantics

First of all, in order to interpret the sentences of  $\mathcal{L}^\sim$  we consider, as stated on the preliminaries,  $(\mathbb{N}, <)$  to be a temporal structure. Hence, a temporal interpretation that associates each time point  $t$  with a truth assignment of all propositional atoms.

The preferential component of the interpretation of our language is directly inspired by the preferential semantics proposed by Shoham [17] and used in the KLM approach [12]. The preference relation  $\prec$  is a strict partial order on our points of time. Following Kraus et al. [12],  $t \prec t'$  means that  $t$  is more preferred than  $t'$ . The reasoner has now the tools to express the preference between points of time by comparing them w.r.t. each other, with time points lower down the order being more preferred than those higher up.

► **Definition 3.** 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  $\min_{\prec}(N) \stackrel{\text{def}}{=} \{t \in N \mid \text{there is no } t' \in N \text{ such that } t' \prec t\}$ .

► **Definition 4 (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  $\emptyset \neq N \subseteq \mathbb{N}$ .

► **Definition 5 (Preferential temporal interpretation).** An  $LTL^\sim$  interpretation on a set of propositional atoms  $\mathcal{P}$ , also called preferential temporal interpretation on  $\mathcal{P}$ , is a pair  $I \stackrel{\text{def}}{=} (V, \prec)$  where  $V$  is a temporal interpretation on  $\mathcal{P}$ , 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  $\mathcal{I}$ .

In what follows, given a preference 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  $\min_{\prec}(t)$ . It is also worth to point out that given a preferential interpretation  $I = (V, \prec)$  and  $\mathbb{N}$ , the set  $\min_{\prec}(t)$  is always a non-empty subset of  $[t, +\infty[$  at any time point  $t \in \mathbb{N}$ .

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 interpretation, and let  $t$  be a time point in  $I$  in  $\mathbb{N}$ . Satisfaction of  $\alpha$  at  $t$  in  $I$ , denoted  $I, t \models \alpha$ , is defined as follows:

- $I, t \models \Box\alpha$  if  $I, t' \models \alpha$  for all  $t' \in \min_{\prec}(t)$ ;
- $I, t \models \Diamond\alpha$  if  $I, t' \models \alpha$  for some  $t' \in \min_{\prec}(t)$ .

The truth values of Boolean connectives and classical modalities are defined as in *LTL*. The intuition behind a sentence like  $\Box\alpha$  is that  $\alpha$  holds in all preferred time points that come after  $t$ .  $\Diamond\alpha$  intuitively means that  $\alpha$  holds on at least one preferred time point relative in the future of  $t$ .

We say  $\alpha \in \mathcal{L}^\sim$  is *preferentially 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 *preferentially satisfiable* iff there is a preferential temporal interpretation  $I$  s.t.  $I, 0 \models \alpha$ . A sentence  $\alpha \in \mathcal{L}^\sim$  is *valid* (denoted by  $\models \alpha$ ) iff for all temporal interpretation  $I$  and time points  $t$  in  $\mathbb{N}$ , we have  $I, t \models \alpha$ .

► **Example 6.** Going back to Example 1, we can see that the time points 5 and 1 are more “normal” than iteration 3. By adding preferential preference  $\prec := \{(5, 3), (1, 3)\}$ , we denote the preferential temporal interpretation by  $I = (V, \prec)$ . We have that  $I, 0 \not\models \Box(x_2 \rightarrow y_3) \wedge \Diamond(x_2 \wedge y_1)$  and  $I, 0 \models \Box(x_2 \rightarrow y_3) \wedge \Diamond(x_2 \wedge y_1)$ .

We can see that the addition of  $\prec$  relation preserves the truth values of all classical temporal sentences. Moreover, for every  $\alpha \in \mathcal{L}$ , we have that  $\alpha$  is satisfiable in *LTL* if and only if  $\alpha$  is preferentially satisfiable in *LTL*<sup>~</sup>.

We discuss some properties of these defeasible modalities next. In what follows, let  $\alpha, \beta$  be well-formed sentences in  $\mathcal{L}^\sim$ . We have duality between our defeasible operators:  $\models \Box\alpha \leftrightarrow \neg \Diamond\neg\alpha$ . We also have  $\models \Box\alpha \rightarrow \Box\alpha$  and  $\models \Diamond\alpha \rightarrow \Diamond\alpha$ . Intuitively, This property states that if a statement holds in all of future time points of any given point of time  $t$ , it holds on all our *future preferred* time points. As intended, this property establishes the defeasible always as “weaker” than the classical always. It can commonly be accepted since the set of all preferred future states are in the future. This is why we named  $\Box$  *defeasible always*. On the other hand, we see that  $\Diamond$  is “stronger” than classical eventually, the statement within  $\Diamond$  holds at a preferable future.

The axiom of distributivity (K) can be stated in terms of our defeasible operators. We can also verify the validity of these two statements  $\models \Box(\alpha \wedge \beta) \leftrightarrow (\Box\alpha \wedge \Box\beta)$  and  $\models (\Box\alpha \vee \Box\beta) \rightarrow \Box(\alpha \vee \beta)$ , the converse of the second statement is not always true.

The reflexivity axiom (T) for the classical operators does not hold in the case of defeasible modalities. We can easily find an interpretation  $I = (V, \prec)$  where  $I, t \not\models \Box\alpha \rightarrow \alpha$ . Indeed, since we can have  $t \notin \min_{\prec}(t)$  for a temporal point  $t$ , we can have  $I, t \models \Box\alpha$  and  $I, t \models \neg\alpha$ .

One thing worth pointing out is the set of future preferred time points changes dynamically as we move forward in time. Given three time points  $t_1 \leq t_2 \leq t_3$ ,  $t_3 \notin \min_{\prec}(t_1)$  whilst  $t_3 \in \min_{\prec}(t_2)$  could be true in some cases. Hence, if  $I, t \models \Box\Box\alpha$  does not imply that for all  $t' \in \min_{\prec}(t)$ ,  $I, t' \models \Box\alpha$ . Therefore, the transitivity axiom (4) does not hold also in our defeasible modalities. On the other hand, given those three time points,  $t_3 \notin \min_{\prec}(t_1)$  implies that  $t_3 \notin \min_{\prec}(t_2)$ .

### 3.3 State-dependent preferential interpretations

We define a class of well-behaved *LTL*<sup>~</sup> interpretations that are useful in the remainder of the paper.

## 19:6 On the Decidability of a Fragment of preferential LTL

► **Definition 7** (State-dependent preferential interpretations). *Let  $I = (V, \prec) \in \mathfrak{J}$ .  $I$  is state-dependent preferential interpretation iff for every  $i, j, i', j' \in \mathbb{N}$ , if  $V(i') = V(i)$  and  $V(j') = V(j)$ , then  $(i, j) \in \prec$  iff  $(i', j') \in \prec$ .*

In what follows,  $\mathfrak{J}^{sd}$  denotes the set of all state-dependent interpretations. The intuition behind setting up this restriction is to have a more compact form of expressing preference over time points. In a way, time points with similar valuations are considered to be identical with regards to  $\prec$ , they express the same preferences towards other time points. Moreover, we have some interesting properties that do not in the general case. In particular, we have the following property :

► **Proposition 8.** *Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$  and let  $i, i', j, j' \in \mathbb{N}$  s.t.  $i \leq i'$ ,  $i' \leq j'$  and  $j \in \min_{\prec}(i)$ . If  $V(j) = V(j')$ , then  $j' \in \min_{\prec}(i')$ .*

This property is specific to the class of state-dependent interpretations. However, the following proposition is true for every  $I \in \mathfrak{J}$ .

► **Proposition 9.** *Let  $I = (V, \prec) \in \mathfrak{J}$  and let  $i, j \in \mathbb{N}$  s.t.  $j \in \min_{\prec}(i)$ . For all  $i \leq i' \leq j$ , we have  $j \in \min_{\prec}(i')$ .*

### 4 A useful representation of preferential structures

One of the objectives of this paper is to establish some computational properties about the satisfiability problem. In order to do this, we introduce into the sequel different structures inspired by the approach followed by Sistla and Clarke in [18]. They observe that in every *LTL* interpretation, there is a time point  $t$  after which every  $t$ -successor's valuation occurs infinitely many times. This is an obvious consequence of having an infinite set of time points and a finite number of possible valuations. That is the case also for *LTL*<sup>~</sup> interpretations.

► **Lemma 10.** *Let  $I = (V, \prec) \in \mathfrak{J}$ . There exists a  $t \in \mathbb{N}$  s.t. for all  $l \in [t, +\infty[$ , there is a  $k > l$  where  $V(l) = V(k)$ .*

For an interpretation  $I \in \mathfrak{J}$ , we denote the first time point where the condition set in Lemma 10 is satisfied by  $\mathfrak{t}_I$ . We can split each temporal structure into two intervals: an initial and a final part.

► **Definition 11.** *Let  $I = (V, \prec) \in \mathfrak{J}$ . We define:  $init(I) \stackrel{\text{def}}{=} [0, \mathfrak{t}_I[$ ;  $final(I) \stackrel{\text{def}}{=} [\mathfrak{t}_I, +\infty[$ ;  $range(I) \stackrel{\text{def}}{=} \{V(i) \mid i \in final(I)\}$ ;  $val(I) \stackrel{\text{def}}{=} \{V(i) \mid i \in \mathbb{N}\}$ ;  $size(I) \stackrel{\text{def}}{=} length(init(I)) + card(range(I))$ , where  $length(\cdot)$  denotes the length of a sequence and  $card(\cdot)$  set cardinality.*

In the size of  $I$  we count the number of time points in the initial part and the number of valuations contained in the final part. In what follows, we discuss some properties concerning these notions and state dependent interpretations.

► **Proposition 12.** *Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$  and let  $i \leq j \leq i' \leq j'$  be time points in  $final(I)$  s.t.  $V(j) = V(j')$ . Then we have  $j \in \min_{\prec}(i)$  iff  $j' \in \min_{\prec}(i')$ .*

► **Lemma 13.** *Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$  and  $i \leq i'$  be time points of  $final(I)$  where  $V(i) = V(i')$ . Then for every  $\alpha \in \mathcal{L}^*$ , we have  $I, i \models \alpha$  iff  $I, i' \models \alpha$ .*

► **Definition 14** (Faithful Interpretations). *Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$ ,  $I' = (V', \prec') \in \mathfrak{J}^{sd}$  be two interpretations over the same set of atoms  $\mathcal{P}$ . We say that  $I, I'$  are faithful interpretations if  $val(I) = val(I')$  and, for all  $i, j, i', j' \in \mathbb{N}$  s.t.  $V'(i') = V(i)$  and  $V'(j') = V(j)$ , we have  $(i, j) \in \prec$  iff  $(i', j') \in \prec'$ .*

Throughout this paper, we write  $init(I) \doteq init(I')$  as shorthand for the condition that states:  $length(init(I)) = length(init(I'))$  and for each  $i \in init(I)$  we have  $V(i) = V'(i)$ .

► **Lemma 15.** *Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$ ,  $I' = (V', \prec') \in \mathfrak{J}^{sd}$  be two faithful interpretations over  $\mathcal{P}$  such that  $V'(0) = V(0)$  (in case  $init(I)$  is empty),  $init(I) \doteq init(I')$ , and  $range(I) = range(I')$ . Then for all  $\alpha \in \mathcal{L}^*$ , we have that  $I, 0 \models \alpha$  iff  $I', 0 \models \alpha$ .*

Lemma 15 implies that the ordering of time points in  $final(\cdot)$  does not matter, and what matters is the  $range(\cdot)$  of valuations contained within it. It is worth to mention that Lemma 13 and 15 hold only in the case interpretations in  $\mathfrak{J}^{sd}$  and they are not always true in the general case.

Sistla & Clarke [18] introduced the notion of acceptable sequences. The general purpose behind it is the ability to build, from an initial interpretation, other interpretations. We adapt this notion for preferential temporal structures. We then introduce the notion of pseudo-interpretations that will come in handy in showing decidability of the satisfiability problem in  $\mathcal{L}^*$  in the upcoming section.

In the sequel, the term temporal sequence or sequence in short, will denote a sequence of ordered integer numbers. A sequence allows to represent a set of time points. Sometimes, we will consider integer intervals as sequences. Moreover, given two sequences  $N_1, N_2$ , the union of  $N_1$  and  $N_2$ , denoted by  $N_1 \cup N_2$ , is the sequence containing only elements of  $N_1$  and  $N_2$ . An acceptable sequence is a temporal sequence that is built relatively to a preferential temporal interpretation  $I$  as follows:

► **Definition 16** (Acceptable sequence w.r.t.  $I$ ). *Let  $I = (V, \prec) \in \mathfrak{J}$  and  $N$  be a sequence of temporal time points.  $N$  is an acceptable sequence w.r.t.  $I$  iff for all  $i \in N \cap final(I)$  and for all  $j \in final(I)$  s.t.  $V(i) = V(j)$ , we have  $j \in N$ .*

The particularity we are looking for is that any picked time point in  $init(\cdot)$  (resp.  $final(\cdot)$ ) will remain in the initial (resp. final) part of the new interpretation. It is worth pointing out that an acceptable sequence w.r.t. a preferential temporal interpretation can be either finite or infinite. Moreover,  $\mathbb{N}$  is an acceptable sequence w.r.t. any interpretation  $I \in \mathfrak{J}$ . The purpose behind the notion of acceptable sequence is to construct new interpretations starting from an  $LTL^\sim$  interpretation.

Given  $N$  an acceptable sequence w.r.t.  $I$ , if  $N$  has a time point  $t$  in  $final(I)$ , then all time points  $t'$  that have the same valuation as  $t$  must be in  $N$ . Thus, we have an infinite sequence of time points. As such, we can define an initial part and a final part, in a similar way as  $LTL^\sim$  interpretations. We let  $init(I, N)$  be the largest subsequence of  $N$  that is a subsequence of  $init(I)$ . Note that if  $N$  does not contain any time point of  $final(I)$ , then  $N$  is finite.

► **Definition 17.** *Let  $I = (V, \prec) \in \mathfrak{J}$ , and let  $N$  be an acceptable sequence w.r.t.  $I$ . We define:  $init(I, N) \stackrel{\text{def}}{=} N \cap init(I)$ ;  $final(I, N) \stackrel{\text{def}}{=} N \setminus init(I, N)$ ;  $range(I, N) \stackrel{\text{def}}{=} \{V(t) \mid t \in final(I, N)\}$ ;  $val(I, N) \stackrel{\text{def}}{=} \{V(t) \mid t \in N\}$ ;  $size(I, N) \stackrel{\text{def}}{=} length(init(I, N)) + card(range(I, N))$ .*

It is worth mentioning that, thanks to Definition 16, given an acceptable sequence w.r.t.  $I$ , we have  $size(I, N) \leq size(I)$ .

► **Definition 18** (Pseudo-interpretation over  $N$ ). *Let  $I = (V, \prec) \in \mathfrak{J}$  and  $N$  be an acceptable sequence w.r.t.  $I$ . The pseudo-interpretation over  $N$  is the tuple  $I^N \stackrel{\text{def}}{=} (N, V^N, \prec^N)$  where:*

- $V^N : N \longrightarrow 2^{\mathcal{P}}$  is a valuation function over  $N$ , where for all  $i \in N$ , we have  $V^N(i) = V(i)$ ,
- $\prec^N \subseteq N \times N$ , where for all  $(i, j) \in N^2$ , we have  $(i, j) \in \prec^N$  iff  $(i, j) \in \prec$ .

## 19:8 On the Decidability of a Fragment of preferential LTL

The truth values of  $\mathcal{L}^*$  sentences in pseudo-interpretations are defined in a similar fashion as for preferential temporal interpretations. With  $\models_{\mathcal{P}}$  we denote the truth values of sentences in a pseudo-interpretation. We highlight truth values for classical and defeasible modalities.

- $I^N, t \models_{\mathcal{P}} \Box \alpha$  if  $I^N, t' \models_{\mathcal{P}} \alpha$  for all  $t' \in N$  s.t.  $t' \geq t$ ;
- $I^N, t \models_{\mathcal{P}} \Diamond \alpha$  if  $I^N, t' \models_{\mathcal{P}} \alpha$  for some  $t' \in N$  s.t.  $t' \geq t$ ;
- $I^N, t \models_{\mathcal{P}} \Box \alpha$  if for all  $t' \in N$  s.t.  $t' \in \min_{\prec^N}(t)$ , we have  $I^N, t' \models_{\mathcal{P}} \alpha$ ;
- $I^N, t \models_{\mathcal{P}} \Diamond \alpha$  if  $I^N, t' \models_{\mathcal{P}} \alpha$  for some  $t' \in N$  s.t.  $t' \in \min_{\prec^N}(t)$ .

► **Proposition 19.** *Let  $I = (V, \prec) \in \mathfrak{I}$ ,  $N_1, N_2$  be two acceptable sequences w.r.t.  $I$ . Then  $N_1 \cup N_2$  is an acceptable sequence w.r.t.  $I$  s.t.  $\text{size}(I, N_1 \cup N_2) \leq \text{size}(I, N_1) + \text{size}(I, N_2)$ .*

► **Proposition 20.** *Let  $I = (V, \prec) \in \mathfrak{I}$  and  $N$  be an acceptable sequence w.r.t.  $I$ . If for all distinct  $t, t' \in N$ , we have  $V(t') = V(t)$  only when both  $t, t' \in \text{final}(I, N)$ , then  $\text{size}(I, N) \leq 2^{|\mathcal{P}|}$ .*

### 5 Bounded-model property

The main contribution of this paper is to establish certain computational properties regarding the satisfiability problem in  $\mathcal{L}^*$ . The algorithmic problem is as follows: Given an input sentence  $\alpha \in \mathcal{L}^*$ , decide whether  $\alpha$  is preferentially satisfiable. In this section, we show that this problem is decidable.

The proof is based on the one given by Sistla and Clarke to show the complexity of propositional linear temporal logic [18]. Let  $\mathcal{L}^*$  be the fragment of  $\mathcal{L}^{\sim}$  that contains only Boolean connectives and temporal operators ( $\Box, \Box, \Diamond, \Diamond$ ). Let  $\alpha \in \mathcal{L}^*$ , with  $|\alpha|$  we denote the number of symbols within  $\alpha$ . The main result of the present paper is summarized in the following theorem, of which the proof will be given in the remainder of the section.

► **Theorem 21 (Bounded-model property).** *If  $\alpha \in \mathcal{L}^*$  is  $\mathfrak{I}^{sd}$ -satisfiable, then we can find an interpretation  $I \in \mathfrak{I}^{sd}$  such that  $I, 0 \models \alpha$  and  $\text{size}(I) \leq |\alpha| \times 2^{|\mathcal{P}|}$ .*

Hence, given a satisfiable sentence  $\alpha \in \mathcal{L}^*$ , there is an interpretation satisfying  $\alpha$  of which the size is bounded. Since  $\alpha$  is  $\mathfrak{I}^{sd}$ -satisfiable, we know  $I, 0 \models \alpha$ . From  $I$  we can construct an interpretation  $I'$  also satisfying  $\alpha$ , i.e.,  $I', 0 \models \alpha$ , which is bounded on its size by  $|\alpha| \times 2^{|\mathcal{P}|}$ . The goal of this section is to show how to build said bounded interpretation. Let  $\alpha \in \mathcal{L}^*$  and let  $I \in \mathfrak{I}^{sd}$  be s.t.  $I, 0 \models \alpha$ . The first step is to characterize an acceptable sequence  $N$  w.r.t.  $I$  such that  $N$  is bounded first of all, and “keeps” the satisfiability of the sub-sentences  $\alpha_1$  of  $\alpha$  i.e., if  $I, t \models \alpha_1$ , then  $I^N, t \models_{\mathcal{P}} \alpha_1$  (see Definition 18). We do so by building a bounded pseudo-interpretation step by step by selecting what to take from the initial interpretation  $I$  for each sub-sentence  $\alpha_1$  contained in  $\alpha$  to be satisfied. In what follows, we introduce *Anchors*( $\cdot$ ) as a strategy for picking out the desired time points.

► **Definition 22 (Acceptable sequence transformation).** *Let  $I = (V, \prec) \in \mathfrak{I}$  and let  $N$  be a sequence of time points. Let  $N'$  be the sequence of all time points  $t'$  in  $\text{final}(I)$  for which there is  $t \in N \cap \text{final}(I)$  with  $V(t') = V(t)$ . With  $AS(I, N) \stackrel{\text{def}}{=} N \cup N'$  we denote the acceptable sequence transformation of  $N$  w.r.t.  $I$ .*

The sequence  $AS(I, N)$  is the acceptable sequence transformation of  $N$  w.r.t.  $I$ . In the previous definition,  $N'$  is the sequence of all time points  $t'$  having the same valuation as some time point  $t \in N$  that is in  $\text{final}(I)$ . It is also worth to point out that  $N'$  can be empty in the case of there being no time point  $t \in N$  that is in  $\text{final}(I)$ .  $N$  is then a finite acceptable sequence w.r.t.  $I$  where  $AS(I, N) = N$ . This notation is mainly used to ensure that we are using the acceptable version of any sequence.

► **Definition 23** (Chosen occurrence w.r.t.  $\alpha$ ). *Let  $I = (V, \prec) \in \mathfrak{I}$ ,  $\alpha \in \mathcal{L}^\sim$  and  $N$  be an acceptable sequence w.r.t.  $I$  s.t. there exists a time point  $t$  in  $N$  with  $I, t \models \alpha$ . The chosen occurrence satisfying  $\alpha$  in  $N$ , denoted by  $\mathfrak{t}_\alpha^{I,N}$ , is defined as follows:*

$$\mathfrak{t}_\alpha^{I,N} \stackrel{\text{def}}{=} \begin{cases} \min_{<} \{t \in \text{final}(I, N) \mid I, t \models \alpha\}, & \text{if } \{t \in \text{final}(I, N) \mid I, t \models \alpha\} \neq \emptyset \\ \max_{<} \{t \in \text{init}(I, N) \mid I, t \models \alpha\}, & \text{otherwise.} \end{cases}$$

Notice that  $<$  above denotes the natural ordering of the underlying temporal structure. The strategy to pick out a time point satisfying a given sentence  $\alpha$  in  $N$  is as follows. If said sentence is in the final part, we pick the first time point that satisfies it, since we have the guarantee to find infinitely many time points having the same valuations as  $\mathfrak{t}_\alpha^{I,N}$  that also satisfy  $\alpha$  (see Lemma 13). If not, we pick the last occurrence in the initial part that satisfies  $\alpha$ . Thanks to Definition 23, we can limit the number of time points taken that satisfy the same sentence.

► **Definition 24** (Selected time points). *Let  $I = (V, \prec) \in \mathfrak{I}$ ,  $N$  be an acceptable sequence w.r.t.  $I$  and  $\alpha \in \mathcal{L}^\sim$  s.t. there is  $t$  in  $N$  s.t.  $I, t \models \alpha$ . With  $ST(I, N, \alpha) \stackrel{\text{def}}{=} AS(I, (\mathfrak{t}_\alpha^{I,N}))$  we denote the selected time points of  $N$  and  $\alpha$  w.r.t.  $I$ . (Note that  $(\mathfrak{t}_\alpha^{I,N})$  is a sequence of only one element.)*

Given a sentence  $\alpha \in \mathcal{L}^\sim$  and an acceptable sequence  $N$  w.r.t.  $I$  s.t. there is at least one time point  $t$  where  $I, t \models \alpha$ , the sequence  $ST(I, N, \alpha)$  is the acceptable sequence transformation of the sequence  $(\mathfrak{t}_\alpha^{I,N})$ . If  $\mathfrak{t}_\alpha^{I,N} \in \text{init}(I)$ , the sequence  $ST(I, N, \alpha)$  is the sequence  $(\mathfrak{t}_\alpha^{I,N})$ . Otherwise, the sequence  $ST(I, N, \alpha)$  is the sequence of all time points  $t$  in  $\text{final}(I)$  that have the same valuation as  $\mathfrak{t}_\alpha^{I,N}$ . In both cases, we can see that  $\text{size}(I, ST(I, N, \alpha)) = 1$ .

Given an interpretation  $I = (V, \prec)$  and  $N$  an acceptable sequence w.r.t.  $I$ , the *representative sentence* of a valuation  $v$  is formally defined as  $\alpha_v \stackrel{\text{def}}{=} \bigwedge \{p \mid p \in v\} \wedge \bigwedge \{\neg p \mid p \notin v\}$ .

► **Definition 25** (Distinctive reduction). *Let  $I = (V, \prec) \in \mathfrak{I}$  and let  $N$  be an acceptable sequence w.r.t.  $I$ . With  $DR(I, N) \stackrel{\text{def}}{=} \bigcup_{v \in \text{val}(I, N)} ST(I, N, \alpha_v)$  we denote the distinctive reduction of  $N$ .*

Given an acceptable sequence  $N$  w.r.t.  $I$ ,  $DR(I, N)$  is the sequence containing the chosen occurrence  $\mathfrak{t}_{\alpha_v}^{I,N}$  that satisfies the representative  $\alpha_v$  in  $N$  for each  $v \in \text{val}(I, N)$ . In other words, we pick the selected time points for each possible valuation in  $\text{val}(I, N)$ . There are two interesting results with regard to  $DR(I, N)$ . The first one is that  $DR(I, N)$  is an acceptable sequence w.r.t.  $I$ . This can easily be proven since  $ST(I, N, \alpha_v)$  is also an acceptable sequence w.r.t.  $I$ , and the union of all  $ST(I, N, \alpha_v)$  is an acceptable sequence w.r.t.  $I$  (see Proposition 19). The second result is that  $\text{size}(I, DR(I, N)) \leq 2^{|\mathcal{P}|}$ . Indeed, thanks to Proposition 19, we can see that  $\text{size}(I, DR(I, N)) \leq \sum_{v \in \text{val}(I, N)} \text{size}(ST(I, N, \alpha_v))$ . Moreover, we have  $\text{size}(I, ST(I, N, \alpha_v)) = 1$  for each  $v \in \text{val}(I, N)$ . On the other hand, there are at most  $2^{|\mathcal{P}|}$  possible valuations in  $\text{val}(I, N)$ . Thus, we can assert that  $\sum_{v \in \text{val}(I, N)} \text{size}(I, ST(I, N, \alpha_v)) \leq 2^{|\mathcal{P}|}$ , and then we have  $\text{size}(I, DR(I, N)) \leq 2^{|\mathcal{P}|}$ .

► **Definition 26** (Anchors). *Let a sentence  $\alpha \in \mathcal{L}^\star$  starting with a temporal operator, let  $I = (V, \prec) \in \mathfrak{I}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . The sequence  $\text{Anchors}(I, T, \alpha)$  is defined as: Let  $\alpha_1 \in \mathcal{L}^\star$ .*

$$\begin{aligned} \text{Anchors}(I, T, \Diamond \alpha_1) &\stackrel{\text{def}}{=} ST(I, \mathbb{N}, \alpha_1); \\ \text{Anchors}(I, T, \Box \alpha_1) &\stackrel{\text{def}}{=} \emptyset; \\ \text{Anchors}(I, T, \heartsuit \alpha_1) &\stackrel{\text{def}}{=} \bigcup_{t \in T} ST(I, AS(I, \min_{\prec}(t)), \alpha_1); \\ \text{Anchors}(I, T, \boxtimes \alpha_1) &\stackrel{\text{def}}{=} DR(I, \bigcup_{t \in T} AS(I, \min_{\prec}(t))). \end{aligned}$$



## 19:10 On the Decidability of a Fragment of preferential LTL

Given an acceptable sequence  $T$  w.r.t.  $I \in \mathcal{J}^{sd}$  where all of its time points satisfy  $\alpha$ , where  $\alpha$  is a sentence starting with a temporal operator,  $Anchors(I, T, \alpha)$  is an acceptable sequence w.r.t.  $I$ . This is due thanks to the notion of selected time points and distinctive reduction (see Definition 24 and 25).  $Anchors(I, T, \alpha)$  contains the selected time points satisfying the sub-sentence  $\alpha_1$  of  $\alpha$  (except for  $\Box\alpha_1$ ). Our goal is to have the selected time points that satisfy  $\alpha_1$  for each  $t \in T$ .

It is worth to point out that the choice of  $Anchors(I, T, \Box\alpha_1) = \emptyset$  is due to the fact  $\alpha_1$  is satisfied starting from the first time  $t_0 \in T$  i.e., for all  $t \geq t_0$ , we have  $I, t \models \alpha$ . So no matter what time point  $t$  we pick after  $t_0$ , we have  $I, t \models \alpha_1$ . On the other hand, by the nature of the semantics of  $\Box\alpha_1$ , all  $t \in \bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i))$  satisfy  $\alpha_1$ . The acceptable sequence  $Anchors(I, T, \Box\alpha_1)$  contains only the selected time points for each distinct valuation in  $\bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i))$ .

► **Lemma 27.** *Let  $\alpha_1 \in \mathcal{L}^*$  be a sentence starting with a temporal operator,  $I = (V, \prec) \in \mathcal{J}^{sd}$  and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  where for all  $t \in T$  we have  $I, t \models \diamond\alpha_1$ . Then for all  $t, t' \in Anchors(I, T, \diamond\alpha_1)$  s.t.  $V(t) = V(t')$  and  $t \neq t'$ , we have  $t, t' \in final(I, Anchors(I, T, \diamond\alpha_1))$ .*

► **Proposition 28.** *Let  $\alpha \in \mathcal{L}^*$  be a sentence starting with a temporal operator,  $I = (V, \prec) \in \mathcal{J}^{sd}$ . Let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  where for all  $t \in T$  we have  $I, t \models \alpha$ . Then, we have  $size(I, Anchors(I, T, \alpha)) \leq 2^{|P|}$ .*

► **Proposition 29.** *Let  $\alpha_1 \in \mathcal{L}^*$ ,  $I = (V, \prec) \in \mathcal{J}^{sd}$ , let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \Box\alpha_1$ , with  $\alpha_1 \in \mathcal{L}^*$ . For all acceptable sequences  $N$  w.r.t.  $I$  s.t.  $Anchors(I, T, \Box\alpha_1) \subseteq N$  and for all  $t_i \in N \cap T$ , we have the following: Let  $I^N = (V^N, \prec^N)$  be the pseudo-interpretation over  $N$  and  $t' \in N$ , if  $t' \notin \min_{\prec}(t_i)$ , then  $t' \notin \min_{\prec^N}(t_i)$ .*

The strategy of building  $Anchors(\cdot)$  is explained by the fact that we want to preserve the truth values of defeasible sub-sentences of  $\alpha$  in the bounded interpretation.

With  $Anchors(\cdot)$  defined, we introduce the notion of  $Keep(\cdot)$ . This function will help us compute recursively starting from the initial satisfiable sentence  $\alpha$  down to its literals, the selected time points to pick in order to build our pseudo-interpretation.

► **Definition 30 (Keep).** *Let  $\alpha \in \mathcal{L}^*$  be in NNF,  $I = (V, \prec) \in \mathcal{J}^{sd}$ , and let  $T$  be an acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . The sequence  $Keep(I, T, \alpha)$  is defined as  $\emptyset$ , if  $T = \emptyset$ ; otherwise it is recursively defined as follows:*

- $Keep(I, T, \ell) \stackrel{\text{def}}{=} \emptyset$ , where  $\ell$  is a literal;
- $Keep(I, T, \alpha_1 \wedge \alpha_2) \stackrel{\text{def}}{=} Keep(I, T, \alpha_1) \cup Keep(I, T, \alpha_2)$ ;
- $Keep(I, T, \alpha_1 \vee \alpha_2) \stackrel{\text{def}}{=} Keep(I, T_1, \alpha_1) \cup Keep(I, T_2, \alpha_2)$ , where  $T_1 \subseteq T$  (resp.  $T_2 \subseteq T$ ) is the sequence of all  $t_1 \in T$  (resp.  $t_2 \in T$ ) s.t.  $I, t_1 \models \alpha_1$  (resp.  $I, t_2 \models \alpha_2$ );
- $Keep(I, T, \diamond\alpha_1) \stackrel{\text{def}}{=} Anchors(I, T, \diamond\alpha_1) \cup Keep(I, Anchors(I, T, \diamond\alpha_1), \alpha_1)$ ;
- $Keep(I, T, \Box\alpha_1) \stackrel{\text{def}}{=} Keep(I, T, \alpha_1)$ ;
- $Keep(I, T, \diamond\alpha_1) \stackrel{\text{def}}{=} Anchors(I, T, \diamond\alpha_1) \cup Keep(I, Anchors(I, T, \diamond\alpha_1), \alpha_1)$ ;
- $Keep(I, T, \Box\alpha_1) \stackrel{\text{def}}{=} Anchors(I, T, \Box\alpha_1) \cup Keep(I, T', \alpha_1)$ , where  $T' = \bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i))$ .

With  $\mu(\alpha)$  we denote the number of classical and non-monotonic modalities in  $\alpha$ .

► **Proposition 31.** *Let  $\alpha \in \mathcal{L}^*$  be in NNF,  $I = (V, \prec) \in \mathcal{J}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . Then, we have  $size(I, Keep(I, T, \alpha)) \leq \mu(\alpha) \times 2^{|P|}$ .*

Given an acceptable sequence  $N$  w.r.t.  $I$ , we need to make sure when a time point  $t \in N$  in our acceptable sequence s.t.  $I, t \models \alpha$ , then  $I^N, t \models_{\neq} \alpha$ . The function  $Keep(I, T, \alpha)$  returns the acceptable sequence of time s.t. if  $Keep(I, T, \alpha) \subseteq N$  and  $t \in T$ , then said condition is met. We prove this in Lemma 32.

► **Lemma 32.** *Let  $\alpha \in \mathcal{L}^*$  be in NNF,  $I = (V, \prec) \in \mathfrak{J}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . For all acceptable sequences  $N$  w.r.t.  $I$ , if  $Keep(I, T, \alpha) \subseteq N$ , then for every  $t \in N \cap T$ , we have  $I^N, t \models_{\neq} \alpha$ .*

Since we build our pseudo-interpretation  $I^N$  by adding selected time points for each sub-sentence  $\alpha_1$  of  $\alpha$ , we need to make sure that said sub-sentence remains satisfied in  $I^N$ .

► **Definition 33** (Pseudo-interpretation transformation). *Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$  and let  $N$  be an infinite acceptable sequence w.r.t.  $I$ . The pseudo-interpretation  $I^N = (V^N, \prec^N)$  can be transformed into a preferential interpretation  $I' = (V', \prec') \in \mathfrak{J}^{sd}$  as follows:*

- for all  $i \geq 0$ , we have  $V'(i) = V^N(t_i)$ ;
- for all  $i, j \geq 0$ ,  $t_i, t_j \in N$ , we have  $(t_i, t_j) \in \prec^N$  iff  $(i, j) \in \prec'$ .

**Proof of Theorem 21.** We assume that  $\alpha \in \mathcal{L}^*$  is  $\mathfrak{J}^{sd}$ -satisfiable. The first thing we notice is that  $|\alpha| \geq \mu(\alpha) + 1$ . Let  $\alpha'$  be the NNF of the sentence  $\alpha$ . As a consequence of the duality rules of  $\mathcal{L}^*$ , we can deduce that  $\mu(\alpha') = \mu(\alpha)$ . Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$  s.t.  $I, 0 \models \alpha'$ . Let  $T_0 = AS(I, (0))$  be an acceptable sequence w.r.t.  $I$ . We can see that  $size(I, T_0) = 1$ . Since for all  $t \in T_0$  we have  $I, t \models \alpha'$  (see Lemma 13), we can compute recursively  $U = Keep(I, T_0, \alpha')$ . Thanks to Proposition 31, we conclude that  $U$  is an acceptable sequence w.r.t.  $I$  s.t.  $size(I, U) \leq \mu(\alpha') \times 2^{|\mathcal{P}|}$ . Let  $N = T_0 \cup U$  be the union of  $T_0$  and  $U$  and let  $I^N = (N, V^N, \prec^N)$  be its pseudo-interpretation over  $N$ . Thanks to Proposition 19, we have  $size(I, N) \leq 1 + \mu(\alpha') \times 2^{|\mathcal{P}|}$ . Thanks to Lemma 32, since  $0 \in N \cap T_0$  and  $Keep(I, T_0, \alpha') \subseteq N$ , we have  $I^N, 0 \models_{\neq} \alpha'$ . In case  $N$  is finite, we replicate the last time point  $t_n$  infinitely many times. Notice that  $size(I, N)$  does not change if we replicate the last element. We can transform the pseudo interpretation  $I^N$  to  $I' \in \mathfrak{J}^{sd}$  by changing the labels of  $N$  into a sequence of natural numbers minding the order of time points in  $N$  (see Definition 33). We can see that  $size(I') = size(I, N)$  and  $I', 0 \models \alpha$ . Consequently, we have  $size(I') \leq 1 + \mu(\alpha') \times 2^{|\mathcal{P}|}$ . Hence, from a given interpretation  $I$  s.t.  $I, 0 \models \alpha$  we can build an interpretation  $I'$  s.t.  $I', 0 \models \alpha$  and  $size(I') \leq 1 + \mu(\alpha') \times 2^{|\mathcal{P}|}$ . Without loss of generality, we conclude that  $size(I') \leq |\alpha| \times 2^{|\mathcal{P}|}$ . ◀

## 6 The satisfiability problem in $\mathcal{L}^*$

We now provide an algorithm allowing to decide whether a sentence  $\alpha \in \mathcal{L}^*$  is  $\mathfrak{J}^{sd}$ -satisfiable or not. For this purpose, first we focus on particular interpretations of the class  $\mathfrak{J}^{sd}$ , namely the ultimately periodic interpretations (UPI in short), and a finite representation of these interpretations, called ultimately periodic pseudo-interpretation (UPPI in short). As we will see in the second part of this section, to decide the  $\mathfrak{J}^{sd}$ -satisfiability of a sentence  $\alpha \in \mathcal{L}^*$ , the proposed algorithm guesses a bounded UPPI in a first step. Then, it checks the satisfiability of  $\alpha$  by the UPI of the guessed UPPI.

► **Definition 34** (UPI). *Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$  and let  $\pi = card(range(I))$ . We say  $I$  is an ultimately periodic interpretation if:*

- for every  $t, t' \in [t_I, t_I + \pi[$  s.t.  $t \neq t'$  (see Definition 10), we have  $V(t) \neq V(t')$ ,
- for every  $t \in [t_I, +\infty[$ , we have  $V(t) = V(t_I + (t - t_I) \bmod \pi)$ .



## 19:12 On the Decidability of a Fragment of preferential LTL

A UPI  $I$  is a state dependent interpretation s.t. each time point's valuation in  $final(I)$  is replicated periodically. Given a UPI,  $\pi = card(range(I))$  denotes the length of the period and the interval  $[t_I, t_I + \pi[$  is the first period which is replicated periodically throughout the final part. It is worth pointing out that for every  $t \in final(I)$ , we have  $V(t) \in \{V(t') \mid t' \in [t_I, t_I + \pi[$ , which is one of the consequences of the definition above. Thanks to Lemma 15, we can prove the following proposition.

► **Proposition 35.** *Let  $\mathcal{P}$  be a set of atomic propositions,  $I = (V, \prec) \in \mathcal{I}^{sd}$ ,  $i = length(initWith(I))$  and  $\pi = card(range(I))$ . There exists an ultimately periodic interpretation  $I' = (V', \prec') \in \mathcal{I}^{sd}$  s.t.  $I, I'$  are faithful interpretations over  $\mathcal{P}$  (Definition 14),  $initWith(I') \doteq initWith(I)$ ,  $range(I') = range(I)$  and  $V'(0) = V(0)$ . Moreover, for all  $\alpha \in \mathcal{L}^*$ , we have  $I, 0 \models \alpha$  iff  $I', 0 \models \alpha$ .*

It is worth to point out that the size of an interpretation and that of its UPI counterparts are equal. It can easily be seen that these interpretations have the same initial part and the same range of valuations in the final part. We can see that  $I$  and  $I'$  are faithful and that  $initWith(I') \doteq initWith(I)$ ,  $range(I') = range(I)$  and  $V'(0) = V(0)$ . Therefore,  $I$  and  $I'$  satisfy the same sentences.

► **Definition 36 (UPPI).** *A model structure is a tuple  $M = (i, \pi, V_M, \prec_M)$  where:  $i, \pi$  are two integers such that  $i \geq 0$  and  $\pi > 0$  (where  $i$  is intended to be the starting point of the period,  $\pi$  is the length of the period);  $V_M : [0, i + \pi[ \rightarrow 2^{\mathcal{P}}$ , and  $\prec_M \subseteq 2^{\mathcal{P}} \times 2^{\mathcal{P}}$  is a strict partial order. Moreover, (I) for all  $t \in [i, i + \pi[$ , we have  $V_M(t) \neq V_M(i - 1)$ ; and (II) for all distinct  $t, t' \in [i, i + \pi[$ , we have  $V_M(t) \neq V_M(t')$ .*

The reason behind setting properties (I) and (II) is that we can build a UPPI from a UPI, and back. Given a UPPI  $M = (i, \pi, V_M, \prec_M)$ , we define the *size of  $M$*  by  $size(M) \stackrel{\text{def}}{=} i + \pi$ . From a UPPI we define a UPI in the following way:

► **Definition 37.** *Given a UPPI  $M = (i, \pi, V_M, \prec_M)$ , let  $I(M) \stackrel{\text{def}}{=} (V, \prec)$ , where for every  $t \geq 0$ ,  $V(t) \stackrel{\text{def}}{=} V_M(t)$ , if  $t < i$ , and  $V(t) \stackrel{\text{def}}{=} V_M(i + (t - i) \bmod \pi)$ , otherwise, and  $\prec \stackrel{\text{def}}{=} \{(t, t') \mid (V(t), V(t')) \in \prec_M\}$ .*

Given a UPPI  $M = (i, \pi, V_M, \prec_M)$ , the interval  $[0, i[$  of a UPPI corresponds to the initial temporal part of the underlying interpretation  $I(M)$  and  $[i, i + \pi[$  represents a temporal period that is infinitely replicated in order to determine the final temporal part of the interpretation.

► **Definition 38 (UPPI's preferred time points).** *Let  $M = (i, \pi, V_M, \prec_M)$  be a UPPI and a time point  $t \in [0, i + \pi[$ . The set of preferred time points of  $t$  w.r.t.  $M$ , denoted by  $min_{\prec_M}(t)$ , is defined as follows:  $min_{\prec_M}(t) \stackrel{\text{def}}{=} \{t' \in [min_{\prec}\{t, i\}, i + \pi[ \mid \text{there is no } t'' \in [min_{\prec}\{t, i\}, i + \pi[ \text{ with } (V_M(t''), V_M(t')) \in \prec_M\}$ .*

► **Proposition 39.** *Let  $M = (i, \pi, V_M, \prec_M)$  be a UPPI,  $I(M) = (V, \prec)$  and  $t, t', t_M, t'_M \in \mathbb{N}$  s.t.:*

$$t_M = \begin{cases} t, & \text{if } t < i; \\ i + (t - i) \bmod \pi, & \text{otherwise.} \end{cases} \quad t'_M = \begin{cases} t', & \text{if } t' < i; \\ i + (t' - i) \bmod \pi, & \text{otherwise.} \end{cases}$$

*We have the following:  $t' \in min_{\prec}(t)$  iff  $t'_M \in min_{\prec_M}(t_M)$ .*

Now that UPPI is defined, we can move to the task of checking the satisfiability of a sentence  $\alpha$ . We define for a UPPI  $M = (i, \pi, V_M, \prec_M)$  and a sentence  $\alpha \in \mathcal{L}^*$  a labelling function  $lab_{\alpha}^M(\cdot)$  which associates a set of sub-sentences of  $\alpha$  to each  $t \in [0, i + \pi[$ .

► **Definition 40** (Labelling function). Let  $M = (i, \pi, V_M, \prec_M)$  be a UPPI,  $\alpha \in \mathcal{L}^*$ . The set of sub-sentences of  $\alpha$  for  $t \in [0, i + \pi[$ , denoted by  $lab_\alpha^M(t)$ , is defined as follows:

- $p \in lab_\alpha^M(t)$  iff  $p \in V_M(t)$ ;  $\neg\alpha_1 \in lab_\alpha^M(t)$  iff  $\alpha_1 \notin lab_\alpha^M(t)$ ;
- $\alpha_1 \wedge \alpha_2 \in lab_\alpha^M(t)$  iff  $\alpha_1, \alpha_2 \in lab_\alpha^M(t)$ ;  $\alpha_1 \vee \alpha_2 \in lab_\alpha^M(t)$  iff  $\alpha_1 \in lab_\alpha^M(t)$  or  $\alpha_2 \in lab_\alpha^M(t)$ ;
- $\diamond\alpha_1 \in lab_\alpha^M(t)$  iff  $\alpha_1 \in lab_\alpha^M(t')$  for some  $t' \in [min_{<}\{t, i\}, i + \pi[$ ;
- $\square\alpha_1 \in lab_\alpha^M(t)$  iff  $\alpha_1 \in lab_\alpha^M(t')$  for all  $t' \in [min_{<}\{t, i\}, i + \pi[$ ;
- $\heartsuit\alpha_1 \in lab_\alpha^M(t)$  iff  $\alpha_1 \in lab_\alpha^M(t')$  for some  $t' \in min_{\prec_M}(t)$ ;
- $\boxtimes\alpha_1 \in lab_\alpha^M(t)$  iff  $\alpha_1 \in lab_\alpha^M(t')$  for all  $t' \in min_{\prec_M}(t)$ .

► **Lemma 41.** Let a UPPI  $M = (i, \pi, V_M, \prec_M)$ ,  $\alpha \in \mathcal{L}^*$  and  $t \in \mathbb{N}$ ,  $I(M), 0 \models \alpha$  iff  $\alpha \in lab_\alpha^M(0)$ .

► **Proposition 42.** Let  $\alpha \in \mathcal{L}^*$ . We have that  $\alpha$  is  $\mathfrak{J}^{sd}$ -satisfiable iff there exists a UPPI  $M$  such that  $I(M), 0 \models \alpha$  and  $size(I(M)) \leq |\alpha| \times 2^{|\mathcal{P}|}$ .

Hence, to decide the satisfiability of a sentence  $\alpha \in \mathcal{L}^*$ , we can first guess a UPPI  $M$  bounded by  $|\alpha| \times 2^{|\mathcal{P}|}$ . Next, using the labelling function of  $M$ , we check the satisfiability of  $\alpha$  by the UPI  $I(M)$ .

► **Theorem 43.**  $\mathfrak{J}^{sd}$ -satisfiability problem for  $\mathcal{L}^*$  sentences is decidable.

## 7 Concluding remarks

In this paper, we have introduced  $LTL^\sim$ , a meaningful extension of linear temporal logic featuring defeasible temporal operators. These are given an intuitive semantics in terms of preferential temporal interpretations in which time points are ordered according to their likelihood (or normality). The main research question of the paper is the decidability of the resulting framework. Here we have defined the class of state-dependent interpretations  $\mathfrak{J}^{sd}$  and the fragment  $\mathcal{L}^*$ , and we have shown that  $\mathfrak{J}^{sd}$ -satisfiability in the referred fragment is a decidable problem.

We are aware that the upper bound established in this paper is intractable in practice. One of our immediate next steps is to tighten the complexity results for the class of state-dependent interpretations. We envisage two options: either the complexity remains the same, in which case we shall explore other well-behaved fragments of  $LTL^\sim$ ; or we show reasoning with  $\mathcal{L}^*$  remains in the same class of LTL, in which case we shall add defeasible counterparts to  $\circ$  and  $\mathcal{U}$  together with a notion of defeasible conditional *à la* KLM to our framework, thereby depicting a complete picture of defeasible model checking. In both cases, the results here established will prove useful.

An outstanding task in the study of preferential temporal reasoning is the definition of a sound and complete analytical tableau method for  $LTL^\sim$ . For that, we can benefit from the work of Giordano et al. [10] and Britz and Varzinczak [5, 6] in similarly-structured logics. Nevertheless, in the case of preferential LTL, the task is far from being an easy one. The first hurdle we need to overcome is in the definition of appropriate tableau rules for our defeasible operators  $\boxtimes$  and  $\heartsuit$ . Indeed, given their non-monotonic semantics, we cannot make use of a recursive rewriting similar to that in Wolper's rules [19] in order to get rid of nested classical modalities. To witness, we have  $\not\models \boxtimes\alpha \leftrightarrow \alpha \wedge \circ\boxtimes\alpha$  and  $\not\models \heartsuit\alpha \leftrightarrow \alpha \vee \circ\heartsuit\alpha$ .

## References

- 1 O. Arieli and A. Avron. General patterns for nonmonotonic reasoning: From basic entailments to plausible relations. *Logic Journal of the IGPL*, 8:119–148, 2000.
- 2 M. Ben-Ari. *Mathematical Logic for Computer Science, third edition*. Springer, 2012.
- 3 K. Britz, T. Meyer, and I. Varzinczak. Preferential reasoning for modal logics. *Electronic Notes in Theoretical Computer Science*, 278:55–69, 2011. Proceedings of the 7th Workshop on Methods for Modalities and the 4th Workshop on Logical Aspects of Multi-Agent Systems. doi:10.1016/j.entcs.2011.10.006.
- 4 K. Britz, T. Meyer, and I. Varzinczak. Semantic foundation for preferential description logics. In Dianhui Wang and Mark Reynolds, editors, *AI 2011: Advances in Artificial Intelligence*, pages 491–500, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- 5 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.
- 6 K. Britz and I. Varzinczak. Preferential tableaux for contextual defeasible  $\mathcal{ALC}$ . In S. Cerrito and A. Popescu, editors, *Proceedings of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, number 11714 in LNCS, pages 39–57. Springer, 2019.
- 7 D. M. Gabbay. Theoretical foundations for non-monotonic reasoning in expert systems. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 439–457, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.
- 8 D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *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.
- 9 L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato. Preferential description logics. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, number 4790 in LNAI, pages 257–272. Springer, 2007.
- 10 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.
- 11 R. Goré. Tableau methods for modal and temporal logics. In M. D’Agostino, D.M. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Kluwer Academic Publishers, 1999.
- 12 S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.
- 13 N. Laverny and J. Lang. From knowledge-based programs to graded belief-based programs, part i: On-line reasoning\*. *Synthese*, 147(2):277–321, November 2005. doi:10.1007/s11229-005-1350-1.
- 14 D. Makinson. *How to Go Nonmonotonic*, pages 175–278. Springer Netherlands, Dordrecht, 2005. doi:10.1007/1-4020-3092-4\_3.
- 15 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.
- 16 Y. Shoham. A semantical approach to nonmonotonic logics. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*, pages 275–279, 1987.
- 17 Y. Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, 1988.
- 18 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985. doi:10.1145/3828.3837.
- 19 P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.

## A Proofs of results in Section 3 and Section 4

► **Proposition 8.** *Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$  and let  $i, i', j, j' \in \mathbb{N}$  s.t.  $i \leq i'$ ,  $i' \leq j'$  and  $j \in \min_{\prec}(i)$ . If  $V(j) = V(j')$ , then  $j' \in \min_{\prec}(i')$ .*

**Proof.** Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$  and let  $i, j, i', j'$  be four time points s.t.  $i \leq i'$ ,  $i' \leq j'$  and  $j \in \min_{\prec}(i)$ . We assume that  $V(j) = V(j')$  and we suppose that  $j' \notin \min_{\prec}(i')$ . Following our supposition,  $j' \notin \min_{\prec}(i')$  means that there exists  $k \in [i', +\infty[$  where  $(k, j') \in \prec$ . From Definition 7, if  $(k, j') \in \prec$  and  $V(j) = V(j')$ , then  $(k, j) \in \prec$ . Since  $(k, j) \in \prec$ , we have  $j \notin \min_{\prec}(i)$ . This conflicts with our assumption of  $j \in \min_{\prec}(i)$ . We conclude that if  $V(j) = V(j')$  then  $j' \in \min_{\prec}(i')$ . ◀

► **Proposition 9.** *Let  $I = (V, \prec) \in \mathfrak{I}$  and let  $i, j \in \mathbb{N}$  s.t.  $j \in \min_{\prec}(i)$ . For all  $i \leq i' \leq j$ , we have  $j \in \min_{\prec}(i')$ .*

**Proof.** Let  $I = (V, \prec) \in \mathfrak{I}$  and let  $i, i', j \in \mathbb{N}$  s.t.  $j \in \min_{\prec}(i)$  and  $i \leq i' \leq j$ . Since  $j \in \min_{\prec}(i)$ , there is no  $j' \in [i, +\infty[$  s.t.  $(j', j) \in \prec$ . Moreover, we have  $i \leq i'$ , we conclude that there is no  $j' \in [i', +\infty[$  s.t.  $(j', j) \in \prec$ . Therefore, we have  $j \in \min_{\prec}(i')$ . ◀

► **Proposition 12.** *Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$  and let  $i \leq j \leq i' \leq j'$  be time points in  $\text{final}(I)$  s.t.  $V(j) = V(j')$ . Then we have  $j \in \min_{\prec}(i)$  iff  $j' \in \min_{\prec}(i')$ .*

**Proof.** Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$ . We have four time points  $i \leq j \leq i' \leq j' \in \text{final}(I)$ , this proof is divided in two parts:

- For the only-if part, we suppose that  $j \in \min_{\prec}(i)$  and we prove that  $j' \in \min_{\prec}(i')$ . We have  $i \leq i'$ ,  $i' \leq j'$ ,  $V(j) = V(j')$  and  $j \in \min_{\prec}(i)$ . Thanks to Proposition 8,  $j' \in \min_{\prec}(i')$ .
- For the if part, we suppose that  $j' \in \min_{\prec}(i')$  and we prove that  $j \in \min_{\prec}(i)$ . We use a proof by contradiction. We assume that  $j' \in \min_{\prec}(i')$  and we suppose that  $j \notin \min_{\prec}(i)$ . This implies that there exists  $k \in [i, +\infty[$  such that  $(k, j) \in \prec$ .
  - Case 1:  $k \in [i', +\infty[$ . From Definition 7, since  $V(j) = V(j')$  and  $(k, j) \in \prec$ , then  $(k, j') \in \prec$  thus  $j' \notin \min_{\prec}(i')$ . This conflicts with our assumption that  $j' \in \min_{\prec}(i')$ .
  - Case 2:  $k \in [i, i'[$ . From Lemma 10, since  $k \in \text{final}(I)$ , then there exists  $k' \in [i', +\infty[$  such that  $V(k') = V(k)$ . From Definition 7, since we have  $V(j') = V(j)$ ,  $V(k') = V(k)$  and  $(k, j) \in \prec$ , then  $(k', j') \in \prec$ , thus  $j' \notin \min_{\prec}(i')$ . This conflicts with our assumption that  $j' \in \min_{\prec}(i')$ . ◀

► **Lemma 13.** *Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$  and  $i \leq i'$  be time points of  $\text{final}(I)$  where  $V(i) = V(i')$ . Then for every  $\alpha \in \mathcal{L}^*$ , we have  $I, i \models \alpha$  iff  $I, i' \models \alpha$ .*

**Proof.** Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$  and  $i \leq i'$  in  $\text{final}(I)$  such that  $V(i) = V(i')$ . We prove that  $I, i \models \alpha$  iff  $I, i' \models \alpha$  using structural induction on  $\alpha$ .

- Base:  $\alpha$  is an atomic proposition  $p$ . For the only-if part, we know that  $I, i \models p$  iff  $p \in V(i)$ . Since  $V(i) = V(i')$ , we have  $p \in V(i')$ , thus  $I, i' \models p$ . Same reasoning applies for the if part.
- $\alpha = \diamond \alpha_1$ . For the only-if part, we assume that  $I, i \models \diamond \alpha_1$ . Following our assumption,  $I, i \models \diamond \alpha_1$  means that there exists  $j \in [i, +\infty[$  s.t.  $j \in \min_{\prec}(i)$  and  $I, j \models \alpha_1$ . Thanks to Lemma 10. Since  $j \in \text{final}(I)$ , there exists  $j' \in [i', +\infty[$  such that  $V(j') = V(j)$ . Thanks to the induction hypothesis, if  $V(j) = V(j')$  and  $I, j \models \alpha_1$  then (I)  $I, j' \models \alpha_1$ . Thanks to Proposition 8,  $V(j) = V(j')$ ,  $i \leq i'$ ,  $i' \leq j'$  and  $j \in \min_{\prec}(i)$  means that (II)  $j' \in \min_{\prec}(i')$ . From (I) and (II), we conclude that  $I, i' \models \diamond \alpha_1$ .

For the if part, we assume that  $I, i' \models \diamond\alpha_1$ .  $I, i' \models \diamond\alpha_1$  means that there is a  $j' \in [i', +\infty[$  such that  $j' \in \min_{\prec}(i')$  and (I)  $I, j' \models \alpha_1$ . We need to prove that  $j' \in \min_{\prec}(i)$ . We suppose that  $j' \notin \min_{\prec}(i)$ . It means that there exists  $k \in [i, +\infty[$  such that  $(k, j') \in \prec$ . From Lemma 10, since  $k \in \text{final}(I)$ , that means there is  $k' \in [i', +\infty[$  such that  $V(k) = V(k')$ . Following the condition set in Definition 7, since  $(k, j') \in \prec$  and  $V(k') = V(k)$ , then  $(k', j') \in \prec$  and thus  $j' \notin \min_{\prec}(i')$ , conflicting with our assumption of  $j' \in \min_{\prec}(i')$ , thus (II)  $j' \in \min_{\prec}(i)$ . From (I) and (II), we conclude that  $I, i \models \diamond\alpha$ . ◀

## B Proofs of results in Section 5

► **Lemma 27.** *Let  $\alpha_1 \in \mathcal{L}^*$  be a sentence starting with a temporal operator,  $I = (V, \prec) \in \mathfrak{J}^{sd}$  and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  where for all  $t \in T$  we have  $I, t \models \diamond\alpha_1$ . Then for all  $t, t' \in \text{Anchors}(I, T, \diamond\alpha_1)$  s.t.  $V(t) = V(t')$  and  $t \neq t'$ , we have  $t, t' \in \text{final}(I, \text{Anchors}(I, T, \diamond\alpha_1))$ .*

**Proof.** Let  $\alpha_1 \in \mathcal{L}^*$ , let  $T$  be a non-empty acceptable sequence w.r.t.  $I \in \mathfrak{J}^{sd}$  where for all  $t \in T$  we have  $I, t \models \diamond\alpha_1$ . Just as a reminder, we have  $\text{Anchors}(I, T, \diamond\alpha_1) = \bigcup_{t_i \in T} ST(I, AS(I, \min_{\prec}(t_i), \alpha_1))$ . Thus, there exists  $t_i \in T$  such that  $t \in ST(I, AS(I, \min_{\prec}(t_i), \alpha_1))$ . Suppose that there exist  $t, t' \in \text{Anchors}(I, T, \diamond\alpha_1)$  with  $t \neq t'$  such that  $t$  is in  $\text{init}(I, \text{Anchors}(I, T, \diamond\alpha_1))$  and  $V(t) = V(t')$ . Notice that  $t \in \text{init}(I)$ , since  $t \in \text{init}(I, \text{Anchors}(I, T, \diamond\alpha_1))$ . Without loss of generality, we assume that  $t < t'$ . From Definition 24, we have  $t \in AS(I, \mathbf{t}_{\alpha_1}^{I, AS(I, \min_{\prec}(t_i))})$ . Thanks to Definition 22 and Definition 23, the fact that  $t' \in \text{init}(I)$ , we can see that : (1) there is no  $t'' \in \text{final}(I, AS(I, \min_{\prec}(t_i)))$  s.t.  $I, t'' \models \alpha_1$  and (2)  $t = \mathbf{t}_{\alpha_1}^{I, AS(I, \min_{\prec}(t_i))} = \max_{\prec} \{t'' \in \text{init}(I, AS(I, \min_{\prec}(t_i))) \mid I, t'' \models \alpha_1\}$ . On the other hand, thanks to Proposition 8, since  $t' < t''$  and  $t' \in \min_{\prec}(t_i)$ , we have  $t'' \in \min_{\prec}(t_i)$ . Hence  $t'' \in AS(I, \min_{\prec}(t_i))$ . Since  $t'' \in \text{Anchors}(I, T, \diamond\alpha_1)$ , we also have  $I, t'' \models \alpha_1$ . From this and the property (1) we can assert that  $t''$  does not belong to  $\text{final}(I, AS(I, \min_{\prec}(t_i)))$ . It follows that  $t'' \in \text{init}(I, AS(I, \min_{\prec}(t_i)))$ . From the property (2) we can assert that  $t \geq t''$ , which leads to a contradiction since  $t < t'$ . Therefore, for all  $t, t' \in \text{Anchors}(I, T, \diamond\alpha_1)$  s.t.  $V(t) = V(t')$ , we must have  $t, t' \in \text{final}(\text{Anchors}(I, T, \diamond\alpha_1))$ . ◀

► **Proposition 28.** *Let  $\alpha \in \mathcal{L}^*$  be a sentence starting with a temporal operator,  $I = (V, \prec) \in \mathfrak{J}^{sd}$ . Let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  where for all  $t \in T$  we have  $I, t \models \alpha$ . Then, we have  $\text{size}(I, \text{Anchors}(I, T, \alpha)) \leq 2^{|\mathcal{P}|}$ .*

**Proof.** Let  $I = (V, \prec) \in \mathfrak{J}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . We show that is the case for our temporal operators:

- $T$  is an acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \diamond\alpha_1$ . From Proposition 27, for all  $t'_i, t'_j \in \text{Anchors}(I, T, \diamond\alpha_1)$  s.t.  $V(t'_i) = V(t'_j)$  we have  $t'_i, t'_j \in \text{final}(I, \text{Anchors}(I, T, \diamond\alpha_1))$ . From Proposition 20, we can conclude that  $\text{size}(\text{Anchors}(I, T, \diamond\alpha_1)) \leq 2^{|\mathcal{P}|}$ .
- Going back to Definition 26, we have  $\text{Anchors}(I, T, \boxtimes\alpha_1) = DR(I, \bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i)))$ . We denote the acceptable sequence  $\bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i))$  by  $N$ . From Definition 25 we have  $\text{Anchors}(I, T, \boxtimes\alpha_1) = DR(I, N) = \bigcup_{v \in \text{val}(I, N)} ST(I, N, \alpha_v)$ . Moreover, we know that  $\text{size}(ST(I, N, \alpha_v)) = 1$  for all  $v \in \text{val}(I, N)$ . Consequently, thanks to Proposition 19, we have  $\text{size}(\bigcup_{v \in \text{val}(I, N)} ST(I, N, \alpha_v)) \leq \text{card}(\text{val}(I, N))$ . We can see that  $\text{card}(\text{val}(I, N)) \leq 2^{|\mathcal{P}|}$ , we can conclude that  $\text{size}(\text{Anchors}(I, T, \boxtimes\alpha_1)) = \text{size}(\bigcup_{v \in \text{val}(I, N)} ST(I, N, \alpha_v)) \leq 2^{|\mathcal{P}|}$ . ◀

► **Proposition 29.** *Let  $\alpha_1 \in \mathcal{L}^*$ ,  $I = (V, \prec) \in \mathfrak{I}^{sd}$ , let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \Box\alpha_1$ , with  $\alpha_1 \in \mathcal{L}^*$ . For all acceptable sequences  $N$  w.r.t.  $I$  s.t.  $\text{Anchors}(I, T, \Box\alpha_1) \subseteq N$  and for all  $t_i \in N \cap T$ , we have the following: Let  $I^N = (V^N, \prec^N)$  be the pseudo-interpretation over  $N$  and  $t' \in N$ , if  $t' \notin \min_{\prec}(t_i)$ , then  $t' \notin \min_{\prec^N}(t_i)$ .*

**Proof.** Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$ , let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \Box\alpha_1$ , with  $\alpha_1 \in \mathcal{L}^*$ . Let  $N$  be an acceptable sequence w.r.t.  $I$  s.t.  $\text{Anchors}(I, T, \Box\alpha_1) \subseteq N$ . Let  $t_i \in N \cap T$ . Let  $t' \in N$  be a time point s.t.  $t' \notin \min_{\prec}(t_i)$ , we discuss these two cases:

- $t' \notin [t_i, +\infty[$ : Since  $t' \notin [t_i, +\infty[$ , then  $t' \notin [t_i, +\infty[ \cap N$ . Therefore, we conclude that  $t' \notin \min_{\prec^N}(t_i)$ .
- $t' \in [t_i, +\infty[$ : Since  $\prec$  satisfies the well-foundedness condition,  $t' \notin \min_{\prec}(t_i)$  implies that there exists a time point  $t'' \in \min_{\prec}(t_i)$  s.t.  $(t'', t') \in \prec$ . Let  $\alpha_{t''}$  be the representative sentence of  $V(t'')$ . For the sake of readability, we shall denote the sequence  $\bigcup_{t \in T} AS(I, \min_{\prec}(t))$  with  $M$ . Notice that there exists  $V \in \text{val}(I, M)$  such that  $V = V(t'')$  since  $t_i \in T$  and  $t'' \in \min_{\prec}(t_i)$ . Thanks to Definition 25, since  $DR(I, M) = \bigcup_{v \in \text{val}(I, M)} ST(I, M, \alpha_v)$  and  $V(t'') \in \text{val}(I, M)$ , we can find  $t''' \in ST(I, M, \alpha_{t''})$  where  $t''' \in DR(I, M) \subseteq N$ ,  $V(t''') = V$  and  $t''' \geq t''$ . Since  $(t'', t') \in \prec$ ,  $I \in \mathfrak{I}^{sd}$  and  $V(t''') = V(t'')$ , we have  $(t''', t') \in \prec$ . Moreover, we have  $t''', t' \in N$ , and therefore  $(t''', t') \in \prec^N$ . Since  $t''' \in [t_i, +\infty[ \cap N$  and  $(t''', t') \in \prec^N$ , we conclude that  $t' \notin \min_{\prec^N}(t_i)$ . ◀

► **Proposition 31.** *Let  $\alpha \in \mathcal{L}^*$  be in NNF,  $I = (V, \prec) \in \mathfrak{I}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . Then, we have  $\text{size}(I, \text{Keep}(I, T, \alpha)) \leq \mu(\alpha) \times 2^{|\mathcal{P}|}$ .*

**Proof.** Let  $I = (V, \prec) \in \mathfrak{I}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$  which  $\alpha \in \mathcal{L}^*$ .

We use structural induction on  $T$  and  $\alpha$  in order to prove this property.

- Base  $\alpha = p$  or  $\alpha = \neg p$ .  $\text{Keep}(I, T, \alpha) = \emptyset$ . Since  $\text{size}(I, \emptyset) = 0 \leq \mu(\alpha) \times 2^{|\mathcal{P}|} = 0$ , then the property holds on atomic propositions.
- $\alpha = \Diamond\alpha_1$ . First of all, we proved in Proposition 28 that (I)  $\text{size}(I, \text{Anchors}(I, T, \Diamond\alpha_1)) \leq 2^{|\mathcal{P}|}$ . On the other hand, thanks to Definition 26 it is easy to see that  $\text{Anchors}(I, T, \Diamond\alpha_1)$  is a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t' \in \text{Anchors}(I, T, \Diamond\alpha_1)$  we have  $I, t' \models \alpha_1$ . By the induction hypothesis on  $\text{Anchors}(I, T, \Diamond\alpha_1)$  and  $\alpha_1$ , we have (II)  $\text{size}(I, \text{Keep}(I, \text{Anchors}(I, T, \Diamond\alpha_1), \alpha_1)) \leq \mu(\alpha_1) \times 2^{|\mathcal{P}|}$ . Thanks to Proposition 19, from (I) and (II), we conclude that  $\text{size}(I, \text{Keep}(I, T, \Diamond\alpha_1)) \leq (1 + \mu(\alpha_1)) \times 2^{|\mathcal{P}|} = \mu(\Diamond\alpha_1) \times 2^{|\mathcal{P}|}$ .
- $\alpha = \Box\alpha_1$ . First of all, we proved in Proposition 28 that (I)  $\text{size}(I, \text{Anchors}(I, T, \Box\alpha_1)) \leq 2^{|\mathcal{P}|}$ . On the other hand, from definition30, we have  $T' = \bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i))$ . It is easy to see that for all  $t' \in T'$  we have  $I, t' \models \alpha_1$  and that  $T'$  is a non-empty acceptable sequence w.r.t.  $I$ . By the induction hypothesis on  $T'$  and  $\alpha_1$ , we have (II)  $\text{size}(I, \text{Keep}(I, T', \alpha_1)) \leq \mu(\alpha_1) \times 2^{|\mathcal{P}|}$ . Thanks to Proposition 19, from (I) and (II) we conclude that  $\text{size}(I, \text{Keep}(I, T, \Box\alpha_1)) \leq (1 + \mu(\alpha_1)) \times 2^{|\mathcal{P}|} = \mu(\Box\alpha_1) \times 2^{|\mathcal{P}|}$ . ◀

► **Lemma 32.** *Let  $\alpha \in \mathcal{L}^*$  be in NNF,  $I = (V, \prec) \in \mathfrak{I}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . For all acceptable sequences  $N$  w.r.t.  $I$ , if  $\text{Keep}(I, T, \alpha) \subseteq N$ , then for every  $t \in N \cap T$ , we have  $I^N, t \models \alpha$ .*



## 19:18 On the Decidability of a Fragment of preferential LTL

**Proof.** Let  $\alpha \in \mathcal{L}^*$  be in NNF,  $I = (V, \prec) \in \mathfrak{J}^{sd}$ , and let  $T$  be a non-empty acceptable sequence w.r.t.  $I$  s.t. for all  $t \in T$  we have  $I, t \models \alpha$ . We consider  $N$  to be an acceptable sequence w.r.t.  $I$  s.t.  $Keep(I, T, \alpha) \subseteq N$  and  $t \in N \cap T$ . Let  $I^N = (N, V^N, \prec^N)$  be the pseudo-interpretation over  $N$ .

We use structural induction on  $T$  and  $\alpha$  in order to prove this property.

- $\alpha = p$  or  $\alpha = \neg p$ . Since  $I, t \models p$  (resp.  $\neg p$ ), it means that  $p \in V(t)$  (resp.  $p \notin V(t)$ ). We know that  $V^N(t) = V(t)$ . We conclude that  $I^N, t \models_{\mathcal{P}} p$  (resp.  $\neg p$ ).
- $\alpha = \diamond\alpha_1$ . We have  $I, t \models \diamond\alpha_1$  and we need to prove that  $I^N, t \models_{\mathcal{P}} \diamond\alpha_1$ .  $I, t \models \diamond\alpha_1$  means that there exists  $t' \in \min_{\prec}(t)$  such that  $I, t' \models \alpha_1$ , therefore  $anchors(I, T, \diamond\alpha_1)$  is non-empty (see Definition 26). We know that  $anchors(I, T, \diamond\alpha_1) \subseteq Keep(I, T, \diamond\alpha_1) \subseteq N$ , consequently  $anchors(I, T, \diamond\alpha_1) \cap N$  is non-empty. Thanks to Definition 26 it is easy to see that for all  $t_1 \in anchors(I, T, \diamond\alpha_1)$  we have  $I, t_1 \models \alpha_1$ . By the induction hypothesis on  $anchors(I, T, \diamond\alpha_1)$  and  $\alpha_1$ , since  $Keep(I, T_1, \alpha_1) \subseteq N$  with  $T_1 = anchors(I, T, \diamond\alpha_1)$ , and  $T_1$  is an acceptable sequence where  $I, t' \models \alpha_1$  for all  $t' \in T_1$ , we conclude that  $I^N, t' \models_{\mathcal{P}} \alpha_1$  (I). Thanks to the construction of the pseudo-interpretation  $I^N$ , since  $t' \in \min_{\prec}(t)$ , therefore  $t' \in \min_{\prec^N}(t)$  (II). From (I) and (II), we conclude that  $I^N, t \models_{\mathcal{P}} \diamond\alpha_1$ .
- $\alpha = \boxtimes\alpha_1$ . We have  $I, t \models \boxtimes\alpha_1$  and we need to prove that  $I^N, t \models_{\mathcal{P}} \boxtimes\alpha_1$ .  $I, t \models \boxtimes\alpha_1$  means that for all  $t' \in \min_{\prec}(t)$  we have  $I, t' \models \alpha_1$ , therefore for all  $t' \in T' = \bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i))$  we have  $I, t' \models \alpha_1$ . In addition, thanks to the well-foundedness condition on  $\prec$ ,  $T'$  is non-empty. We know that  $anchors(I, T, \boxtimes\alpha_1) \subseteq Keep(I, T, \boxtimes\alpha_1) \subseteq N$  and that  $anchors(I, T, \boxtimes\alpha_1) = DR(I, T')$  consequently  $T' \cap N$  is non-empty. We use proof by contradiction. Suppose that  $I^N, t \not\models_{\mathcal{P}} \boxtimes\alpha_1$ , which means there exists  $t' \in \min_{\prec^N}(t)$  s.t.  $I^N, t' \not\models_{\mathcal{P}} \alpha_1$ . Thanks to Proposition 29, if  $t' \in \min_{\prec^N}(t)$ , then  $t' \in \min_{\prec}(t)$ . Just a reminder, we have  $T' = \bigcup_{t_i \in T} AS(I, \min_{\prec}(t_i))$  where for all  $t'' \in T'$  we have  $I, t'' \models \alpha_1$  (Note that  $T'$  is a non-empty acceptable sequence w.r.t.  $I$ ). By the induction hypothesis on  $T'$  and  $\alpha_1$ , since  $Keep(I, T', \alpha_1) \subseteq N$ , and  $t' \in AS(I, \min_{\prec}(t)) \subseteq T'$ , therefore  $I^N, t' \models_{\mathcal{P}} \alpha_1$ . This conflicts with our supposition. We conclude that there is no  $t' \in \min_{\prec^N}(t)$  s.t.  $I^N, t' \not\models_{\mathcal{P}} \alpha_1$ , and therefore  $I^N, t \models_{\mathcal{P}} \boxtimes\alpha_1$ . ◀

## C Proof of results in Section 6

**NB:** The results marked (\*) are introduced here, while they are omitted in the main text.

► **Proposition 39.** Let  $M = (i, \pi, V_M, \prec_M)$  be a UPPI,  $I(M) = (V, \prec)$  and  $t, t', t_M, t'_M \in \mathbb{N}$  s.t.:

$$t_M = \begin{cases} t, & \text{if } t < i; \\ i + (t - i) \bmod \pi, & \text{otherwise.} \end{cases} \quad t'_M = \begin{cases} t', & \text{if } t' < i; \\ i + (t' - i) \bmod \pi, & \text{otherwise.} \end{cases}$$

We have the following:  $t' \in \min_{\prec}(t)$  iff  $t'_M \in \min_{\prec_M}(t_M)$ .

**Proof.** Let  $M = (i, \pi, V_M, \prec_M)$  be a UPPI,  $I(M) = (V, \prec)$  and  $t, t' \in \mathbb{N}$ .

- For the only-if part, we assume that  $t' \in \min_{\prec}(t)$ . Following our assumption, there is no  $t'' \in [t, +\infty[$  s.t.  $(t'', t') \in \prec$ . We use a proof by contradiction. Suppose that  $t'_M \notin \min_{\prec_M}(t_M)$ , which means there exists  $t''_M \in [\min_{\prec}\{t_M, i\}, i + \pi[$  with  $(V_M(t''_M), V_M(t'_M)) \in \prec_M$ . Going back to Definition 37,  $V_M(t''_M) = V(t')$  and  $V_M(t'_M) = V(t)$ . Consequently,  $(V(t'_M), V_M(t')) \in \prec_M$ . Thanks to Definition 37, (I)  $(t''_M, t') \in \prec$ . There are two possible cases for  $t, t'$ . If  $t \in [0, i[$  then  $t_M = t$  and (II)  $t''_M \in [t, i + \pi[$ . From (I) and (II), there exists  $t''_M > t$  such that  $(t''_M, t') \in \prec$ . This conflicts with our supposition. If  $t \in [i, +\infty[$ , then  $t''_M \in [i, i + \pi[$



and  $t, t', t''$  are in  $\text{final}(I(M))$ . Thanks to proposition 10, there exists  $t'' > t$  such that  $V(t'') = V(t_M)$ . Since  $I(M) \in \mathfrak{J}^{sd}$  and  $(t''_M, t') \in \prec$  then  $(t'', t) \in \prec$ . Consequently, there exists  $t'' > t$  such that  $(t'', t) \in \prec$ . This conflicts with our supposition.

- For the if part, we assume that  $t'_M \in \text{min}_{\prec_M}(t_M)$ . Following our assumption, there is no  $t''_M \in [\text{min}_{\prec}\{t_M, i\}, i + \pi[$  with  $(V_M(t''_M), V_M(t'_M)) \in \prec_M$ . We use proof by contradiction. Suppose that  $t' \notin \text{min}_{\prec}(t)$ , which means there exists  $t''' > t$  such that  $(t''', t') \in \prec$ . Let  $t'''_M$  be defined as follows:

$$t'''_M = \begin{cases} t''', & \text{if } t''' < i; \\ i + (t''' - i) \bmod \pi, & \text{otherwise.} \end{cases}$$

Thanks to definition 37,  $V(t''') = V_M(t'''_M)$ ,  $V(t') = V_M(t'_M)$  and since  $(t''', t') \in \prec$  then  $(V(t'''), V(t')) \in \prec_M$ . Consequently, (I)  $(V(t'''), V(t'_M)) \in \prec_M$ . . From (I) and (II), we have  $t'_M \notin \text{min}_{\prec_M}(t_M)$ . This conflicts with our supposition. ◀

► **Proposition 42.** *Let  $\alpha \in \mathcal{L}^*$ . We have that  $\alpha$  is  $\mathfrak{J}^{sd}$ -satisfiable iff there exists a UPPI  $M$  such that  $\mathfrak{l}(M), 0 \models \alpha$  and  $\text{size}(\mathfrak{l}(M)) \leq |\alpha| \times 2^{|\mathcal{P}|}$ .*

**Proof.** Let  $\alpha \in \mathcal{L}^*$ .

- For the only if part, let  $\alpha$  be  $\mathfrak{J}^{sd}$ -satisfiable. Thanks to Theorem 21 and Proposition 35, there exists a UPI  $I = (V, \prec) \in \mathfrak{J}^{sd}$  s.t.  $I, 0 \models \alpha$  and  $\text{size}(I) \leq |\alpha| \times 2^{|\mathcal{P}|}$ . We define the UPPI  $M(I)$  from  $I$ . It can be checked that  $\mathfrak{l}(M(I)) = I$ . Therefore, from  $\mathfrak{J}^{sd}$ -satisfiable sentence  $\alpha$ , we can find a UPPI  $M$  such that  $\mathfrak{l}(M), 0 \models \alpha$  and  $\text{size}(\mathfrak{l}(M)) \leq |\alpha| \times 2^{|\mathcal{P}|}$ .
- For the if part, let  $M = (i, \pi, V_M, \prec_M)$  be a UPPI s.t.  $\mathfrak{l}(M), 0 \models \alpha$ . Since  $\mathfrak{l}(M) \in \mathfrak{J}^{sd}$ , therefore  $\alpha$  is  $\mathfrak{J}^{sd}$ -satisfiable. ◀

